НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“ ” _____ _____ 2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою “Комп’ютерні системи та мережі”
спеціальності 123 “Комп’ютерна інженерія”

на тему: Система обліку працівників виробничого підприємства

Виконав: студент 4 курсу, групи ІО-01
(шифр групи)

Луценко Владислав Вікторович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник ст. вик. Сімоненко Андрій Валерійович _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) ст. вик. Виноградов Ю.М. _____
(назва розділу) (посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2024 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти: перший (бакалавр)

Освітньо-професійна програма: “Комп’ютерні системи та мережі”

Спеціальність: 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ:

Завідувач кафедри

_____ Сергій СТІРЕНКО
(підпис)

“ ” _____ _____ 2024 р.

ЗАВДАННЯ

на бакалаврський дипломний проєкт студента

Луценко Владислав Вікторович

1. Тема проєкту: “Система обліку працівників виробничого підприємства”
   Керівник проєкту старший викладач Сімоненко Андрій Валерійович,
   Затверджені наказом по університету від 27 травня 2024 року № 2112-с
2. Термін здачі студентом закінченого проекту 11 червня 2024 р.
3. Вихідні дані до проєкту: технічна документація, теоретичні дані.
4. Зміст розрахунково-пояснювальної записки:
   Аналіз предметної області
   Архітектура та проєктування системи. Вибір технологій розробки
   Розробка системи

5. Перелік графічного матеріалу: структурна схема системи (схема структурна), діаграма даних (схема функціональна), схема алгоритму (схема принципова).
6. Консультанти розділів проєкту:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Виноградов Ю.М.		

7. Дата видачі завдання “25” грудня 2023 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	Затвердження теми проєкту	25.12.2023	
2.	Вивчення та аналіз даних	26.12.2023 - 12.02.2024	
3.	Розробка архітектури та загальної структури системи	12.02.2024 - 04.03.2024	
4.	Розробка структур окремих підсистем	04.03.2024 - 18.03.2024	
5.	Програмна реалізація системи	18.03.2024 - 15.04.2024	
6.	Оформлення пояснювальної записки	15.04.2024 - 20.05.2024	
7.	Захист програмного продукту	24.05.2024	
8.	Передзахист	3.06.2024	
9.	Захист	18.06.2024	

Студент-дипломник: _____ ЛУЦЕНКО Владислав
(підпис) (ім'я, прізвище)

Керівник проєкту: _____ СІМОНЕНКО Андрій
(підпис) (ім'я, прізвище)

АНОТАЦІЯ

Бакалаврський дипломний проєкт присвячений розробці системи для обліку працівників виробничого підприємства. Метою даного проєкту є створення зручного програмного інструменту для автоматизації процесів обліку персоналу на підприємстві.

В ході роботи було проаналізовано існуючі рішення в сфері обліку, розроблено гнучку систему, яка відповідає визначеним завданням обліку та потребам користувачів. Для розробки додатку використану платформу .NET та мову C#, фреймворк ASP.NET. Для реалізації бази даних обрано Microsoft SQL Server, а для її взаємодії з додатком використано Entity Framework. Такий вибір технології є поширеним та затребуваним, утворюючи одну екосистему.

Як результат, отримуємо веб-додаток з реалізацією необхідного функціоналу, який можна легко і швидко масштабувати та оновлювати.

Ключові слова: веб-додаток, облік працівників, .NET, C#, ASP.NET Core, база даних, SQL Server.

ANNOTATION

The bachelor's diploma project is dedicated to developing an employee accounting system for a manufacturing enterprise. This project aims to create a convenient software tool for automating personnel accounting processes at the enterprise.

In the course of the work, the existing solutions in the field of accounting were analyzed to develop a flexible system that will meet the defined accounting tasks and user needs. The application was developed using the .NET platform, C# language, and the ASP.NET framework. Microsoft SQL Server was chosen to implement the database, while the Entity Framework was used to interact with the application. This choice of technology is widely used and in demand, forming a single ecosystem.

As a result, we get a web application that implements the necessary functionality, which can be easily and quickly scaled and updated.

Keywords: web application, employee accounting, .NET, C#, ASP.NET Core, database, SQL Server.

Довідки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпляра	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	*ІАЛЦ.467200.002 ТЗ*	Система обліку	3		
			працівників виробничого			
			підприємства			
			Технічне завдання			
	А4	*ІАЛЦ.467200.003 ПЗ*	Система обліку	61		
			працівників виробничого			
			підприємства			
			Пояснювальна записка			
	А4	*ІАЛЦ.467200.004 Д1*	Система обліку	1		
			працівників виробничого			
			підприємства			
			Структурна схема системи			
			(схема структурна)			
	А4	*ІАЛЦ.467200.005 Д2*	Система обліку	1		
			працівників виробничого			
			підприємства			
			Діаграма даних			
			(схема функціональна)			
	А4	*ІАЛЦ.467200.006 Д3*	Система обліку	1		
			працівників виробничого			
			підприємства			
			Схема алгоритму			
			(схема принципова)			
	А4	*ІАЛЦ.467200.007 Д4*	Система обліку	14		
			працівників виробничого			
			підприємства			
			Текст коду програми			

Зм	Лист	№ докум.	Підп.	Дата	***ІАЛЦ.467200.001 ОА***		
Розроб.		Луценко В.В.			*Система обліку працівників виробничого підприємства* Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Сімоненко А.В.					
Н. контр.		Виноградов Ю.М.				***КПІ ім. Ігоря Сікорського, ФІОТ, гр. ІО-01***	

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЕКТУ

на тему: Система обліку працівників виробничого підприємства

Київ – 2024

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Технічне завдання видано на розробку додатку для обліку працівників виробничого підприємства.

Областю застосування цієї системи є сфера управління людськими ресурсами.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», який був затверджений факультетом «Інформатики та обчислювальної техніки», кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даної роботи є створення корисного та гнучкого додатку для автоматизації процесів обліку працівників на підприємстві. Розробка призначена спростити роботу працівників відділу кадрів, надаючи їм зручний інструмент для цього.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки проекту виступає технічна документація, публікації та статті в мережі Інтернет, науково-технічна література зі сфери інформаційних технологій.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблений додаток має надавати можливість зберігати та опрацьовувати інформацію про працівників, робочий процес, нещасні випадки. Реалізація системи автентифікації та авторизації для користувачів з розділенням доступу. Моніторинг змін в системі, простий та зрозумілий інтерфейс.

5.2. Вимоги до програмного забезпечення

— ОС Windows 10, Mac або Linux

— Наявність веб-браузеру

5.3. Вимоги до апаратної частини

— ЦП з тактовою частотою не менше 2 ГГц

— RAM не менше ніж 8 ГБ

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	25.12.2023
Вивчення та аналіз завдання	26.12.2023 - 12.02.2024
Розробка архітектури та загальної структури системи	12.02.2024 - 04.03.2024
Розробка окремих частин системи	04.03.2024 - 18.03.2024
Програмна реалізація системи	18.03.2024 - 15.04.2024
Виправлення помилок	15.04.2024 - 17.05.2024
Оформлення пояснювальної записки	20.05.2024

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

На тему: Система обліку працівників виробничого підприємства

Київ – 2024

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

HR	Human Resources
HCM	Human Capital Management
SaaS	Software-as-a-service
ОС	Операційна система
UI	User Interface
ASP.NET	Active Server Pages Network Enabled Technologies
MVC	Model-View-Controller
SQL	Structured Query Language
ORM	Object-Relational Mapping
DTO	Data Transfer Object
HTTP	HyperText Transfer Protocol
CSS	Cascading Style Sheets
HTML	HyperText Markup Language

ВСТУП

Будь-яким підприємствам для здійснення своєї діяльності необхідно обробляти велику кількість різноманітної інформації, пов'язаною з окремим аспектами їх функціонування. Одним із головних елементів, який потребує управління, є облік працівників. Без збирання, зберігання і обробки даних про робочий персонал підприємства, воно просто не зможе працювати. Традиційні методи ведення обліку, як-то паперові журнали чи прості електронні таблиці, не забезпечують необхідної швидкості та якості управлінням даними і є просто застарілими. За допомогою них складно проводити аналітику, яка мала б сприяти покращенню ефективності роботи підприємства. Тому постає необхідність у створенні програмних засобів, які б могли автоматизувати процеси обліку кадрів.

Особливо важливим є створення програмного забезпечення, що буде враховувати спеціалізовані потреби саме виробничих підприємств. Через зміни у світових процесах, все більше зростає роль таких підприємств, адже вони забезпечують основні потреби економіки країни та слугують її опорою.

Сучасний ринок подібних програмних рішень продовжує активно зростати [1]. І хоч він і може запропонувати немало варіантів для виконання такої задачі, завжди існує потреба в окремих спеціалізованих інструментах, які водночас могли б запропонувати необхідний рівень пластичності та комфорту. Світова нестабільність вимагає від підприємств вміти швидко пристосовуватись до змін, через що і програмні засоби мають бути готовими до таких викликів.

Метою цієї дипломної роботи є створення веб-додатку, що міг би запропонувати гнучке і зручне рішення по обліку працівників на виробничому підприємстві. Такий веб-додаток має полегшити роботу працівників відділу кадрів, оптимізувати процеси обліку, що сприятиме загальному покращенню продуктивності роботи підприємства.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальні положення

Для обліку працівників або, інакше кажучи, управління людськими ресурсами підприємства (як і будь-якої іншої організації) зазвичай існує окремий структурний підрозділ — відділ кадрів, або HR відділ. Окрім безпосередньої задачі роботи з інформацією про персонал, цей відділ здійснює нагляд за різними аспектами працевлаштування, як-от дотримання трудового законодавства та стандартів зайнятості, проведення співбесід та відбір кандидатів, управління ефективністю роботи, адміністрування виплат, упорядкування документів, а також деякими іншими аспектами рекрутингового процесу і звільнення працівників [2]. Також варто зазначити, що оскільки відділ кадрів працює з усіма працівниками в організації, він має доступ до всієї організаційної структури підприємства. Відповідно, можемо зробити висновок, що HR відділ виступає певною сполучною ланкою між керівництвом організації та її співробітниками. Це вчергове підкреслює важливість ефективної роботи такого відділу для правильного функціонування всієї організації.

Все вищенаведене про HR відділ і його обов'язки актуальне і для виробничих підприємств. Однак, через особливості своєї діяльності, виробничі підприємства мають свою специфіку, яка накладає певний відбиток і на роботу відділу кадрів. На виробничому підприємстві для виготовлення продукції використовується велика кількість обладнання підвищеної небезпеки. Порушення техніки безпеки при роботі з таким устаткуванням може призвести до серйозних травм, тому проведення інструктажів з техніки безпеки і облік подібних випадків є дуже важливим для забезпечення безпеки праці. Іншим важливим аспектом є кваліфікація працівників. Для роботи зі

спеціалізованим обладнанням потрібні працівники, що пройшли відповідне навчання і є сертифікованими для роботи з ним. Тому облік кваліфікації і проведення відповідних навчань також є необхідним. Окремо потрібно відзначити необхідність обліку робочого часу працівників. На окремих підприємствах, де налагоджено конвеєрне виробництво або існує потреба в безперебійних поставках продукції, необхідно чітко планувати робочі зміни для забезпечення відсутності простоїв і максимальної результативності виробництва.

1.2 Визначення завдань обліку та опис додатку

Ознайомившись із загальною інформацією про роботу HR відділу, перейдемо до більш детального розгляду того, що повинно обліковуватись та як наш додаток має це забезпечувати.

Найперше, робітнику відділу кадрів потрібно працювати з інформацією про працівника. Він повинен мати паспортні дані працівників, такі як ПІБ, номер картки платника податків та інші, для їх оформлення згідно вимог законодавства. Необхідно знати посаду працівника та час прийому на роботу, для нарахування заробітної плати відповідно стажу; відділ, в якому той працює, інформацію про відпустки, лікарняні, поточний стан здоров’я для кращого управління продуктивністю роботи.

Відповідно однією з перших функцій додатку має бути саме робота з працівниками. Додаток повинен надавати функціонал для збереження, зміни та видалення даних про працівників. Не менше важливим є швидкий та зручний перегляд інформації про працівників на окремій сторінці, а це — пошук та сортування даних за певними критеріями, як-то ПІБ чи відділ. Важливо, щоб UI відповідної сторінки не був перевантаженим, адже це прямо впливає на комфорт роботи користувача із системою.

Іншим завданням обліку є облік відвідуваності робітників. Відділу кадрів потрібно знати, коли і який працівник прибув на роботу і пішов з неї.

Це важливо як з точки зору забезпечення безпеки підприємства, адже таким чином можна відслідковувати небажану чи підозрілу активність, так і для кращого управління часом робітників.

Тому і додаток повинен зберігати інформацію про те коли і який саме працівник прибув чи пішов з робочого місця. Працівник HR відділу має мати можливість так само легко з нею взаємодіяти і переглядати.

Окремо варто зазначити і облік нещасних випадків на виробництві. Такі випадки порушують виробничий процес та завдають шкоди здоров'ю працівників. Визначення причин та наслідків, потерпілих, необхідно для запобігання цих подій, і пов'язаних з ними фінансових витрат в майбутньому.

Задля забезпечення розмежування доступу між окремими працівниками HR відділу (щоб кожен працівник мав свій сектор відповідальності), додаток має реалізовувати систему автентифікації та авторизації. А для відслідковування окремих змін, додаток повинен вести їх логування.

Підсумовуючи, додаток повинен надавати зручний функціонал для автоматизації розглянутих завдань обліку.

1.3 Огляд існуючих рішень

Ідея автоматизації обліку працівників з'явилась не вчора, тому на ринку існує немало різних програмних рішень, кожне з яких має свої особливості і ринкову нішу. Детально проаналізувавши деякі з них, ми зможемо дізнатись їхні ключові особливості і функції, які потрібно брати до уваги про розробці власного продукту, щоб мати можливість задовольнити потреби замовників і бути конкурентноздатними на ринку.

1.3.1 SAP SuccessFactors

SAP SuccessFactors — це комплексне хмарне рішення для менеджменту людського капіталу (HCM) [3], що працює за моделлю «програма як сервіс»

(SaaS). Воно включає в себе як стандартні функції по менеджменту HR, так і різноманітні модулі для управління окремими аспектами обліку працівників, відповідно до потреб замовника. SAP SuccessFactors це потужний інструмент, яким користуються великі компанії.

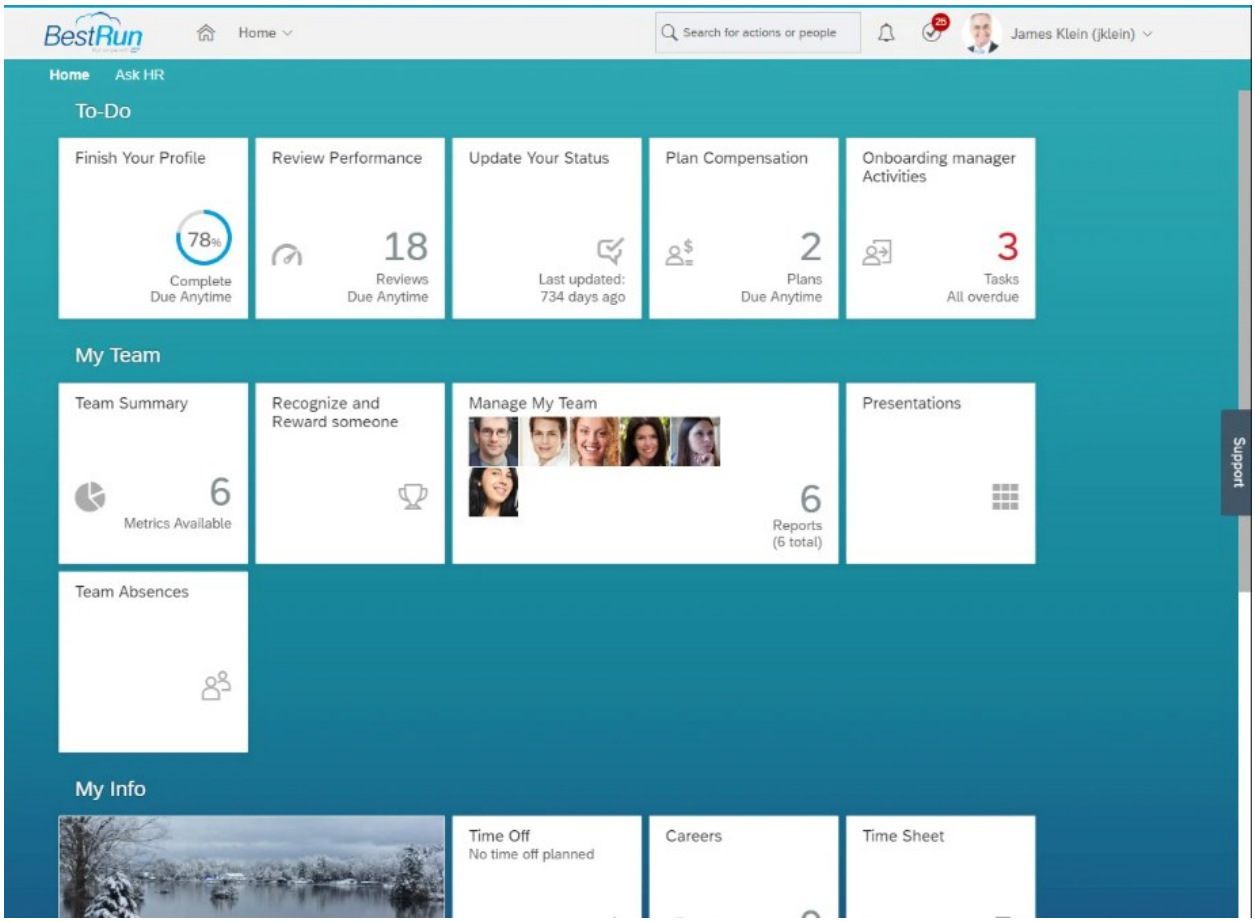


Рисунок 1.1 — Інтерфейс головної сторінки SAP SuccessFactors [3]

Переваги платформи:

— комплексне, All-in-One рішення, що включає всі аспекти обліку в одній платформі;
— надає перевагу індивідуальному, персоналізованому досвіду користувачів;
— зручний доступ з мобільних пристроїв в будь-який час;
— великі можливості по інтеграції з додатками інших компаній[4];
— дружній до користувачів інтерфейс;

Недоліки платформи:

— нестача прозорої інформації щодо цінової політики;

— відсутність безкоштовної версії чи безкоштовного періоду для ознайомлення;

— не найкраще рішення для малого бізнесу через перевантажений функціонал;

— обмежені можливості по кастомізації;

— частина користувачів вважають інтерфейс застарілим[5];

1.3.2 BambooHR

BambooHR також представляє з себе хмарне рішення для управління людськими ресурсами, але орієнтоване для малих та середніх підприємств. BambooHR надає простий та інтуїтивний інтерфейс для управління різними аспектами обліку працівників: керування даними працівників, розміщення вакансій і відстеження кандидатів, прийом на роботу, управління відпустками і контроль відсутності, управління продуктивністю робітників, звітність та аналітика.

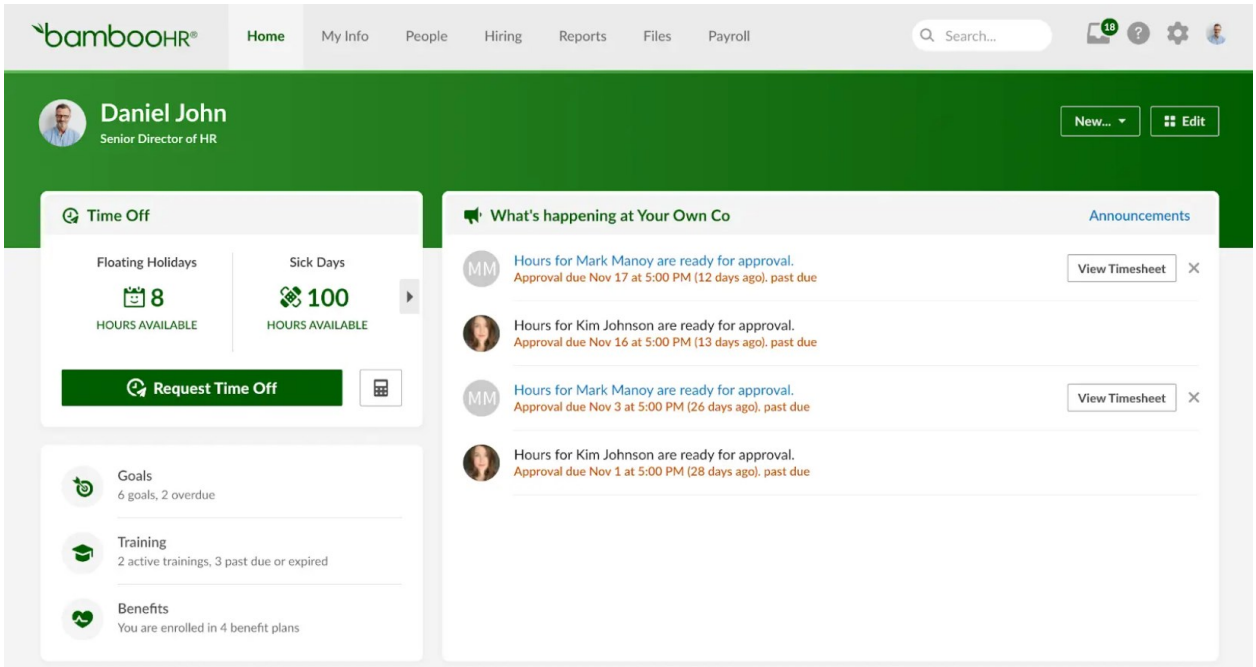


Рисунок 1.2 — Інтерфейс головної сторінки BambooHR [6]

Переваги платформи:

— дружній та інтуїтивно зрозумілий інтерфейс;
— безкоштовний період для користування [7];
— високий рівень кастомізації для задоволення специфічних потреб бізнесу;
— підтримує велику кількість сторонніх додатків для інтеграції;
— широка мережа партнерів;
— централізоване управління даними;

Недоліки платформи:
— обмежений набір можливостей для великих підприємств;
— не зовсім прозоре ціноутворення;
— певні функції направлені лише на працівників зі США [8];

1.3.3 Zoho People

Zoho People це іще одне хмарне рішення для управління HR, розроблене компанією Zoho Corporation. Воно позиціонується як рішення для компаній різного розміру, від малих до великих підприємств. Zoho People створене для автоматизації всього спектру HR-процесів — від рекрутингу, до управління продуктивністю.

Переваги платформи:
— наявність безкоштовного тарифного плану для організацій з менше ніж п’ятьма працівниками;
— простий та інтуїтивний інтерфейс;
— конкурентне ціноутворення, що підлаштовується під потреби замовника;
— інтеграція з іншими додатками із екосистеми Zoho;
— широкий функціонал [9];

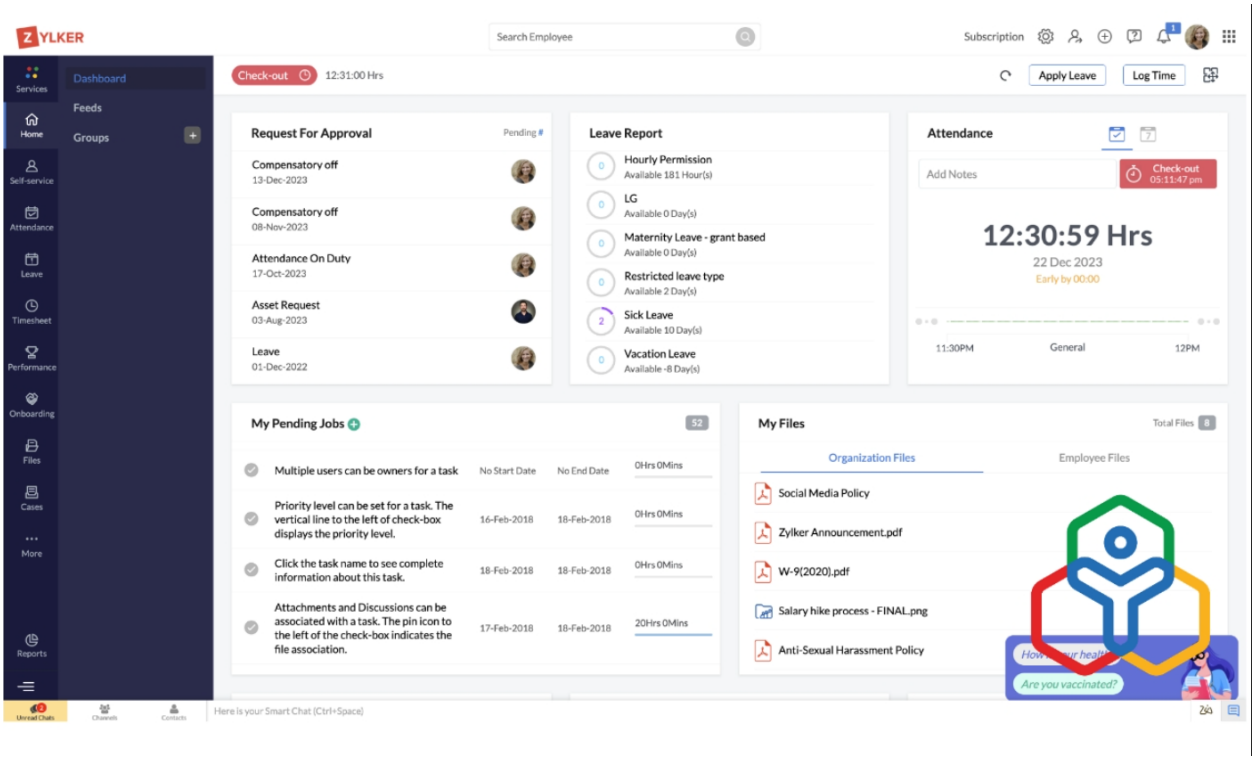


Рисунок 1.3 — Інтерфейс головної сторінки Zoho People

Недоліки платформи:

— обмежені можливості по інтеграції зі сторонніми додатками;

— брак певних специфічних функцій, необхідних компаніям з великою кількістю працівників;

— обмежена функціональність мобільного додатку;

— повільна клієнтська підтримка;

1.3.4 Workday HCM

Workday HCM, як і всі попередні, є хмарним рішенням для управлінням HR процесами. Воно орієнтоване на середні і великі компанії, надаючи широкий спектр функцій для автоматизації та оптимізації обліку кадрів.

Переваги даної платформи:

— широкі можливості по кастомізації для задоволення бізнес потреб;

— інтеграція з більш ніж шістьмастами сторонніми сервісами;

— надання спеціальних функцій, специфічних для окремої галузі [10];

— мобільний додаток для доступу з мобільних пристроїв;

— потужні інструменти зі звітності і збору аналітики;

— централізований доступ до інформації і функцій;

— інтуїтивна візуалізація даних для створення звітності [11];

— регулярні оновлення;

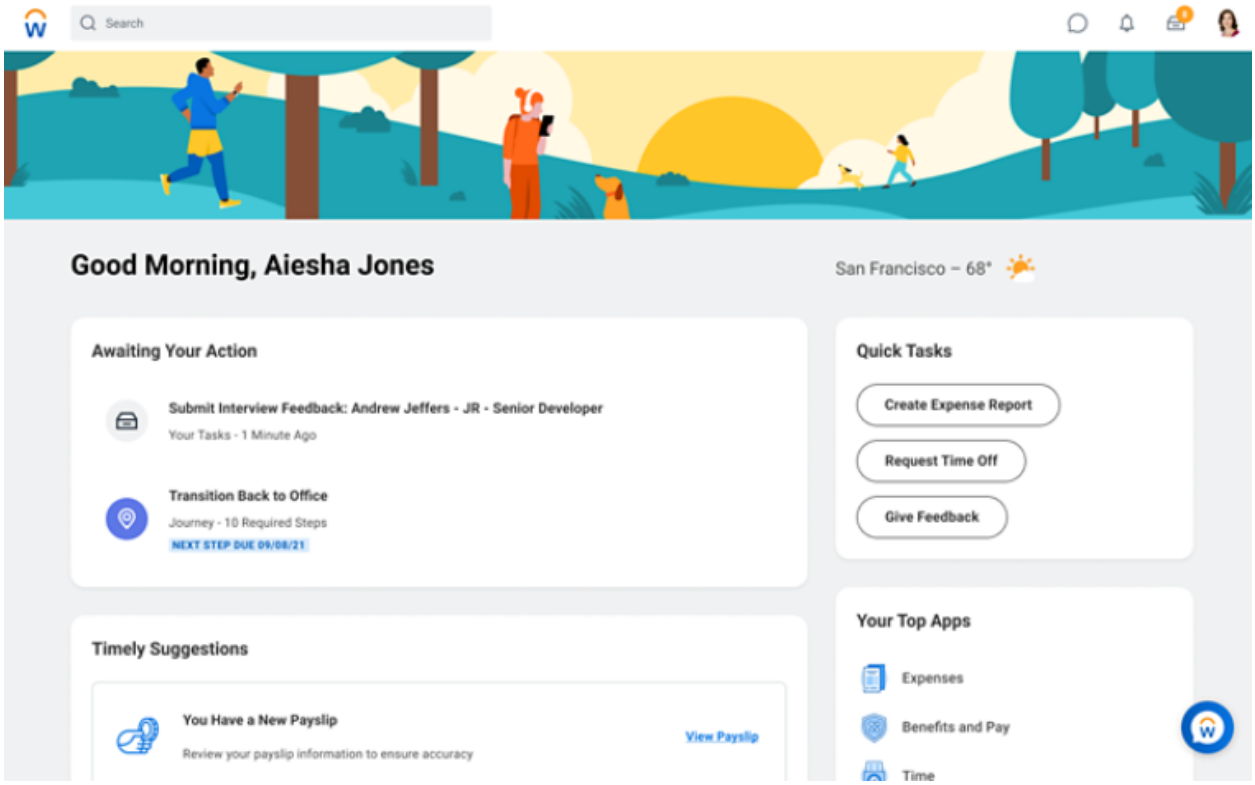


Рисунок 1.4 — Інтерфейс головної сторінки Workday HCM [12]

Недоліки даної платформи:

— відсутність прозорої інформації про ціну;

— відсутність безкоштовної версії чи періоду;

— загальна дороговизна [5];

— необхідність тривалої і трудомісткої конфігурації та навчання;

— обмежена підтримка інших мов;

— поганий технічний стан мобільного додатку;

1.3.5 Порівняльний аналіз існуючих рішень

Для аналізу ринку систем по менеджменту людського капіталу, було розглянути декілька різних платформ. Це як провідні у своїй галузі – SAP SuccessFactors, Workday HCM, так і менш популярні рішення – BambooHR та Zoho People, які однак можуть скласти достойну конкуренцію своїм опонентам. Це дослідження надає корисну інформацію про можливості існуючих рішень та чого їм бракує для задоволення конкретних потреб виробничих підприємств.

Окрім загальних особливостей та функціоналу кожної з платформ, необхідні розглянути і окремі деталі та специфічні функції, які є необхідними для виробничих підприємств, щоб розуміти потреби при розробці власного рішення. Результати цього аналізу продемонстровані у таблиці 1.1.

Таблиця 1.1 — Порівняльний аналіз HCM рішень

	Назва рішення				
Функціонал	SAP SuccessFactors	BambooHR	Zoho People	Workday HCM	Пояснення
Персональна інформація	+	+	+	+	Збереження інформації про працівників
Деталі праце-влаштування	+	+	+	+	Інформація про посаду і відділ робітника

Продовження таблиці 1.1

	Назва рішення				
Функціонал	SAP SuccessFactors	BambooHR	Zoho People	Workday HCM	Пояснення
Час роботи	+	+	+	+	Час перебування робітника на робочому місці
Присутність на роботі	+	+	+	+	Можливість відстеження чи був працівник на роботі
Поточний статус	+	+	+	+	Інформація про лікарняні, відпустки і т.п.
Стан здоров'я	+	±	–	±	Відстеження інформації про те, чи травмований працівник
Нещасні випадки	+	+	–	±	Інформації про нещасні випадки, причину, хто постраждав

Кінець таблиці 1.1

	Назва рішення				
Функціонал	SAP SuccessFactors	BambooHR	Zoho People	Workday HCM	Пояснення
Аналітика	+	+	+	+	Збір аналітичних даних
Авторизація, Розділення доступу	+	+	+	+	Доступ до інформації відповідно до прав користувача
Моніторинг змін в системі	+	+	+	+	Відстеження хто і яку інформацію вносить в систему
Зручний інтерфейс	–	+	+	+	Можливість зручної і зрозумілої роботи з системою

Проведений аналіз свідчить про те, що всі чотири платформи, які було розглянуті, містять реалізацію ключових функцій необхідних для управлінням HR. Однак в деяких аспектах вони мають недоліки, які впливають на зручність роботи при проведенні обліку кадрів на виробничих підприємствах. Наприклад, SAP SuccessFactors хоч і має широкий фукціонал, проте за відгуками користувачів його інтерфейс є дещо архаїчним і тому менш

зручним. BambooHR теж пропонує значні можливості, але є нарікання на певні аспекти щодо відстеження стану здоров'я працівників. Зі свого боку Zoho People та Workday HCM так само спотикаються на роботі з обліком здоров'я робітників, хоч і пропонують зручний інтерфейс й багато інших функцій.

Результати цього дослідження дозволять краще зрозуміти напрямок розвитку власного програмного рішення, що пропонує і потребує ринок подібних систем, на які аспекти варто звернути увагу в майбутньому.

1.4 Вимоги до системи, що розробляється

Отже, метою та ідеєю даного проекту є створення додатку, який буде зручним інструментом для автоматизації процесів обліку кадрів на виробничому підприємстві. Застосунок має простим та зрозумілим у використанні, задовольняючи необхідні вимог, що були узагальнено наведені в пункті 1.2 цього розділу.

Огляд предметної області дозволив визначити, який функціонал має реалізовувати фінальний додаток. А дослідження і порівняння існуючих рішень вказує на аспекти, які потребують окремої уваги. Ключовими функціями, що необхідно втілити в застосунку є:

1. Реалізація можливості обліку конкретно працівників (створення, видалення, редагування), зручна взаємодія з даною інформацією завдяки функціям пошуку та сортування, зрозумілому UI.
2. Реалізація аналогічного функціоналу для інших завдань обліку, як-то відвідуваність, нещасні випадки та інші.
3. Відслідковування змін в системі завдяки логуванню подій та взаємодії з інформацією.
4. Реалізація автентифікації та авторизації для управління доступом.

ВИСНОВОК ДО РОЗДІЛУ 1

В цьому розділі було проаналізовано ключові аспекти, пов'язані зі сферою обліку працівників, досліджено декілька доступних рішень на ринку і їхні особливості, сформовано вимоги до нашого додатку.

Було з'ясовано, що задачу обліку кадрів полягає у своїй суті в роботі з різноманітною інформацією про працівників, яка необхідна для підприємству для ведення своєї діяльності. Ця інформація включає в себе і особисті дані працівників, які потрібні для оформлення працівника згідно законодавства відповідної країни, і інформації про робочі години для коректного нарахування заробітної плати, і інші дані, які потрібні для організації роботи з максимальною ефективністю. Було приділено увагу і особливостям функціонування виробничих підприємств та як ці особливості впливають на облік HR в цих умовах. Тут важливими є кваліфікація робітників, робота з нещасними випадками на виробництві, планування часу.

Рішення по автоматизації HR процесів, що представлені на ринку, демонструють широкі можливості і можуть бути гарним вибором для різних замовників. Але і вони не позбавлені окремих недоліків, що можуть бути особливо важливим для виробничих підприємств. Тому існує простір і потенціал для розвитку конкурентних рішень, які могли б зосередитись на цій ніші й принести нові ідеї та інновації.

На основі отриманої інформації було виокремлено потреби, які має задовольняти результуючий додаток.

Підсумовуючи, сектор рішень по автоматизації обліку кадрів є конкурентним місцем, що продовжує розвиватись. Для розвитку в цій сфері необхідно враховувати потреби клієнтів, адаптовуватись і розширювати власний функціонал.

РОЗДІЛ 2
АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ СИСТЕМИ. ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ

2.1 Структура системи та її складові

Для реалізації вимог, що були визначені в попередньому розділі, було обрано веб-додаток з монолітною архітектурою. Зі свого боку, сам додаток буде реалізовувати класичну тришарову архітектуру, що складається з трьох рівнів (шарів): візуального рівня – Presentation Layer, рівня бізнес логіки – Business Logic Layer та рівня доступу до даних – Data Access Layer. В якості сховища даних використовуватиметься Microsoft SQL Server. Загальний вигляд структури системи можна побачити на рисунку 2.1.

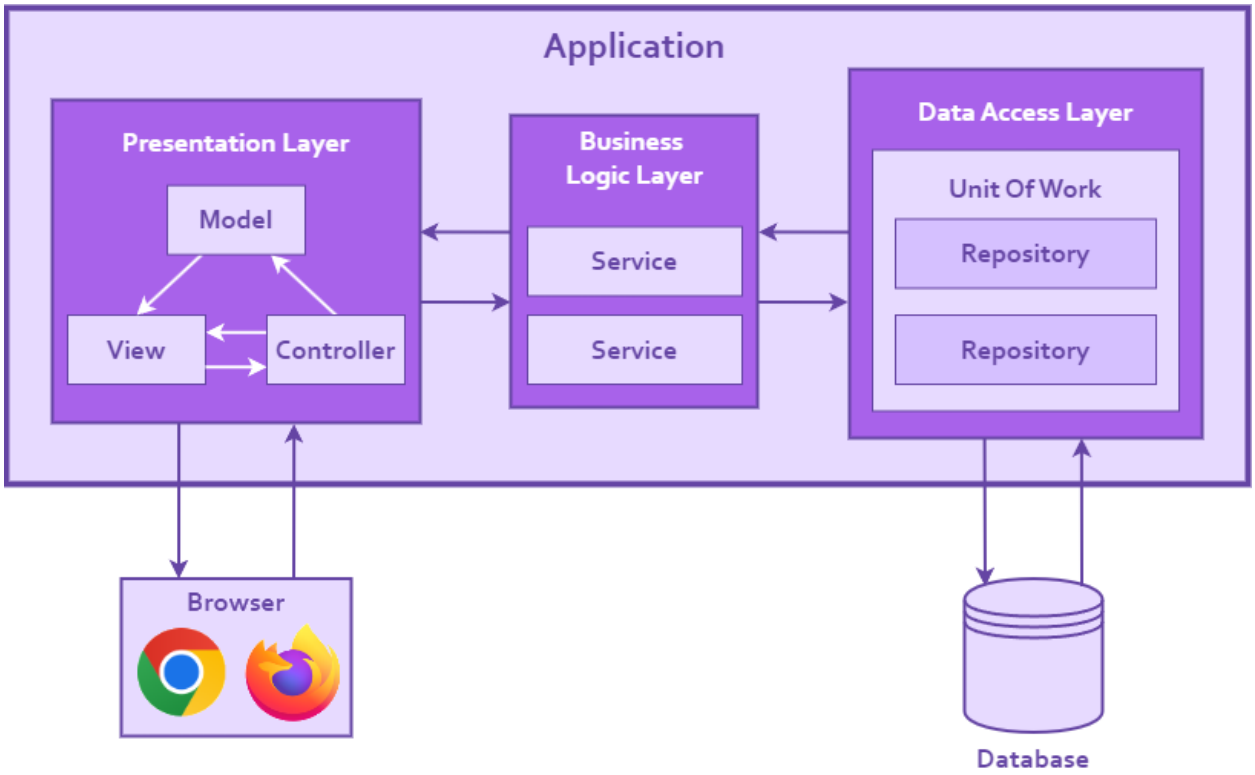


Рисунок 2.1 — Загальна структура додатку

Користувач взаємодіє з додатком через браузер, а візуальний рівень застосунку, який відповідає за користувацький інтерфейс та взаємодію

користувача з інтерфейсом програми [13], обробляє дії користувача і реагує на них. Він же отримує введену користувачем інформацію.

Варто звернути увагу на те, що візуальний рівень має свою внутрішню структуру, реалізовуючи поширений архітектурний шаблон для веб-додатків — MVC. Він складається з Моделей (Model), Представлень (View) та Контролерів (Controller). View являє собою елемент користувацького інтерфейсу, як-то, наприклад, окрема сторінка. Всі дії користувача, які він здійснює з View, обробляє саме Controller. Він же, зі свого боку, працює з Model для отримання чи збереження інформації. Model в цьому випадку є окремі класи, необхідні для коректної роботи з інформацією, та сервіси, що надає цьому рівню рівень бізнес логіки.

Рівень бізнес-логіки отримує дані та запити від візуального рівня, виконує обробку даних, відповідно до потреб додатку; співпрацює з рівнем доступу до даних для їх збереження чи отримання. Він надає візуальному рівню окремі програмні структури — сервіси. Ці сервіси виступають сполучною ланкою для отримання даних візуальним рівень.

Не простим є і рівень доступу до даних, який здійснює комунікацію з джерелом даних, якою зазвичай є база даних [13]. Тут знаходяться моделі, які є програмними об’єктам для роботи з інформацією, яка була отримана з сутностей бази даних. Також він реалізує патерни проектування Repository (Репозиторій) та Unit Of Work (Одиниця роботи). Вони необхідні для створення додаткового шару абстракції, що забезпечує кращу підтримку, розширення та тестування.

Ознайомившись із структурою системи, розглянемо більш детально окремі архітектурні рішення, чому були обрані саме вони та які існують варіанти.

2.2 Архітектурні принципи розробки

2.2.1 Визначення поняття архітектури

Для розробки власного рішення необхідно обирати таку архітектуру програми, яка б найкраще підходила для вирішення поставлених задач. Але для початку варто розібратись з визначенням того, що таке програмного забезпечення архітектура.

«*Архітектура програмного забезпечення комп'ютерної системи це набір структур необхідних для міркування про систему, які включають в себе елементи програмного забезпечення, відносини між ними і їхні властивості.*» [14, с.37]

Інакше кажучи, архітектура це набір елементів, які описують як систему, так і відносини всередині цієї системи. Тому від правильного вибору архітектури залежить наскільки добре буде працювати майбутній додаток, наскільки зручною буде його розробка, нарощення функціоналу і підтримка.

2.2.2 Вибір типу додатку

Якщо говорити загалом, одним з основних архітектурних рішень при розробці додатку є вибір між веб і десктопним додатком.

Веб-додаток це програма, яка працює на віддаленому сервері, і доступ до якої здійснюється через браузер по мережі інтернет. Тобто користувач взаємодіє з програмою у вікні браузера, але вся обробка даних відбувається на віддаленому сервері.

Зауважимо, що варто розрізняти поняття «веб-сайту» і «веб-додатку». Веб-сайт це здебільшого набір статичних даних, який має просту функцію по демонстрації якоїсь інформації, наприклад, статей на простому новинному ресурсі. Можливості користувача по взаємодії з ним досить обмежені. Веб-

додаток ж надає складніший функціонал і є інтерактивним — дані оновлюються в реальному часі в результаті взаємодії з користувачем.

На противагу веб-додатку, десктопний додаток це програма, яка завантажується і встановлюється безпосередньо на користувацький комп'ютер, та працює під управлінням користувацької операційної системи.

Обидва ці варіанти мають свої особливості, які необхідно розглянути більше детально, щоб обрати той, що краще підходитиме для нашого проекту.

Як вже було згадано вище, веб-додаток працює на віддаленому сервері з доступом через мережу інтернет, що забезпечує високу гнучкість та незалежність від певної платформи [15]. Таким чином, цим додатком можливо користуватись на будь-якому пристрої що має браузер та доступ до інтернету — персональний комп'ютер, ноутбук, смартфон і навіть Raspberry Pi (одноплатний комп'ютер). Через те, що програма працює віддалено, її непотрібно окремо встановлювати і оновлювати самому користувачу. Це все відбувається без його участі. Але це коштує продуктивності і швидкості реакції додатку на зміни, адже даним потрібен час, щоб пройти по мережі від користувача до серверу і назад. Також сам браузер, в якому користувач і взаємодіє з додатком, витрачає додаткові ресурси комп'ютера.

Зі свого боку десктопний додаток може працювати без обов'язкового під'єднання до інтернету (залежно від призначення додатку) і напряму взаємодіяти з апаратним забезпеченням користувацького пристрою, надаючи більш оптимізований і послідовний користувацький досвід [15]. Адже прямий доступ до «заліза» комп'ютеру забезпечує кращу продуктивність, через відсутність додаткового шару абстракції у вигляді браузера, як при роботі з веб-додатком. Таке рішення є найкращим варіантом, коли програмі потрібен безпосередній доступ до «заліза» для своєї роботи, чи висока продуктивність є основним пріоритетом. Але цей вибір не обходиться без своїх недоліків. Більш близька робота з «залізом» комп'ютера означає прив'язку до конкретної платформи і ОС. Через це доводиться розробляти програму під кожну

платформу окремо, що збільшує вартість розробки. Також користувачу необхідно вручну встановлювати програму на кожен пристрій і слідкувати за оновленнями, що створює певні незручності.

Для нашого рішення було обрано веб-додаток, тому що:

— Питання продуктивності не є першочерговим пріоритетом, бо наша предметна область не передбачає роботи з великими потоками даних, де це питання було б одним з перших.
— Постійне під'єднання до мережі теж не є недоліком, адже організація так чи інакше повинна мати централізовану базу даних.
— Легкість розробки під одну платформу і відповідна дешевизна, швидкість розгортання і впровадження оновлень, можливість роботи на будь-якому пристрої сповна компенсують цей недолік.
— Всі існуючі рішення в даній предметній області, що були розглянуті раніше, також, в тому чи іншому виді, представляють саме веб-додатки.

2.2.3 Аналіз монолітної та мікросервісної архітектур

Іншим важливим архітектурним рішенням є вибір між монолітною та мікросервісною архітектурою додатку. Це рішення значно впливає на розробку самого додатку, адже ці архітектури суттєво відрізняються і вимагають використання різних підходів для їх реалізації.

Розглянемо спочатку монолітну архітектуру. Монолітна архітектура вважається традиційним способом створення додатків [16]. Це випробуваний часом підхід, суть якого полягає в тому, що додаток створюється як єдиний неподільний блок. Цей блок може включати в себе окремі частинки як-то UI чи бізнес-логіка, але вони всі взаємодіють як єдине ціле.

Перевагами монолітної архітектури є її простота, менша вартість початкової розробки й менша експлуатаційна складність. Вона доволі легка в розумінні й тому не потребує особливих зусиль при початковій розробці. Також через свою простоту вона не потребує якоїсь додаткової

інфраструктури чи складного менеджменту, що і зменшує витрати на операційну діяльність.

Але з ростом складності додатку та його розширенням починають проявлятись і обмеження даної архітектури. Так, при досягнення певного об'єму коду, додаток стає надто складним для розуміння, бо банально важко прослідкувати що від чого залежить і що за що відповідає, коли все тісно взаємопов'язано. Через це стає все важче масштабувати застосунок, бо не вийде масштабувати лише якусь частину, необхідно масштабовувати весь додаток. Важко проваджувати і нові технологічні рішення, адже вони вимагають переробки значної частини додатку, якщо не його повністю [16].

Другим варіантом є мікросервісна архітектура. Суть її полягає в тому, що додаток розділяється на набір окремих, менших і незалежних частинок — сервісів. Ці сервіси виконують окремі функції, але працюючи разом вони складають по суті той самий додаток. Схематичне порівняння цих архітектур зображено на рисунку 2.2

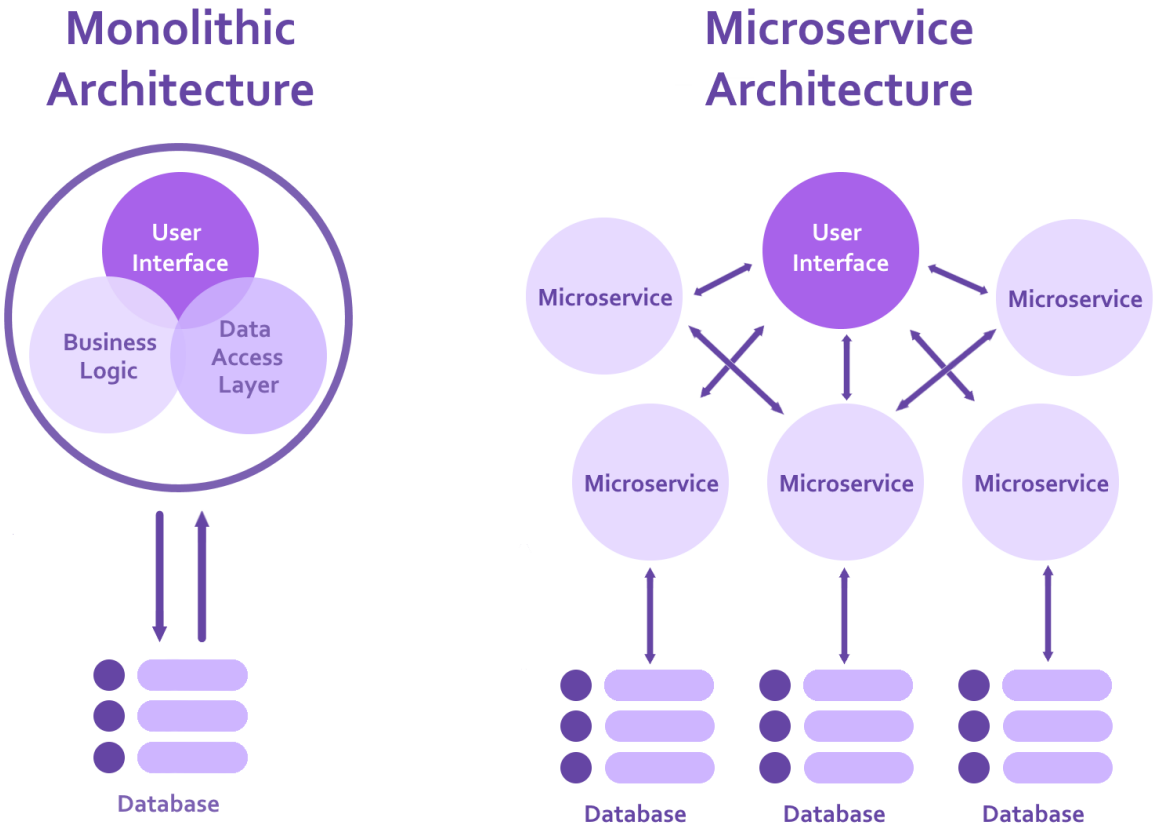


Рисунок 2.2 — Порівняння монолітної та мікросервісної архітектур

Такий підхід надає рішенню гнучкості, масштабованості і відмовостійкості. З окремими сервісами простіше працювати і розширювати їхні можливості, не боючись нашкодити іншим функціям додатку. Якщо необхідно додати новий функціонал — достатньо створити лише окремий сервіс, який його реалізує. І жодних проблем через конфлікти з іншими частинами додатку. При виникненні проблеми з одним із сервісів, інші продовжать працювати як і раніше, бо вони всі де-факто незалежні. З ростом числа користувачів можна легко наростити кількість однакових сервісів, щоб впоратись з навантаженням.

Хай там як, але ці переваги тягнуть за собою вищі вимоги до інфраструктури і серверних ресурсів. Через те, що кожен сервіс незалежний і працює в окремому ізольованому середовищі — контейнері, необхідно більше обчислювальних потужностей серверів. А це додаткові витрати. Іншим недоліком стає складність організації взаємодії між сервісами. Для виконання певних функцій необхідна взаємна робота багатьох сервісів, яку складно організовувати при великій їх кількості. Відповідно і розробка мікросервісів коштує дорожче, через витрати на інфраструктуру та складний менеджмент.

Зважаючи на наведені вище дані, для нашого додатку було обрано монолітну архітектуру. Основними причинами для цього стали простота і легкість розробки. Кодова база нашого додатку не настільки велика, щоб зіткнутись з недоліками цього підходу, а швидкість впровадження нових функцій, через відповідну простоту, є важливішими на цьому етапі. Це робить рішення більш конкурентноздатним на ринку.

2.2.4 Вибір тришарової архітектури

Структура тришарової архітектури або «three-layer architecture» вже була досить детально розглянута раніше, тому зупинимось на причинах її вибору.

Тришарова архітектура забезпечує розділення програми на відокремлені рівні, де кожен рівень має чіткі обов’язки і відповідає виключно за свою

частину функціоналу. Таке розділення забезпечує модульність, кращу підтримку і обслуговування додатку, адже зміни в одному рівні майже не впливають на інший. В результаті виходить додаток із зрозумілою структурою та чітким розділенням між рівнями.

2.2.5 Шаблони Repository та Unit Of Work

Патерн Repository використовується як спосіб керувати логікою доступу до даних з централізованої локації [17], фактично представляючи з себе колекцію. Шаблон Unit Of Work використовують в союзі з ним для управління багатьма різними операціями (Insert, Update, Delete) в одній транзакції. Таким чином, Unit Of Work координує роботу кількох репозиторіїв.

Наш проект реалізовує вищенаведені шаблони з наступних причин:

— Ці патерни покликані створити рівень абстракції між рівнем доступу до даних та рівнем бізнес логіки, що сприяє чіткішому розділенню відповідальності та робить підтримку коду легшою [18].

— Додатковий шар абстракції полегшує автоматизоване тестування бізнес логіки, бо відпадає необхідність безпосередньої взаємодії з джерелом даних [18]. Це дозволяє здійснювати юніт тестування та покращує загальну якість коду.

— Також, завдяки цьому можна легко змінювати джерело даних, що лежить в основі цього шару, якщо виникне потреба.

— В поєднанні з техніками «лінивого завантаження», можна оминути зайві звернення до бази даних, покращивши продуктивність роботи додатку.

Як результат, таким чином, створюється добре структурований шар доступу до даних, що легко тестується, підтримується і є слабкозв’язаним по відношенню до рівня бізнес логіки.

2.3 Вибір технологій розробки

Існує немало технологій, за допомогою яких можливо реалізувати наш додаток і відповідні архітектурні рішення, мова про які йшла раніше. Однак для цього проекту було вибрано C# і платформу .NET.

2.3.1 Платформа .NET та мова програмування C#

NET це безкоштовний, кросплатформовий фреймворк з відкритим вихідним кодом для розробки програмного забезпечення. Він дозволяє створювати велику кількість різних додатків: веб-додатки, мобільні, десктопні застосунки, хмарні сервіси, ігри та багато іншого [19]. .NET надає розробникам окреме середовище виконання, набір бібліотек та інструментів для розробки.

Для розробки на .NET можна використовувати різні мови програмування, але основною є C#. C# — це об’єктно-орієнтована мова програмування загального призначення [20]. Завдяки свої об’єктно-орієнтованій природі, строгій типізації, автоматичному управлінню пам’яттю, вона забезпечує зручну та легку розробку додатків будь-якої складності.

Основним перевагами, що і забезпечили вибір цієї платформи, є те, що це сучасне та актуальне рішення. Воно постійно розвивається і поповнюється новим функціями. Розробкою .NET займається один з найбільших світових технологічних гігантів — компанія Microsoft. Дана платформа від початку задумувалась як універсальне, високопродуктивне та масштабоване рішення, чим і завоювала визнання серед розробників. Вона має велику та активну спільноту розробників, що сприяє розробці продуктів, завдяки наявності великої кількості необхідних матеріалів та документації.

2.3.2 ASP.NET Core MVC

Для комфортної розробки веб-додатків на платформі .NET було створено окремий фреймворк, що базується на .NET — ASP.NET Core MVC. ASP.NET Core MVC забезпечує структурований підхід для створення веб-додатків завдяки вбудованому використанню архітектурного шаблону MVC (Модель-Представлення-Контролер). Він надає такі функції як маршрутизація, прив'язка моделей, валідація та впровадження залежностей [21], що робить розробку легше і швидше. А враховуючи кросплатформну підтримку, модульність і простоту, цей фреймворк впевнено втримує головні позиції на ринку розробки веб-застосунків.

2.3.3 Entity Framework Core

Окремо варто згадати Entity Framework Core (EF Core). EF Core це сучасний, легкий ORM фреймворк, розроблений Microsoft. Він надає .NET розробникам можливість працювати з базою даних, використовуючи предметно-специфічні об'єкти та властивості, абстрагуючись від таблиць та стовпчиків, що лежать в основі бази даних. Як і інші продукти екосистеми .NET, EF Core підтримує багатьох різних провайдерів баз даних і надає систему міграцій і відслідковування змін в сутностях [22].

2.4 Вибір бази даних

Останнім аспектом, який ще не було розглянуто, залишився вибір бази даних для додатку. Враховуючи технології, що були розглянуті і обрані в попередніх розділах, було вирішено обрати Microsoft SQL Server.

Microsoft SQL Server, як можна зрозуміти з назви, розроблений тією ж компанію, що і платформа .NET, тому ідеально інтегрується з нею, що надає безшовний досвід. SQL Server є потужною і багатофункціональною системою управління реляційними базами даних, що пропонує багато переваг над

іншими рішенням. Це і легкість роботи та адміністрування, висока продуктивність та масштабованість, активна спільнота і широка екосистема. Варто відмітити і високу захищеність даних, що особливо важливо при роботі з персональними даними працівників. А вбудовані інструменти для аналітики стануть в нагоді для отримання додаткової інформації і забезпечення більш ефективної роботи по обліку кадрів, що позитивно впливає на роботу всього підприємства, яке буде використовувати дане рішення.

В підсумку, Microsoft SQL Server забезпечує всім необхідним для комфортної розробки та надає окремі переваги, що є особливо корисним у вибраній предметній області.

ВИСНОВОК ДО РОЗДІЛУ 2

В цьому розділі було розглянуто структуру системи додатку, різні архітектурні аспекти, технології розробки та базу даних, які є ключовими для розробки якісного додатку.

Було визначено, що додаток має бути монолітним веб-додатком з тришаровою архітектурою. Окремо описано і зв'язки між окремими елементами.

Визначено поняття архітектури програмного забезпечення, розглянуто вибір між настільним та веб-додатком, та обґрунтовано вибір саме веб-додатку, через його легкість розробки та універсальність, швидкість і простоту впровадження оновлень.

Інший важливий елемент, що був досліджений, це вибір між монолітною та мікросервісною архітектурами. Хоч мікросервісна архітектура і пропонує високу гнучкість, масштабованість та відмовостійкість, вибір все ж впав на «моноліт», бо він більш простий і дозволяє швидше впроваджувати нові функції, а відносно малий розмір додатку майже нівелює його недоліки.

Також було розглянуто шаблони проектування Repository та Unit Of Work., що були застосовані у шарі доступу до даних. Вони створюють додатковий шар абстракції, що краще розділяє шар доступу до даних з шаром бізнес-логіки, роблячи його легшим в тестуванні і слабкозв'язаним.

Крім того, було обрано платформу .NET з мовою C#, та фреймворки ASP.NET Core MVC і EF Core. Такий вибір обґрунтовується їх тісною інтеграцією, широкими можливостям, сучасністю та актуальністю цих засобів розробки. Обраною базою даних стала Microsoft SQL Server, оскільки вона ідеально вписується в обрану екосистему та пропонує хороші переваги.

Отож, вибір хорошої архітектури та оптимальних засобів розробки забезпечує легку та швидку розробку необхідного рішення з автоматизації обліку кадрів.

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ

3.1 Організація бази даних

В основі кожної системи, якій необхідно маніпулювати даними, лежить база даних, яка має забезпечувати надійний та легкий доступ до цих даних. Правильна організація такої бази даних забезпечує надійне зберігання інформації, вберігає від можливих помилок при роботі з цими даними. Щоб створити таку базу даних, потрібно всю інформацію, що потребує збереження, згрупувати в окремі таблиці (сутності) і визначити, які зв'язки мають бути між цими сутностями. Кожна таблиця має свій набір унікальних полів та відповідає за окрему частину інформації, що зберігається. База даних в цілому має бути нормалізована відповідно до вимог нормальних форм, щоб запобігти зайвому дублюванню даних, усунути аномалії оновлення, додавання чи видалення записів.

Для задоволення потреб нашого додатку було створено набір таблиць, де кожна таблиця відповідає окремій сутності в системі:

Employees. Ця сутність є основною в системі, бо відповідає за збереження інформації про робітника, його робочого статусу, стану здоров'я і т.п. Інакше кажучи, таблиця Employees є списком працівників із даними про них. Кожен працівник має унікальний ідентифікатор, який потрібен для ідентифікації цього працівника в системі, оскільки інші існуючі ознаки можуть бути складними й не надто надійними для цього. Наприклад, прізвища і імена можуть повторюватись, а паспортні дані мають різну структури, залежно від серії паспорту. Структуру даної таблиці можна розглянути нижче в таблиці 3.1:

Таблиця 3.1 — Структура таблиці Employees

Назва поля	Тип поля в БД	Опис
Id	int (Primary Key)	Первинний ключ, унікальний ідентифікатор кожного робітника в системі.
Name	nvarchar	Ім’я робітника
Surname	nvarchar	Прізвище робітника
Patronym	nvarchar	Ім’я по батькові робітника
BirthDate	datetime2	Дата народження робітника
PassportNumber	nvarchar	Номер паспорту (номер документу) робітника для оформлення його згідно законодавчих норм
TIN	nvarchar	Реєстраційний номер облікової картки платника податків, або його аналог у інших країнах
HiringDate	datetime2	Дата прийому працівника на роботу
Position	nvarchar	Посада, на яку було прийнято працівника
DepartmentId	int (Foreign Key)	Унікальний ідентифікатор відділу підприємства. Є посиланням на Id відділу в таблиці “Departments”
WorkStatusId	int (Foreign Key)	Унікальний ідентифікатор запису про робочий статус. Є посиланням на Id запису в таблиці “WorkStatuses”
HealthStatusId	int (Foreign Key)	Унікальний ідентифікатор запису про статус здоров’я. Є посиланням на Id запису в таблиці “HealthStatuses”

Кінець таблиці 3.1

Назва поля	Тип поля в БД	Опис
HiredById	int (Nullable)	Унікальний ідентифікатор робітника, який оформив прийом даного робітника на роботу
AccessLevel	nvarchar	Рівень доступу працівника до системи

В результаті маємо всю необхідну інформацію про працівника, яку можна легко отримати, користуючись унікальними Id.

Departments. Ця таблиця необхідна для збереження інформації про відділи, що існують на підприємстві, та взаємовідносини між ними. Знаючи до якого відділу належить той чи інший працівник, можна легко його знайти, відсортувавши за певним відділом. Оскільки, працівник пов'язаний з відділом, так само можна переглянути список всіх працівників у певному відділі.

Таблиця 3.2 — Структура таблиці Departments

Назва поля	Тип поля в БД	Опис
Id	int (Primary Key)	Первинний ключ, унікальний ідентифікатор відділу
ParentId	int (Nullable)	Унікальний ідентифікатор відділу підприємства, дочірнім до якого є цей відділ
Name	nvarchar	Назва відділу
CreationDate	datetime2	Дата створення даного відділу
Description	nvarchar (Nullable)	Коротка інформація про відділ

Як можна було помітити раніше, сутність "Departments" пов'язана з "Employees" через відносини "один до багатьох". Це через те, що відділ може

мати багато працівників, але кожен окремий працівник прив'язаний лише до одного відділу. А завдяки зберіганню Id батьківського відділу, можливо вибудувати чітку ієрархію відділів на підприємстві.

WorkStatuses. Дана таблиця використовується для збереження інформації про можливі стани або статуси робітника, наприклад, чи він на роботі, чи у відпустці й т.п. Таку інформацію можливо було б зберігати і просто окремим полем в таблиці працівників, але щоб запобігти неузгодженості даних, які вводили б різні користувачі системи, було вирішено виокремити їх в окрему сутність. Тим більше, так можна в будь-який момент швидко змінити потрібний статус, щоб він відповідав реальному стану справ.

Таблиця 3.3 — Структура таблиці WorkStatuses

Назва поля	Тип поля в БД	Опис
Id	int (Primary Key)	Первинний ключ, унікальний ідентифікатор робочого статусу
Status	nvarchar	Сам статус (стан)

Сутність "WorkStatuses" теж пов'язана з "Employees" зв'язком "один до багатьох", оскільки багато працівників можуть мати один і той самий робочий статус одночасно.

HealthStatuses. Таблиця аналогічна за своєю суттю до попередньої, тільки зберігає інформацію про стан здоров'я робітника.

Таблиця 3.4 — Структура таблиці HealthStatuses

Назва поля	Тип поля в БД	Опис
Id	int (Primary Key)	Первинний ключ, унікальний ідентифікатор стану здоров'я
Status	nvarchar	Сам статус (стан)

Вона теж пов’язана теж пов’язана з “Employees” зв’язком “один до багатьох”, бо той самий “стан здоров’я” в системі одночасно може бути у різних робітників.

Attendances. Ця сутність потрібна для збереження інформації про відвідуваність працівником робочого місця. Вона зберігає відомості про те, коли і який робітник прийшов і пішов з об’єкту, що важливо для виробничих підприємств, які мають свою закриту територію і дороге обладнання. За її допомогою можна відслідкувати як сумлінність робітників, так і небажану чи підозрілу активність.

Таблиця 3.5 — Структура таблиці Attendances

Назва поля	Тип поля в БД	Опис
Id	int (Primary Key)	Первинний ключ, унікальний ідентифікатор запису про відвідування
Date	datetime2	Дата запису
EntryTime	datetime2	Час коли робітник прийшов на роботу
ExitTime	datetime2	Час, о котрій робітник пішов з роботи
EmployeeId	int (Foreign Key)	Унікальний ідентифікатор працівника. Є посиланням на Id працівника в таблиці “Employees”

Ця сутність пов’язана з сутністю “Employees” зв'язком “багато то одного”, тому що на один запис в таблиці працівників є багато пов’язаних записів із таблиці “Attendances”.

WorkAccidents. Дана таблиця відіграє досить важливу роль, адже вона зберігає інформацію про всі нещасні випадки, що трапились на підприємстві, а саме: причину, час і місце події з описом обставин цього випадку. Завдяки їй

можна визначати, де на підприємстві є проблеми з дотриманням техніки безпеки, збирати аналітику по отриманим даним.

Таблиця 3.6 — Структура таблиці WorkAccidents

Назва поля	Тип поля в БД	Опис
Id	int (Primary Key)	Первинний ключ, унікальний ідентифікатор випадку
Name	nvarchar	Назва окремого випадку
Cause	nvarchar	Опис причини нещасного випадку
AccidentDateTime	datetime2	Дата та час виадку
Description	nvarchar	Поле для опису окремих деталей нещасного випадку та іншої інформації

В результаті отримуємо таблицю зі всією необхідною інформацією, яку можна швидко знайти, про кожен випадок,. Але вона була б не повна, без інформації про тих робочих, які постраждали в кожному з нещасних випадків. Для того, щоб зв'язати цю таблицю з таблицею робітників, нам знадобиться проміжна таблиця.

EmployeeAccidents. Проміжна таблиця між сутностями "Employees" та "WorkAccidents", яка відображає зв'язок "багато до багатьох" між ними.

Таблиця 3.7 — Структура таблиці EmployeeAccidents

Назва поля	Тип поля в БД	Опис
Id	int (Primary Key)	Первинний ключ, унікальний ідентифікатор запису, потрібний для зручнішої роботи з записами в таблиці

Кінець таблиці 3.7

Назва поля	Тип поля в БД	Опис
EmployeeId	int (ForeignKey)	Унікальний ідентифікатор працівника. Є посиланням на Id працівника в таблиці “Employees”
AccidentId	int (Foreing Key)	Унікальний ідентифікатор нещасного випадку. Є посиланням на Id випадку в таблиці “WorkAccidents”

Завдяки такій структурі з ідентифікаторами працівника і нещасного випаду, яка показує зв’язок між ними, один працівник може бути пов’язаним з багатьма різними випадками, як і кожен випадок з різною кількістю робітників.

EventLogs. Окремо варто згадати про таблицю з логуванням подій в системі. Її головна мета зберігати інформацію про те, які зміни відбуваються в системі, як-то додавання нових працівників, чи оновлення інформації про них. Також вона містить дані про час, коли подібні зміни були внесені, та хто став ініціатором цих змін — працівник, що вніс ці зміни. Це додатковий інструмент контролю за тим, що відбувається всередині системи, та спосіб зрозуміти, коли щось пішло не так, якщо це буде потрібно. Її структура наведена в таблиці 3.8.

Таблиця 3.8 — Структура таблиці EvenLogs

Назва поля	Тип поля в БД	Опис
Id	int (Primary Key)	Первинний ключ, унікальний ідентифікатор запису
EventType	nvarchar	Тип події (наприклад, додавання нового робітника)
EventDateTime	datetime2	Дата та час події

Кінець таблиці 3.8

Назва поля	Тип поля в БД	Опис
EventDescription	nvarchar	Опис події. Тут може бути інформація про те, кого чи що було додано, оновлено чи видалено
InitiatorId	int (Foreign Key)	Унікальний ідентифікатор працівника, який став ініціатор події

Отож, в результаті отримуємо чітко вибудовану структуру бази даних, де кожна таблиця зберігає тільки потрібну інформацію, всі сутності мають унікальні ідентифікатори і пов’язані між собою за допомогою зовнішніх ключів. Завдяки цьому, ми можемо легко отримувати необхідну інформацію, робити комплексні запити, що сполучатимуть кілька таблиць, якщо це буде потрібно з якихось причин.

Окрім описаних раніше таблиць, коротко доторкнемось до ще кількох. Це таблиця для збереження історії міграцій та таблиці для реєстрації користувачів. Перша повністю відповідає своїй назві і відслідковує, які зміни були внесені в базу даних зі сторони додатку за допомогою інструментів створення міграцій. Другі потрібні для безпечного збереження інформації про зареєстрованих користувачів. Вони є стандартизованими та генеруються автоматично, тому детально розглядати їх не будемо.

Наостанок зачепимо ще один компонент нашої бази даних, а саме використання тригерів. Тригер є особливою збереженою процедури в базі даних, виклик якої відбуваються при настанні особливої події. Цими подіями можуть бути Insert — додавання даних, Delete — видалення та Update — зміна даних у стовпці таблиці. База даних цього проекту використовує тригери для наповнення таблиці “EventLogs” новими записами, при виникненні

вищезазначених подій. Приклад скрипту створення одного з подібних тригерів для події вставки даних зображено на рисунку 3.1:

```
CREATE TRIGGER trg_InsertEmployee
ON dbo.Employees
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO EventLogs(EventType, EventDateTime, EventDescription, InitiatorId)
    SELECT
        'InsertEmployee' AS EventType,
        GETDATE() AS EventDateTime,
        'New employee (ID: ' + CAST(i.Id AS NVARCHAR(50)) + ', Name: ' + i.Name
        + ', Surname: ' + i.Surname + ', DepartmentId: ' + CAST(i.DepartmentId AS nvarchar(50))
        + ') was created by employee with ID: ' + CAST(i.HiredById AS nvarchar(50)),
        i.HiredById AS InitiatorId
    FROM
        inserted AS i;
END;
```

Рисунок 3.1 — Скрипт створення тригеру

Після того, як було розглянуто всі особливості влаштування бази даних, перейдемо до розгляду архітектури додатку та її реалізацію.

3.2 Архітектура додатку

Як вже було зазначено в попередньому розділі, наш додаток реалізує тришарову архітектуру з шаром доступу до даних (Data Access Layer — DAL), шаром бізнес логіки (Business Logic Layer — BLL) та візуальним шаром, чи шаром представлення або презентації (Presentation Layer — PL). Це необхідно для розділення компонентів програми, що дозволяє краще адаптовувати програму до потреб користувачів, проваджувати нові технології, не боючись того, що доведеться сильно змінювати інші частини програми.

Для реалізації цього в середовищі розробки створюються окремі проєкти для різних шарів. Приклад цього продемонстровано на рисунку 3.1.

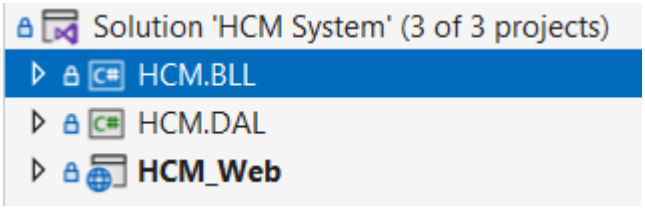


Рисунок 3.2 — Структура рішення додатку

Тепер, розглянемо кожен рівень та які програмні компоненти були реалізовані на кожному з них.

3.3 Рівень доступу до даних

Цей рівень є основою додатку і слугує сполучною ланкою між базою даних та вищими логічними шарами застосунку. Він здійснює запити до бази даних для виконання операцій створення, перегляду, оновлення та видалення даних. Для реалізації цього функціоналу було використано ORM фреймворк Entity Framework Core. Завдяки його можливостям, можна просто створити всередині програми класи, що виступатимуть моделями даних, а система міграцій перетворить їх самостійно в таблиці для бази даних, та створить їх всередині неї. Такий підхід називається «Code First».

Основними компонентами цього рівня є моделі даних, контекст бази даних, міграції, репозиторії та одиниця роботи.

3.3.1 Моделі даних

Моделі даних являють собою класи C#, кожен з яких відповідає певній таблиці в базі даних. Властивості цього класу відповідають окремим стовпчикам таблиці, а кожен об’єкт такого класу загалом є програмною репрезентацією окремого запису в таблиці. Такі класи можуть містити окремі навігаційні властивості, які потрібно для зручного отримання потрібних даних і відповідають більше окремим таблицям, аніж деяким стовпчикам в ній. Окрім цього, вони можуть мати анотації даних для окремих властивостей, які

відповідають за конфігурацію роботи EF Core — вказують на первинні, зовнішні ключі тощо.

У проекті реалізовано всі моделі згідно з таблицями в базі даних, їх перелік нижче:

— Employee — основна модель, що відповідає за збереження інформації про робітника, містить навігаційні властивості для відділу, робочого статусу та стану здоров'я.

— Department — репрезентує відділ підприємства. Всередині відділів може бути різна кількість працівників.

— WorkStatus — робочий статус робітника. Один і той же статус може мати велика кількість працівників.

— HealthStatus — стан здоров'я робітника. Працює аналогічно WorkStatus.

— Attendance — являє собою відвідуваність працівника, містить інформацію про час і дату, коли працівник прийшов і покинув робоче місце.

— WorkAccident — модель для збереження свідчень про нещасні випадки, їх причини, час, коли вони відбувалися, та ін.

— EmployeeAccident — модель для зв'язування моделей робітника та нещасних випадків на роботі.

— EventLog — представляє сутність для запису даних про події в системі, їх тип, суть, час, ініціатора.

Приклад одної з подібних моделей, можна побачити на рисунку 3.2.

```
1   using System.ComponentModel.DataAnnotations;
2   using System.ComponentModel.DataAnnotations.Schema;
3   using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;
4
5   namespace HCM.DAL.Models
6   {
        20 references
7       public class Employee
8       {
9           [Key]
            13 references
10          public int Id { get; set; }
11          [Required]
            11 references
12          public string Name { get; set; }
13          [Required]
            11 references
14          public string Surname { get; set; }
            10 references
15          public string Patronym { get; set; }
16          [Required]
17          [Display(Name = "Date of birth")]
            10 references
18          public DateTime BirthDate { get; set; }
19          [Required]
20          [Display(Name = "Passport number")]
            10 references
21          public string PassportNumber { get; set; }
22          [Required]
23          [Display(Name = "Taxpayer identification number")]
            10 references
24          public string TIN { get; set; }
25          [Required]
26          [Display(Name = "Date of employment")]
            9 references
27          public DateTime HiringDate { get; set; }
28          [Required]
            11 references
29          public string Position { get; set; }
30
            10 references
31          public int DepartmentId { get; set; }
```

Рисунок 3.3 — Програмне представлення сутності Employees в додатку

3.3.2 Контекст бази даних

Контекст бази даних це центральний компонент фреймворку EF Core, який забезпечує зв'язок між моделями даних та фізичною базою даних. Програмно, він являє собою C# клас, який координує доступ до бази даних і керує операціями з даними. Окрім цього він часто містить в собі набір класів DbSet — колекцій сутностей, які використовуються для виконання операцій з базою.

В застосунку використано контекст бази даних ApplicationDbContext. Він вживається для взаємодії між моделями застосунку та бізнес логікою.

```
1   using HCM.DAL.Models;
2   using Microsoft.AspNetCore.Identity;
3   using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
4   using Microsoft.EntityFrameworkCore;
5   using System.Globalization;
6
7   namespace HCM.DAL.Data
8   {
        24 references
9       public class ApplicationDbContext : IdentityDbContext<IdentityUser>
10      {
            0 references
11          public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
12              : base(options)
13          {
14
15          }
16
            1 reference
17          public DbSet<Employee> Employees { get; set; }
            0 references
18          public DbSet<Department> Departments { get; set; }
            1 reference
19          public DbSet<Attendance> Attendances { get; set; }
            1 reference
20          public DbSet<WorkAccident> WorkAccidents { get; set;}
            1 reference
21          public DbSet<EventLog> EventLogs { get; set; }
22
            1 reference
23          public DbSet<WorkStatus> WorkStatuses { get; set; }
            1 reference
24          public DbSet<HealthStatus> HealthStatuses { get; set; }
            0 references
25          public DbSet<EmployeeAccident> EmployeeAccidents { get; set;}
26
            0 references
27          public DbSet<IdentityUser> IdentityUsers { get; set; }
```

Рисунок 3.4 — Реалізація ApplicationDbContext

3.3.3 Міграції

Міграції в EF Core є механізмом для автоматичного управління структурою бази даних під час розробки та розгортання додатків. Створивши чи змінивши необхідні моделі, достатньо лише ввести команду Add-Migration, щоб згенерувати код для створення схеми бази даних. А виконавши потім команду Update-Database всі зміни автоматично застосуються до бази даних. Міграції забезпечують атомарність операцій — якщо в міграції буде помилка, всі зміни будуть відхилені.

Це дуже зручний інструмент для відслідковування і керування змінами в базі даних, що допомагає підтримувати відповідність моделі та фізичної бази даних.

3.3.4 Репозиторії та Одиниця роботи

Як було зазначено в попередньому розділі, патерн репозиторій використовується для розділення рівня доступу до даних з рівнем бізнес-логіки завдяки створенню додаткового рівня абстракції, що полегшує підтримку та тестування проекту.

Для реалізації цього шаблону в нашому проекті було створено загальний інтерфейс IRepository (див. рис. 3.4), від якого наслідуються окремі, для кожної моделі, інтерфейси. Це потрібно для реалізації принципу розділення інтерфейсів. Згодом їх реалізують окремі класи, відповідно до власних потреб.

Зі свого боку, Одиниця роботи, який вживається для групування операцій над даними, інкапсулює всі створені репозиторії. Таким чином отримуємо центральну точку взаємодії з базою даних. Структура Одиниці роботи та фінального результату в проєкті зображені на рисунках 3.5 та 3.6.

```
1   using System.Linq.Expressions;
2
3   namespace HCM.DAL.Repository.IRepository
4   {
5       public interface IRepository<T> where T : class
6       {
            10 references
7           IEnumerable<T> GetAll(string? includeProperties = null);
            4 references
8           T Get(Expression<Func<T, bool>> filter, string? includeProperties = null);
            3 references
9           void Add(T entity);
            2 references
10          void Remove(T entity);
            1 reference
11          void RemoveRange(IEnumerable<T> entities);
12      }
13  }
```

Рисунок 3.5 — Інтерфейс IRepository

```
2   namespace HCM.DAL.Repository.IRepository
3   {
4       public interface IUnitOfWork
5       {
            2 references
6           IEmployeeRepository Employee { get; }
            4 references
7           IWorkStatusRepository WorkStatus { get; }
            4 references
8           IHealthStatusRepository HealthStatus { get; }
            4 references
9           IDepartmentRepository Department { get; }
            3 references
10          IAttendanceRepository Attendance { get; }
            6 references
11          IWorkAccidentRepository WorkAccident { get; }
            2 references
12          IEventLogRepository EventLog { get; }
13
            4 references
14          void Save();
15      }
16  }
```

Рисунок 3.6 — Інтерфейс для шаблону Unit Of Work

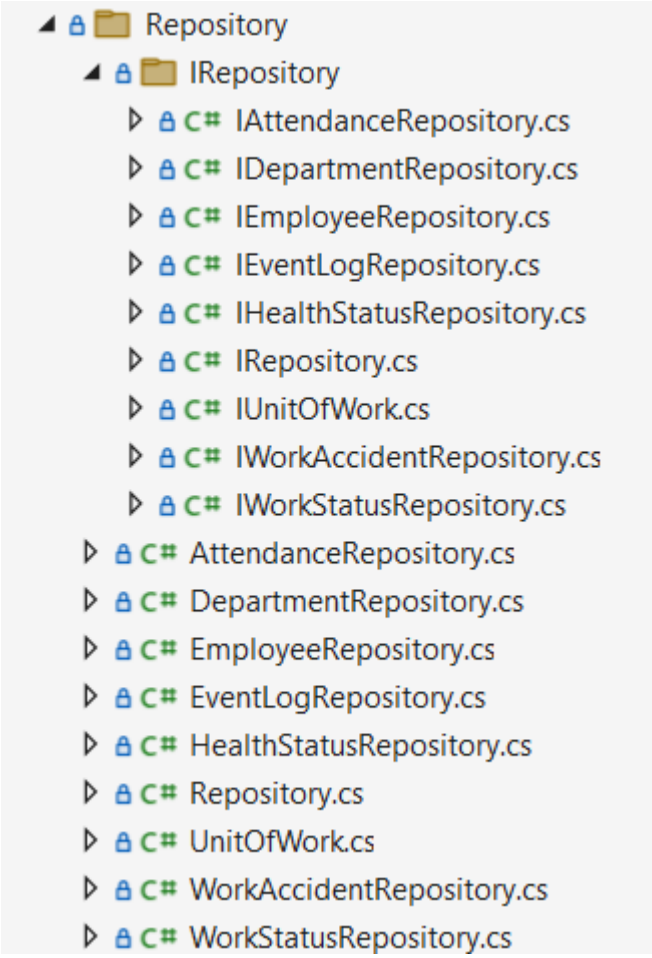


Рисунок 3.7 — Реалізація патернів в проєкті

Як результат, отримуємо готовий рівень доступу до даних, що забезпечить ефективну роботу з базою даних.

3.4 Рівень бізнес логіки

Рівень бізнес логіки забезпечує виконання бізнес-правил, необхідних для роботи додатку. Обробка даних на цьому рівні є незалежною від рівня доступу до даних та презентаційного рівня. Основними компонентами цього рівня є моделі DTO та сервіси.

3.4.1 Моделі DTO

Моделі DTO є простими об’єктами, що використовуються для передачі даних між рівнями. Вони є лише сховищами даних, без складної поведінки. Приклад такої моделі продемонстрований на рисунку 3.7.

```
3   namespace HCM.BLL.DTOs
4   {
        15 references
5       public record EmployeeDTO
6       {
            2 references
7           public int Id { get; set; }
            0 references
8           public string Name { get; set; }
            0 references
9           public string Surname { get; set; }
            0 references
10          public string Patronym { get; set; }
            0 references
11          public DateTime BirthDate { get; set; }
            0 references
12          public string PassportNumber { get; set; }
            0 references
13          public string TIN { get; set; }
            1 reference
14          public DateTime HiringDate { get; set; }
            0 references
15          public string Position { get; set; }
```

Рисунок 3.8 — Приклад частини моделі EmployeeDTO

3.4.2 Сервіси

Сервіси це класи, які створені для виконання специфічних бізнес операцій. Вони залежать від рівня доступу до даних, адже їм потрібно виконувати операції з певними даними, і можуть залежати від інших сервісів для реалізації своїх функцій.

```
9   namespace HCM.BLL.Services
10  {
        1 reference
11      public class EmployeeService : IEmployeeService
12      {
13          private IUnitOfWork _database;
14          private IMapper _mapper;
15
            0 references
16          public EmployeeService(IUnitOfWork database, IMapper mapper)
17          {
18              _database = database;
19              _mapper = mapper;
20          }
21
            1 reference
22          public void AddEmployee(EmployeeDTO employee)
23          {
24              if (employee.Id != 0)
25              {
26                  employee.HiredById = currentUser;
27                  employee.HiringDate = DateTime.Now;
28                  _database.Employee.Add(_mapper.Map<Employee>(employee));
29                  _database.Employee.Save();
30              }
31              else
32              {
33                  throw new InvalidDataException("Employee id cannot be zero");
34              }
35          }
36
            1 reference
37          public void UpdateEmployee(EmployeeDTO employee)
38          {
39              if (employee.Id != 0)
40              {
41                  _database.Employee.Update(_mapper.Map<Employee>(employee));
42                  _database.Employee.Save();
43              }
44              else
```

Рисунок 3.9 — Приклад частини сервісу працівника EmployeeService

3.5 Візуальний рівень

Візуальний рівень або рівень презентації — це той рівень, на якому реалізується логіка взаємодії системи з користувачем. В нашому проекті він реалізований за допомогою фреймворку для створення веб-додатків ASP.NET Core MVC. Оскільки це веб-додаток, що працює в браузері, вся взаємодія між

складовими частинами (контролерами та файлами представлення) відбувається з використанням HTTP запитів.

Окрім цього, цей рівень реалізує архітектурний шаблон MVC, який де-факто стандартом в розробці веб-додатків. З цього ми отримуємо, що ключовими елементами даного рівня є контролери, файли представлення та моделі представлення. Крім них є і інші файли, як от CSS стилі, JavaScript скрипти, але окремо на них зосереджуватись не будемо.

3.5.1 Контролери

Контролер є тим механізмом, що обробляє вхідні HTTP запити. Він отримує дані з моделі й відображає представлення у відповідь на запити, виконуючи відповідну бізнес-логіку.

```
1   using HCM.BLL.DTOs;
2   using HCM.BLL.Interfaces;
3   using HCM_Web.ViewModels;
4   using Microsoft.AspNetCore.Mvc;
5
6   namespace HCM_Web.Controllers
7   {
        1 reference
8       public class EmployeeController : Controller
9       {
10          private readonly IEmployeeService _employeeService;
11          private readonly IMapper _mapper;
                0 references
12          public EmployeeController(IEmployeeService employeeService, IMapper mapper)
13          {
14              _employeeService = employeeService;
15              _mapper = mapper;
16          }
                0 references
17          public IActionResult Index()
18          {
19              return View();
20          }
21
22          [HttpPost]
                0 references
23          public IActionResult Upsert(EmployeeVM employeeVM)
24          {
25              if (ModelState.IsValid)
26              {
27                  if (employeeVM.Employee.Id == 0) //create
28                  {
29                      _employeeService.AddEmployee(_mapper.Map<EmployeeDTO>(employeeVM));
30                      TempData["success"] = "Employee added successfully";
31                  }
32                  else //update
33                  {
34                      _employeeService.UpdateEmployee(_mapper.Map<EmployeeDTO>(employeeVM));
35                      TempData["success"] = "Employee updated successfully";
36                  }
37
38                  return RedirectToAction("Index");
```

Рисунок 3.10 — Приклад частини контролера EmployeeController

3.5.2 Представлення

В ASP.NET Core MVC представлення є окремим файлом, який містить код користувацького інтерфейсу. В ньому використовується мова розмітки HTML з використанням Razor — спеціального движка представлень, який дозволяє комбінувати код HTML та код на мові C#. Такий файл має розширення ".cshtml" і використовується для відображення даних додатку, зазвичай однієї зі сторінок чи її частини. Приклад такого файлу можна побачити на рисунку 3.10:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - HCM Web</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min-flatly.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
    <link rel="stylesheet" href="~/HCM_Web.styles.css" asp-append-version="true" />
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/bootstrap-icons.min.css">
    <link rel="stylesheet" href="https://cdn.datatables.net/2.0.6/css/dataTables.dataTables.min.css" />
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/css/toastr.min.css" />
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm bg-primary border-bottom box-shadow mb-3" data-bs-theme="dark">
            <div class="container-fluid">
                <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">HCM Web</a>
                <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-controls="navbarSupportedContent"
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
                    <ul class="navbar-nav me-auto">
                        <li class="nav-item">
                            <a class="nav-link" asp-area="" asp-controller="Employee" asp-action="Index">Employee</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link" asp-area="" asp-controller="Attendance" asp-action="Index">Attendance</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link" asp-area="" asp-controller="WorkAccident" asp-action="Index">Work Accidents</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link" asp-area="" asp-controller="Department" asp-action="Index">Departments</a>
                        </li>
                        <li class="nav-item dropdown">
                            <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown" aria-expanded="false">
                                Info Management
                            </a>
                            <ul class="dropdown-menu">
```

Рисунок 3.11 — Приклад частини файлу представлення, розмітка головної сторінки

Для покращення користувацького досвіду при роботі з додатком, для представлень було використано JavaScript бібліотеку "**toastr**". Вона використовується для створення коротких спливаючих повідомлень. Демонстрація роботи зображена на рисунку 3.11.

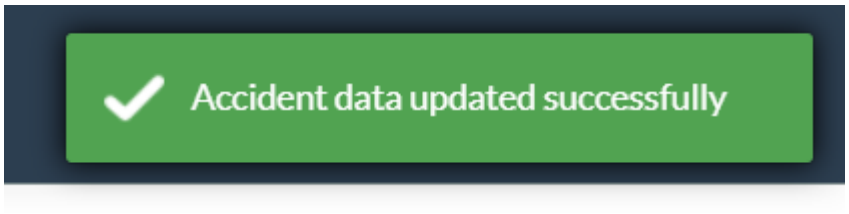


Рисунок 3.12 — Демонстрація роботи toastr

Крім неї, також було додано бібліотеку “**sweetalert2**” — вона дозволяє створювати красиві, адаптивні та настроювані спливаючі вікна для різних подій. Приклад подібного вікна (рис. 3.12):

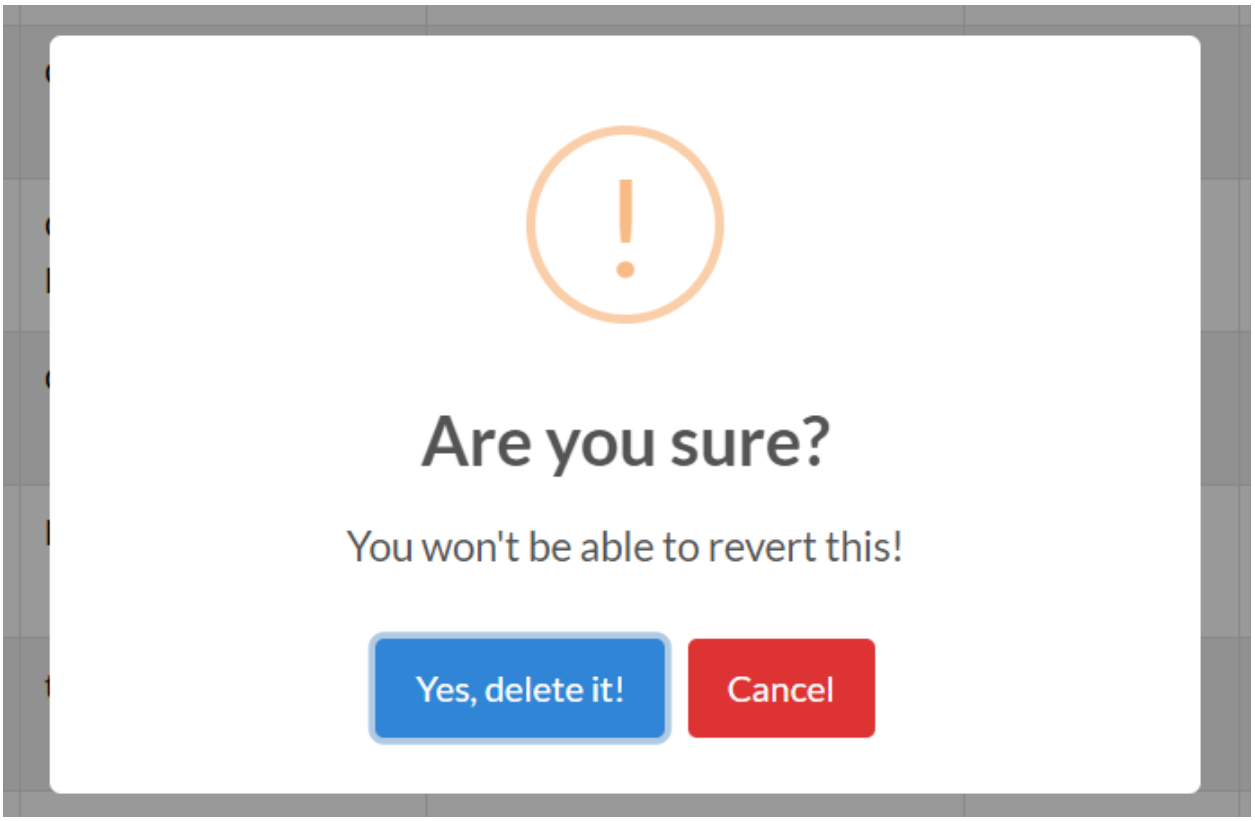


Рисунок 3.13 — Демонстрація вікна sweetalerts

Як результат, завдяки використанню розглянутих вище технологій та бібліотек, вдалося створити зручний користувацький інтерфейс, що активно реагує на дії користувача, надаючи йому постійний зворотній зв’язок. Це поліпшує якісь взаємодії користувачів із системою та, як наслідок, робить їхню роботу ефективніше.

3.6 Демонстрація роботи

Після того, як розібрались з будовою застосунку, перейдемо до розгляду функціоналу додатку. Для перегляду інформації про працівників, було створено окрему сторінку (рис. 3.13). На ній є інтерактивна таблиця зі списком усіх працівників. Записи в таблиці можна сортувати за прізвищем та ім'ям працівників, посадою тощо. Можна здійснювати пошук по потрібним параметрам, наприклад, за назвою відділу (рис. 3.14). Також в таблиці є кнопки для взаємодії із записами: оранжева для редагування, червона — видалення.

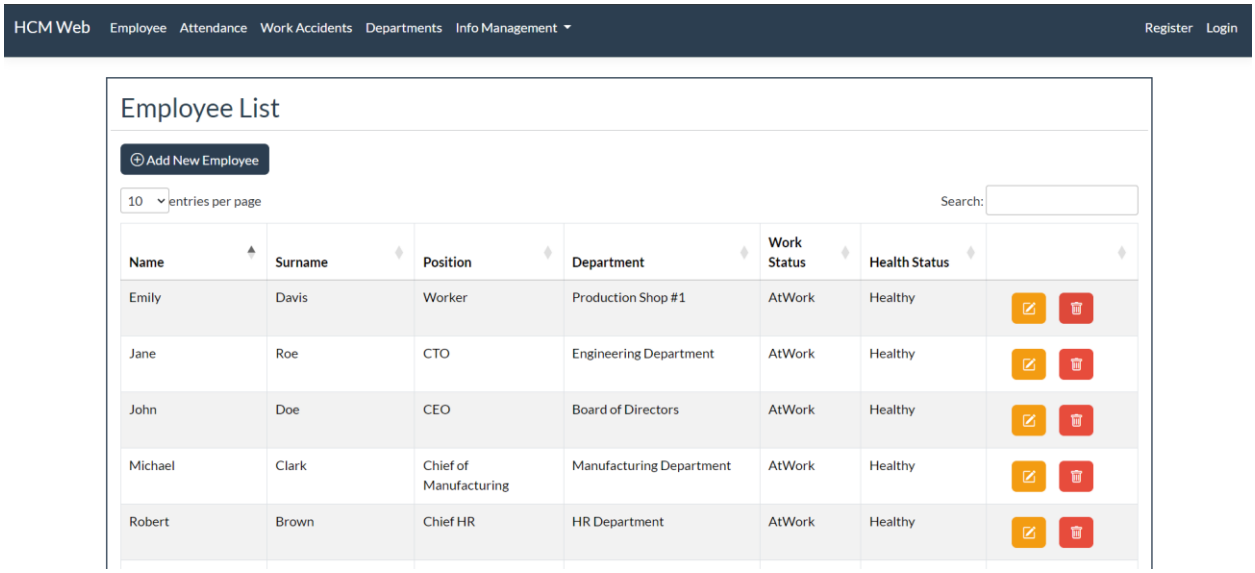


Рисунок 3.14 — Сторінка перегляду даних про працівників

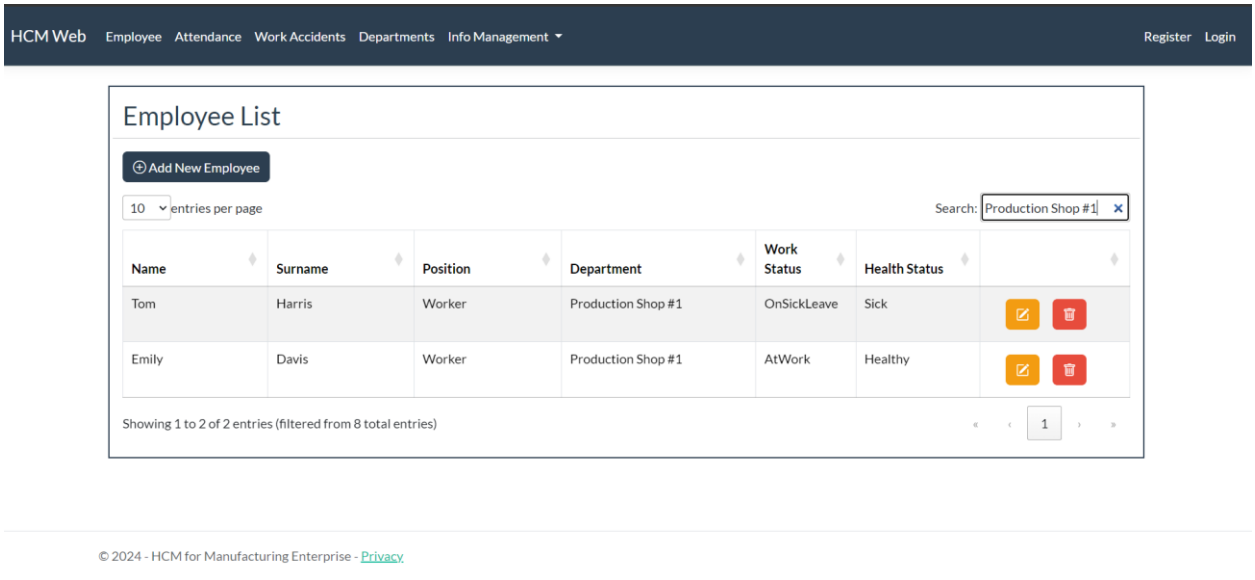


Рисунок 3.15 — Пошук працівників за назвою відділу

Для додавання нового працівника, потрібно натиснути на відповідну кнопку, яка перенесе на окрему сторінку з формою для заповнення даних. Якщо при заповненні даних не зазначити якесь поле, система валідації вкаже на нього, підписавши червоним текстом (рис 3.15).

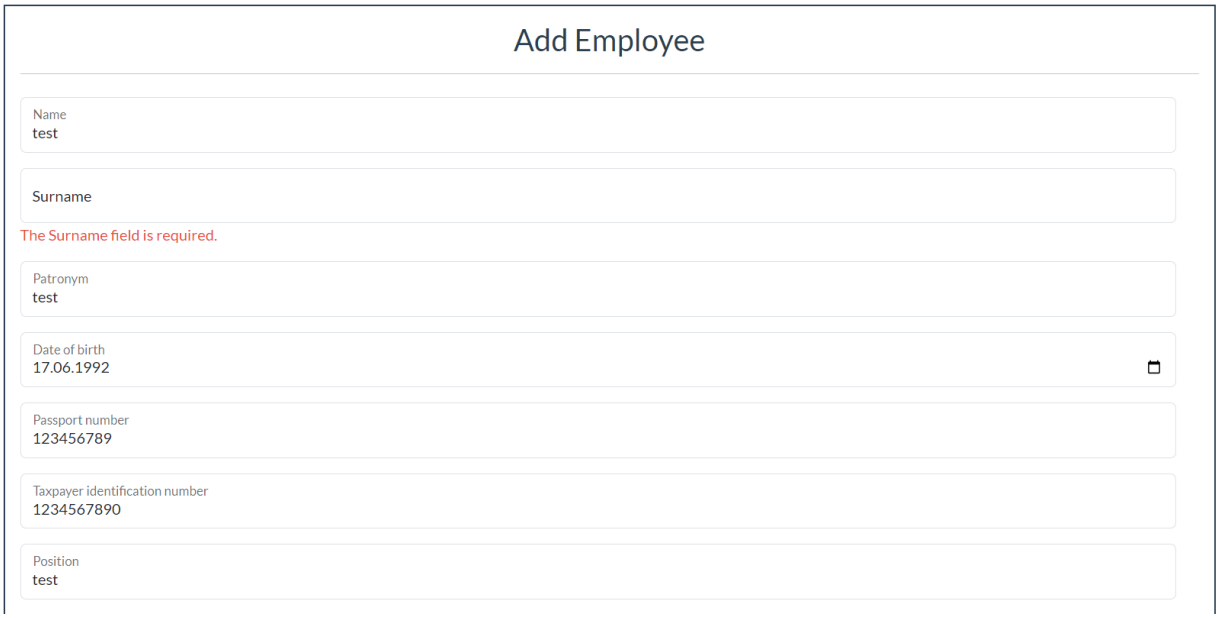


Рисунок 3.16 — Форма додавання працівників з валідацією

У формі реалізовані випадаючі списки для вибору відділу, в якому має працювати робітник, вибор робочого статусу і станів здоров'я (рис. 3.16).

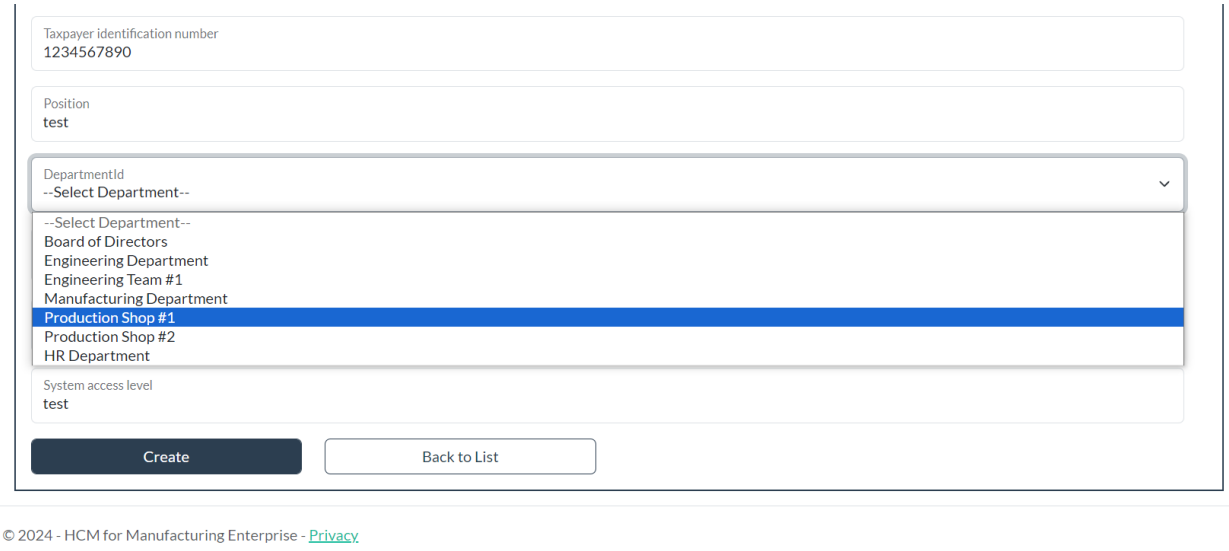


Рисунок 3.17 — Випадаючий список відділів у формі

Для зміни даних про працівника, необхідно натиснути на кнопку редагування. Після успішного збереження змін, повідомлення про це з’явиться у верхньому правому куті (рис. 3.17).

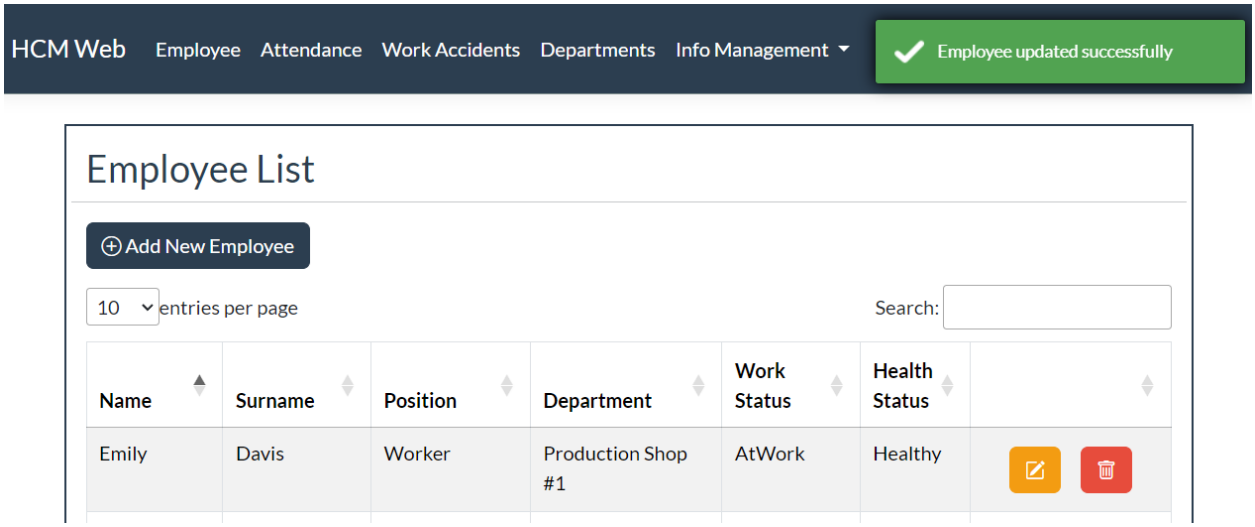


Рисунок 3.18 — Повідомлення про успішне оновлення даних

Для видалення робітника слід натиснути кнопку видалення. Після цього з’явиться спливаюче вікно, яке запитає підтвердження для здійснення операції (рис. 3.18). Якщо таке підтвердження буде надане, запис із таблиці зникне.

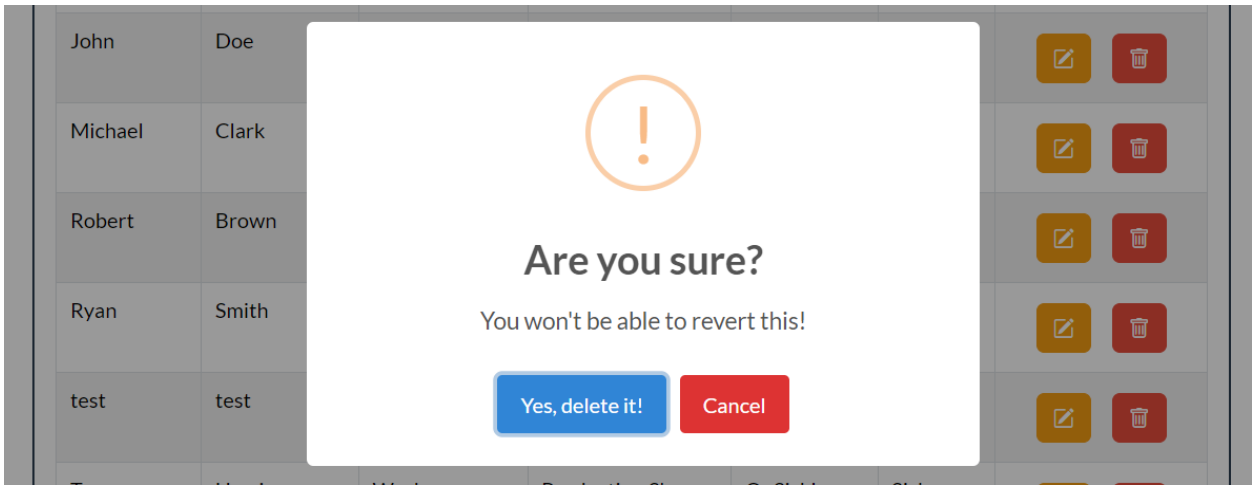


Рисунок 3.19 — Вспливаюче вікно при видаленні робітника

Аналогічним чином реалізована взаємодія і з іншими аспектами обліку, як-то, наприклад, відвідуваність чи нещасні випадки (рис 3.19-3.20).

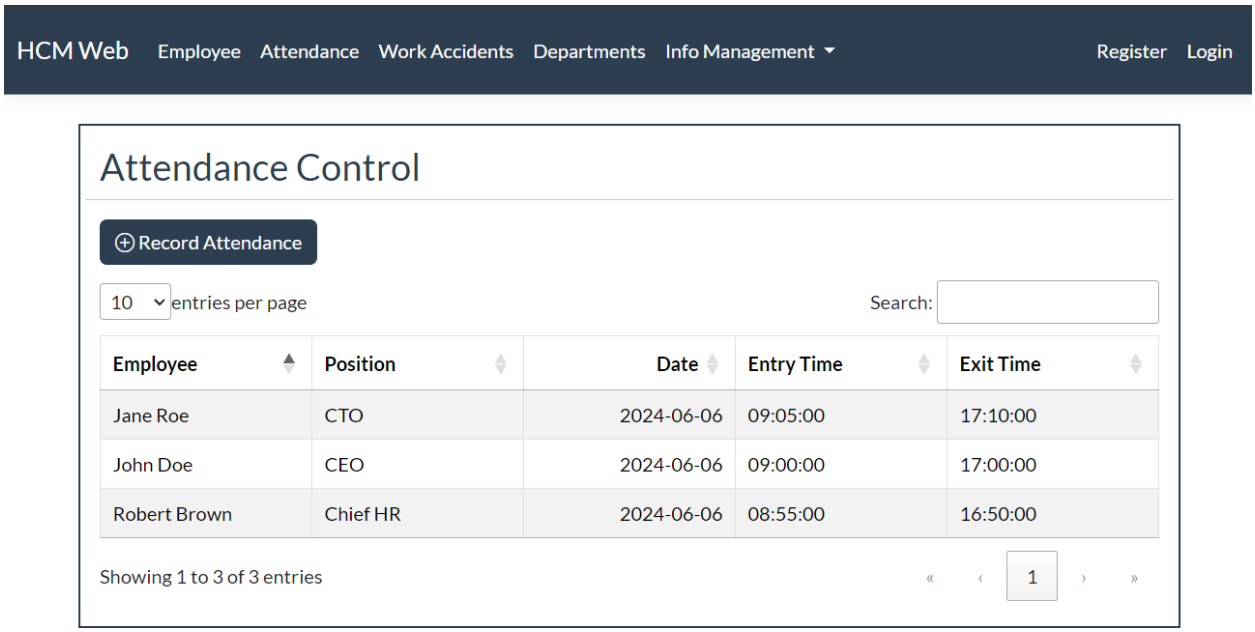


Рисунок 3.20 — Сторінка перегляду відвідуваності

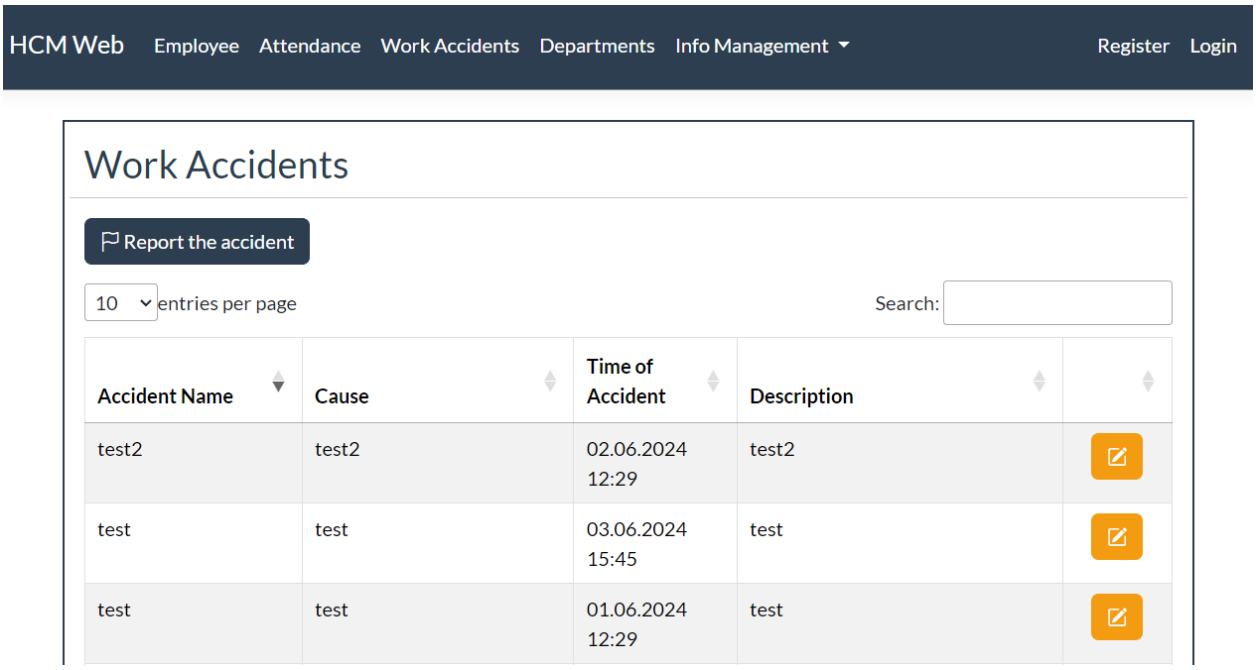


Рисунок 3.21 — Сторінка перегляду інформації про нещасні випадки

Також було реалізовано функціонал автентифікації. Для цього створені окремі сторінки для реєстрації та входу користувачів в систему, які містять валідацію введених даних, інші сторінки. Їх зображено на рисунках 3.21 та 3.22.

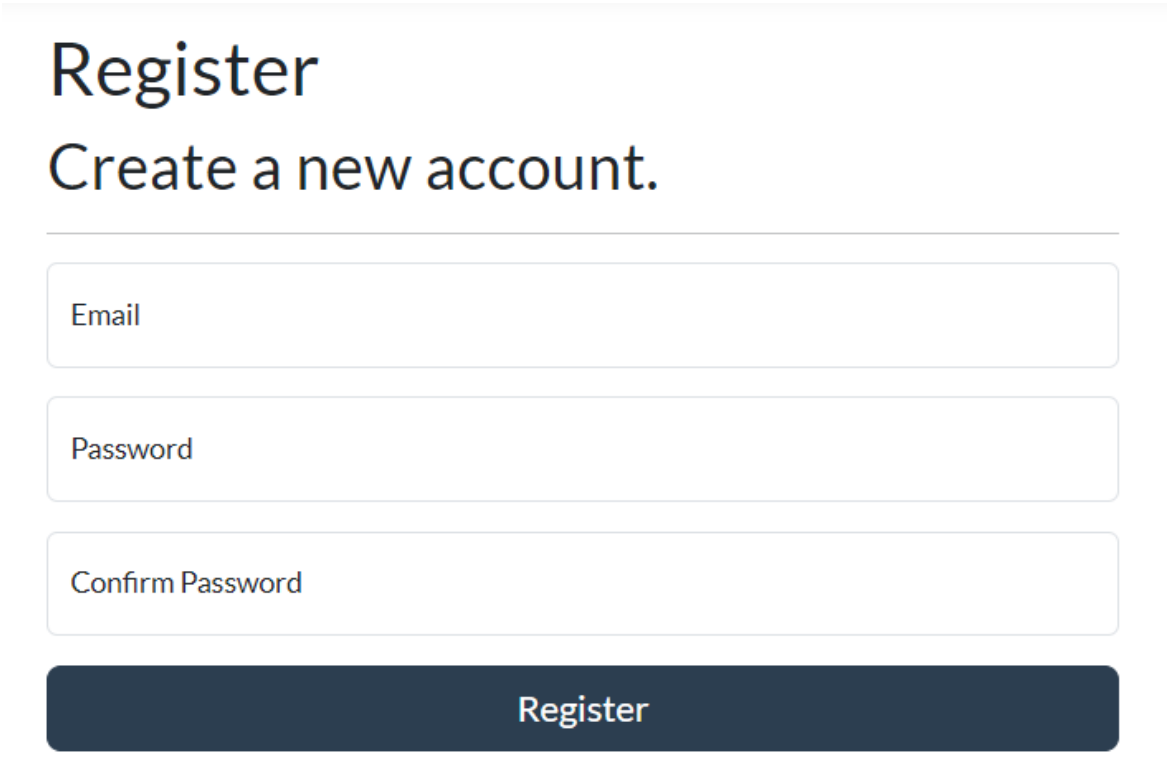


Рисунок 3.22 — Форма реєстрації

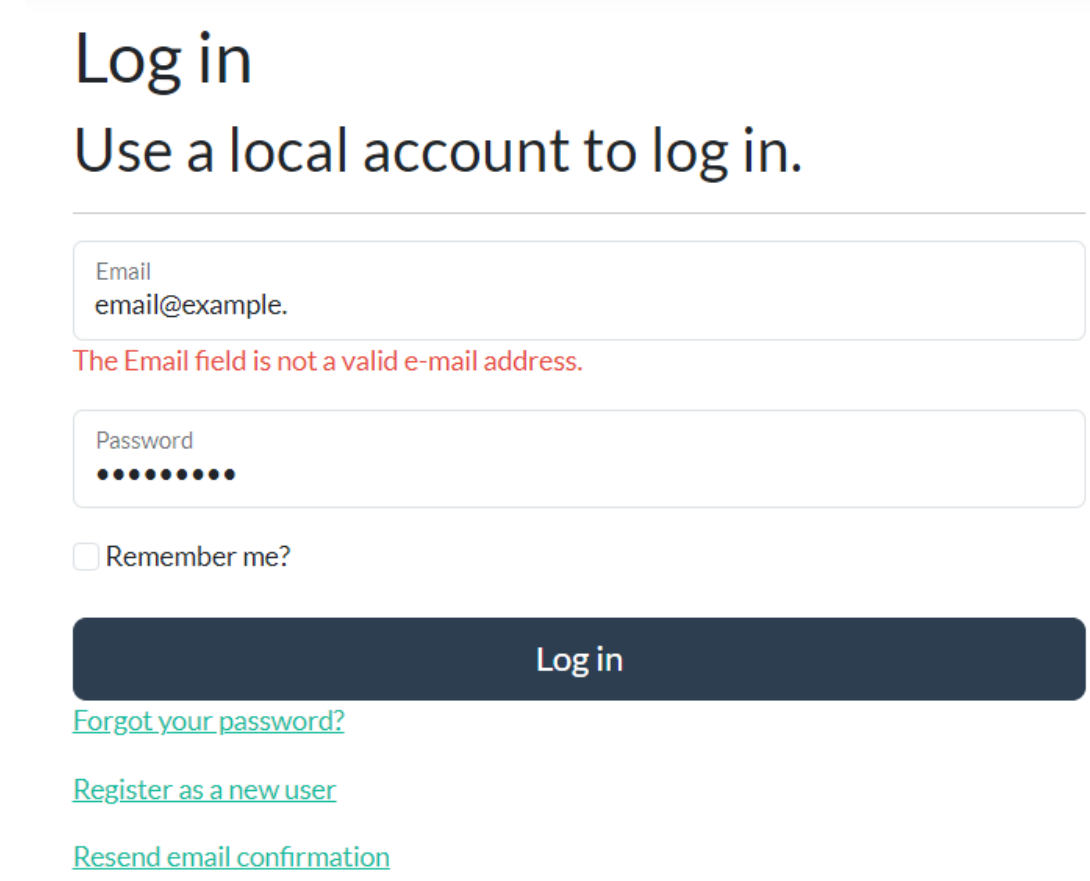


Рисунок 3.23 — Форма входу з демонстрацією валідації даних

ВИСНОВОК ДО РОЗДІЛУ 3

В даному розділі було досліджено і обговорено всі елементи розробленої системи, починаючи від організації бази даних і до конкретної реалізації задуманої архітектури застосунку. Було продемонстровано роботу готового додатку з його функціональними можливостями.

База даних займає одне з основних місць про розробці системи, адже вона є фундаментом всього програмного рішення. Від правильної її реалізації залежить робота всього додатку. Тому було приділено велику увагу до її проектування, розглянуто будову всіх сутностей та зв'язків між ними.

Наступним елементом, що був дослідженим, стала архітектура додатку. В застосунку було реалізовано тришарову архітектуру, яка має забезпечити еластичність і масштабованість системи, для адаптації до мінливих умов ринку.

Першим шаром, що був розглянутий, став рівень доступу до даних. Він використовує Entity Framework Core для взаємодії з базою даних, реалізує набір класів-моделей, які є програмними відповідниками сутностям бази даних. Для легкості впровадження змін в структуру БД, використано систему міграцій. Реалізація шаблонів Репозиторій та Одиниця роботи забезпечує чітке розділення шарів, та полегшує тестування.

Рівень бізнес-логіки здійснює виконання бізнес-операцій. Він надає сервіси, що використовують попередній рівень для отримання доступу до даних та здійснюють їх обробку.

Презентаційний рівень відповідає за взаємодію системи з користувачем. Для його втілення було використано технологію ASP.NET Core MVC, в якій контролери обробляють користувацькі запити, а представлення представляють користувацький інтерфейс.

В результаті, отримуємо зручний застосунок з достатньо широким функціоналом, який можна легко розширити і адаптувати.

ВИСНОВОК

Упродовж роботи над дипломним проєктом було спроєктовано та успішно розроблено систему для обліку працівників виробничого підприємства. Система надає необхідні функції для проведення обліку, які задовольняють потреби ринку та запити користувачів. Робота складається з трьох основних розділів, які розглядають окремі аспекти створення системи.

Перший розділ було присвячено вивченню предметної області та аналізу сфери існуючих програмних продуктів по менеджменту HR. Було визначено що таке облік кадрів, в чому полягають завдання обліку та що має обліковуватись. Проведено вивчення та порівняння основних гравців на ринку. На основі отриманої інформації сформовано вимоги, яким має відповідати наше рішення.

У другому розділі було описано структуру майбутньої системи, визначено, що вона буде реалізовувати монолітний веб-додаток з тришаровою архітектурою. Було порівняно вибір десктопного і веб-додатків, та обґрунтовано вибір варіанту веб-додатку легкістю розробки під одну платформу, можливістю швидкого розгортанні і впровадження оновлень, доступністю роботи на будь-якій системі, що підтримує з'єднання до мережі інтернет та має веб-переглядач. Досліджено і альтернативу монолітній архітектурі застосунку — мікросервісну. Але вибір все одно було зроблено на користь монолітної архітектури завдяки простоті розробки та бистроті впровадження нових функцій. Зі схожих причин обрано і тришарову архітектуру додатку. Такий підхід забезпечує модульність, масштабованість та підтримку додатку. В якості платформи, на якій буде розроблятись система, вибрано платформу .NET та мову C# через їх сучасність, широкий набір інструментів і можливостей, що добре інтегруються в одну екосистему, високу затребуваність на ринку. Провайдером бази даних обрано Microsoft SQL

Server. Він легко інтегрується з вибраною платформою розробки і забезпечує надійне сховище для даних.

Третій розділ стосувався безпосередньої розробки системи. В ньому було розглянуто всі аспекти побудови бази даних, реалізації різних шарів тришарової архітектури, продемонстровано роботу додатку. Правильна структура бази даних забезпечить надійне зберігання даних і відсутність можливих помилок. А реалізація різних функцій в окремих шарах створить основу для надійної роботи застосунку.

В результаті роботи було отриманий комплексний додаток для обліку працівників, який має весь необхідний функціонал по роботі з працівниками, відвідуваністю, нещасними випадками, має систему логування змін в системі. Готовий продукт відповідає сучасним вимогами ринку подібних рішень, та може бути розгорнутий для використання в реальному середовищі. Плоди даної роботи можуть бути використані для наступної модифікації і адаптації до потреб окремих користувачів. Технічне завдання на дипломний бакалаврський проєкт виконано повністю.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Human Capital Management Market Size, Share, Industry Trends, & Statistics - 2030. *MarketsandMarkets*. [Електронний ресурс] – URL: https://www.marketsandmarkets.com/Market-Reports/human-capital-management-market-193746782.html (дата звернення: 20.05.2024).
2. Contributors to Wikimedia projects. Human resources - Wikipedia. *Wikipedia, the free encyclopedia*. [Електронний ресурс] – URL: https://en.wikipedia.org/wiki/Human_resources (дата звернення: 20.05.2024).
3. What is SAP SuccessFactors? A Look at Cloud HR with SAP | SAP PRESS. *Learn SAP Fundamentals with SAP PRESS | SAP Solution Overviews*. [Електронний ресурс] – URL: https://learning.sap-press.com/sap-successfactors (дата звернення: 20.05.2024).
4. Iwuozor J. SuccessFactors review (2024): features, pros & cons. *Forbes Advisor*. [Електронний ресурс] – URL: https://www.forbes.com/advisor/business/software/successfactors-review/ (дата звернення: 20.05.2024)
5. Comparing Workday, Oracle HCM Cloud & SAP SuccessFactors for large businesses. *ROCKCREST*. [Електронний ресурс] – URL: https://www.rockcrest.com/insights/comparing-workday-oracle-hcm-cloud-and-sap-successfactors-for-large-businesses- (дата звернення: 20.05.2024).
6. Introducing insights dashboard and homepage redesign | BambooHR. *BambooHR: The Complete HR Software for People, Payroll & Benefits*. [Електронний ресурс] – URL: https://www.bamboohr.com/blog/bamboohr-insights-dashboard-homepage-redesign (дата звернення: 20.05.2024).

7. Holznienkemper L. BambooHR review (2024): features, pricing & more. *Forbes Advisor*. [Електронний ресурс] – URL: https://www.forbes.com/advisor/business/software/bamboohr-review/ (дата звернення: 20.05.2024).
8. BambooHR review (2024): pricing, features, pros and cons. *TechRepublic*. [Електронний ресурс] –URL: https://www.techrepublic.com/article/bamboohr-review/ (дата звернення: 20.05.2024).
9. Zoho People review – pricing, comparisons, and FAQs. *The SMB Guide*. [Електронний ресурс] – URL: https://www.thesmbguide.com/zoho-people-reviews (дата звернення: 21.05.2024).
10. Workday review: pricing, features, pros and cons. *TechRepublic*. [Електронний ресурс] – URL: https://www.techrepublic.com/article/workday-review/ (дата звернення: 21.05.2024).
11. 20 pros and cons of Workday - EducationalWave. *EducationalWave.* [Електронний ресурс] – URL: https://www.luxwisp.com/pros-and-cons-of-workday/ (дата звернення: 21.05.2024).
12. Global HR management system | Workday. *Workday Enterprise Management Cloud | Finance, HR, Planning, Spend | Workday US*. [Електронний ресурс] – URL: https://www.workday.com/en-us/products/human-capital-management/human-resource-management.html (дата звернення: 21.05.2024).
13. Rubin D. The three layered architecture. *Medium*. [Електронний ресурс] – URL: https://medium.com/@deanrubin/the-three-layered-architecture-fe30cb0e4a6 (дата звернення: 22.05.2024).
14. Documenting software architectures: Views and beyond / [C. Paul, B. Felix, B. Len та ін.]. – Upper Saddle River, NJ: Addison-Wesley, 2011. – 537 с. – (2).
15. Web apps vs. native apps vs. hybrid apps - difference between types of web and mobile applications - AWS. *Amazon Web Services, Inc.* [Електронний ресурс] – URL: https://aws.amazon.com/compare/the-difference-between-web-apps-native-apps-and-hybrid-apps/?nc1=h_ls (дата звернення: 22.05.2024).

16. Beznos M. Microservices vs monolithic architecture: how each of them can impact your business?. *Software Development Company - N-iX*. [Електронний ресурс] – URL: https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/ (дата звернення: 22.05.2024).
17. Saraç A. What is repository pattern?. *LinkedIn: Log In or Sign Up*. [Електронний ресурс] – URL: https://www.linkedin.com/pulse/what-repository-pattern-alper-saraç (дата звернення: 22.05.2024).
18. Implementing the repository and unit of work patterns in an ASP.NET MVC application (9 of 10). *Microsoft Learn: Build skills that open doors in your career*. [Електронний ресурс] – URL: https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application (дата звернення: 22.05.2024).
19. Introduction to .NET - .NET. *Microsoft Learn: Build skills that open doors in your career*. [Електронний ресурс] – URL: https://learn.microsoft.com/en-us/dotnet/core/introduction (дата звернення: 23.05.2024).
20. Overview - A tour of C#. *Microsoft Learn: Build skills that open doors in your career*. [Електронний ресурс] – URL: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview (дата звернення: 23.05.2024).
21. Overview of ASP.NET Core MVC. *Microsoft Learn: Build skills that open doors in your career*. [Електронний ресурс] – URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-8.0 (дата звернення: 23.05.2024).
22. Overview of Entity Framework Core - EF Core. *Microsoft Learn: Build skills that open doors in your career*. [Електронний ресурс] – URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 23.05.2024).

ДОДАТОК 1

Система обліку працівників виробничого підприємства

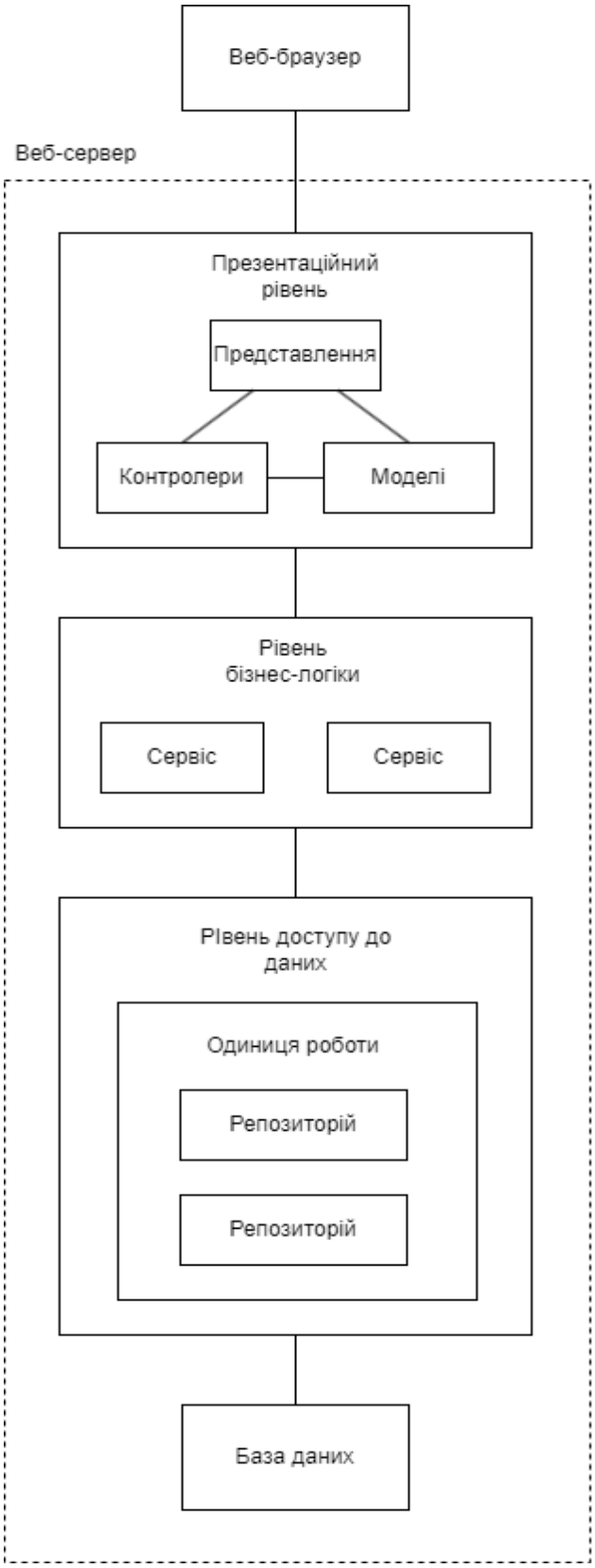
Структурна схема системи

(Схема структурна)

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ – 2024



					ІАЛЦ.467200.004 Д1		
Зм	Лист	№ докум.	Підп.	Дата			
Розроб.		Луценко В.В.			*Система обліку працівників виробничого підприємства*	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Сімоненко А.В.					
Реценз.					Структурна схема системи (схема структурна)	***КПІ ім. Ігоря Сікорського, ФІОТ, гр. ІО-01***	
Н. контр.		Виноградов Ю.М.					
Затверд.							

ДОДАТОК 2

Система обліку працівників виробничого підприємства

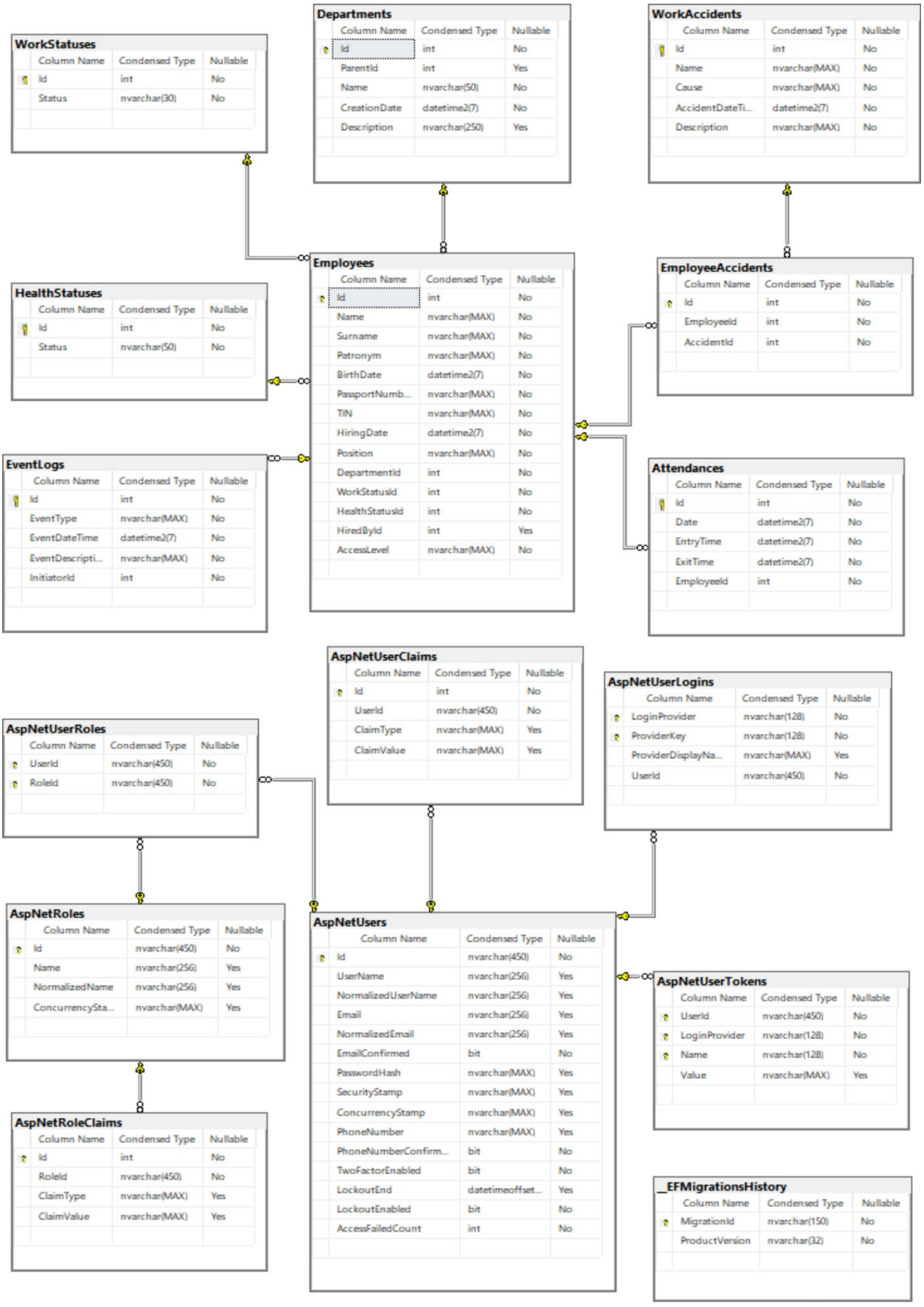
Діаграма даних

(Схема функціональна)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ – 2024



					ІАЛЦ.467200.005 Д2		
Зм	Лист	№ докум.	Підп.	Дата			
Розроб.		Луценко В.В.			*Система обліку працівників виробничого підприємства*	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Сімоненко А.В.					
Реценз.							
Н. контр.		Виноградов Ю.М.			Діаграма даних (схема функціональна)		
Затверд.							

КПІ ім. Ігоря Сікорського, ФІОТ, гр. ІО-01

ДОДАТОК 3

Система обліку працівників виробничого підприємства

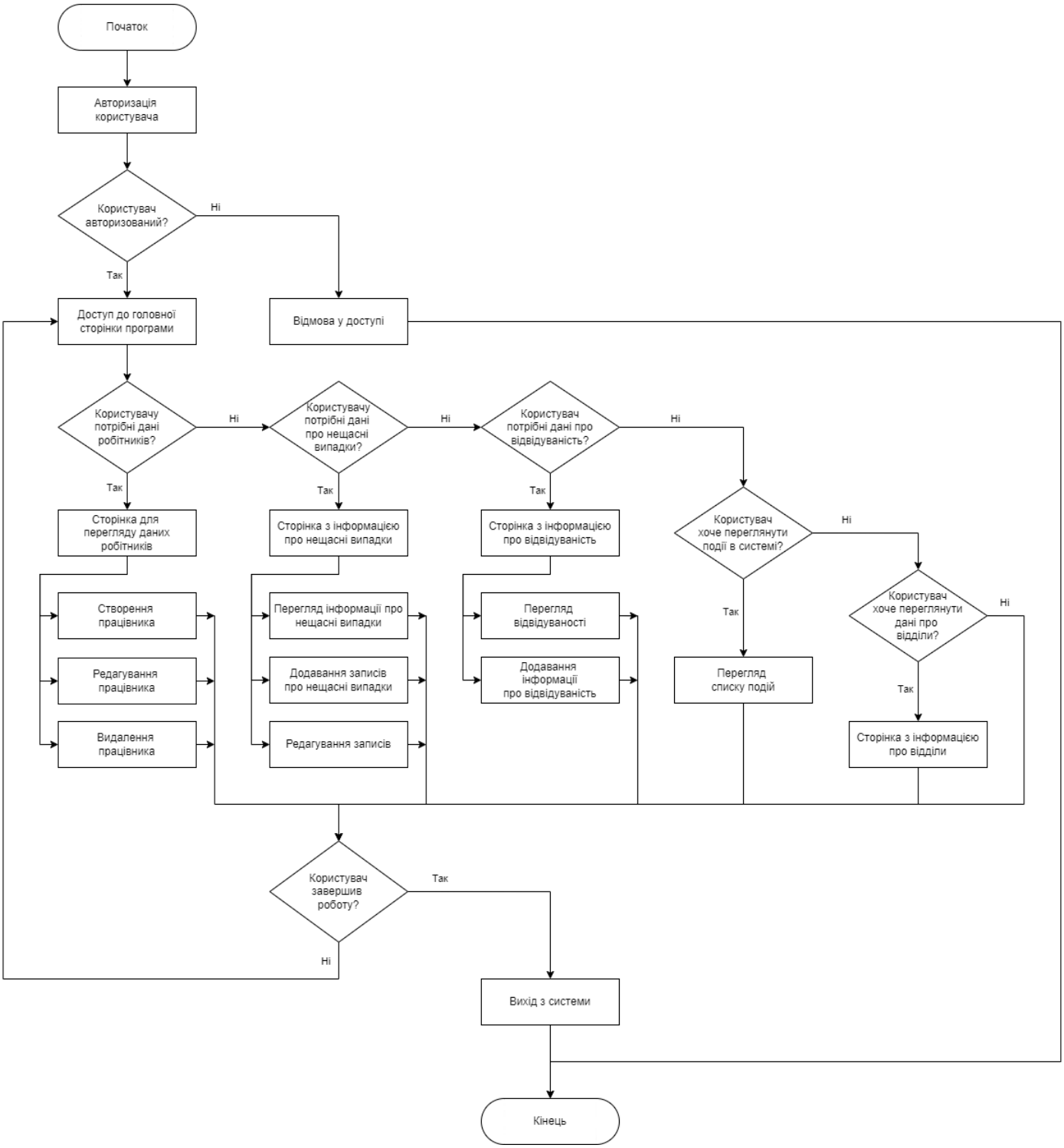
Схема алгоритму

(Схема принципова)

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ – 2024



					ІАЛЦ.467200.006 ДЗ		
Зм	Лист	№ докум.	Підп.	Дата			
Розроб.		Луценко В.В.			*Система обліку працівників виробничого підприємства*	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Сімоненко А.В.					
Реценз.					Схема алгоритму (схема принципова)	***КПІ ім. Ігоря Сікорського, ФІОТ, гр. ІО-01***	
Н. контр.		Виноградов Ю.М.					
Затверд.							

ДОДАТОК 4

Система обліку працівників виробничого підприємства

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 14

Київ – 2024

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;

namespace HCM.DAL.Models
{
    public class Employee
    {
        [Key]
        public int Id { get; set; }
        [Required]
        public string Name { get; set; }
        [Required]
        public string Surname { get; set; }
        public string Patronym { get; set; }
        [Required]
        [Display(Name = "Date of birth")]
        public DateTime BirthDate { get; set; }
        [Required]
        [Display(Name = "Passport number")]
        public string PassportNumber { get; set; }
        [Required]
        [Display(Name = "Taxpayer identification number")]
        public string TIN { get; set; }
        [Required]
        [Display(Name = "Date of employment")]
        public DateTime HiringDate { get; set; }
        [Required]
        public string Position { get; set; }

        public int DepartmentId { get; set; }
        [ForeignKey("DepartmentId")]
        [ValidateNever]
        public Department Department { get; set; }

        public int WorkStatusId { get; set; }
        [ForeignKey("WorkStatusId")]
        [ValidateNever]
        public WorkStatus WorkStatus { get; set;}

        public int HealthStatusId { get; set; }
        [ForeignKey("HealthStatusId")]
        [ValidateNever]
        public HealthStatus HealthStatus { get; set;}

        public int? HiredById { get; set; }
        [Required]
        [Display(Name = "System access level")]
        public string AccessLevel { get; set; }

    }
}
```

```
using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace HCM.DAL.Models
{
    public class Department
    {
        [Key]
        public int Id { get; set; }
        public int? ParentId { get; set; }
        [ForeignKey("ParentId")]
        [ValidateNever]
        public Department Parent { get; set; }
        [Required, MaxLength(50)]
        [Display(Name = "Department name")]
        public string Name { get; set; }
        [Required]
        [Display(Name = "Date of creation")]
        public DateTime CreationDate { get; set; }
        [MaxLength(250)]
        public string? Description { get; set; }
    }
}
```

```
using System.ComponentModel.DataAnnotations;

namespace HCM.DAL.Models
{
    public class WorkStatus
    {
        [Key]
        public int Id { get; set; }
        [Required, MaxLength(30)]
        [Display(Name = "Work status")]
        public string Status { get; set; }
    }
}
```

```
using System.ComponentModel.DataAnnotations;

namespace HCM.DAL.Models
{
    public class HealthStatus
    {
        [Key]
        public int Id { get; set; }
        [Required, MaxLength(50)]
        [Display(Name = "Health status")]
        public string Status { get; set; }
    }
}
```

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace HCM.DAL.Models
{
```

```
    public class Attendance
    {
        [Key]
        public int Id { get; set; }
        [Required]
        public DateTime Date { get; set; }
        public DateTime EntryTime { get; set; }
        public DateTime ExitTime { get; set; }

        public int EmployeeId { get; set; }
        [ForeignKey("EmployeeId")]
        public Employee Employee { get; set; }
    }
}


using System.ComponentModel.DataAnnotations;

namespace HCM.DAL.Models
{
    public class WorkAccident
    {
        [Key]
        public int Id { get; set; }
        [Required]
        public string Name { get; set; }
        [Required]
        [Display(Name = "Cause of the accident")]
        public string Cause { get; set; }
        [Required]
        [Display(Name = "Time and date of the accident")]
        public DateTime AccidentDateTime { get; set; }
        [Required]
        public string Description { get; set; }
    }
}


using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace HCM.DAL.Models
{
    public class EmployeeAccident
    {
        [Key]
        public int Id { get; set; }

        [Required]
        public int EmployeeId { get; set; }
        [ForeignKey(nameof(EmployeeId))]
        public Employee Employee { get; set; }

        [Required]
        public int AccidentId { get; set; }
        [ForeignKey(nameof(AccidentId))]
        public WorkAccident Accident { get; set; }
    }
}


using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
```

```
namespace HCM.DAL.Models
{
    public class EventLog
    {
        [Key]
        public int Id { get; set; }
        [Required]
        public string EventType { get; set; }
        [Required]
        public DateTime EventDateTime { get; set; }
        public string EventDescription { get; set; }

        public int InitiatorId { get; set; }
        [ForeignKey("InitiatorId")]
        public Employee Employee { get; set; }

    }
}


using HCM.DAL.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using System.Globalization;

namespace HCM.DAL.Data
{
    public class ApplicationDbContext : IdentityDbContext<IdentityUser>
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {

        }

        public DbSet<Employee> Employees { get; set; }
        public DbSet<Department> Departments { get; set; }
        public DbSet<Attendance> Attendances { get; set; }
        public DbSet<WorkAccident> WorkAccidents { get; set;}
        public DbSet<EventLog> EventLogs { get; set; }

        public DbSet<WorkStatus> WorkStatuses { get; set; }
        public DbSet<HealthStatus> HealthStatuses { get; set; }
        public DbSet<EmployeeAccident> EmployeeAccidents { get; set;}

        public DbSet<IdentityUser> IdentityUsers { get; set; }
```

```
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);


            modelBuilder.Entity<Employee>().ToTable(tb =>
tb.UseSqlOutputClause(false));
        }
    }
}


using System.Linq.Expressions;

namespace HCM.DAL.Repository.IRepository
{
    public interface IRepository<T> where T : class
    {
        IEnumerable<T> GetAll(string? includeProperties = null);
        T Get(Expression<Func<T, bool>> filter, string? includeProperties = null);
        void Add(T entity);
        void Remove(T entity);
        void RemoveRange(IEnumerable<T> entities);
    }
}


using HCM.DAL.Models;


namespace HCM.DAL.Repository.IRepository
{
    public interface IEmployeeRepository : IRepository<Employee>
    {
        void Update(Employee employee);
    }
}
using HCM.DAL.Data;
using HCM.DAL.Repository.IRepository;
using Microsoft.EntityFrameworkCore;
using System.Linq.Expressions;

namespace HCM.DAL.Repository
{
    public class Repository<T> : IRepository<T> where T : class
    {
        private readonly ApplicationDbContext _db;
        internal DbSet<T> dbSet;

        public Repository(ApplicationDbContext db)
        {
            _db = db;
            dbSet = _db.Set<T>();
        }

        public void Add(T entity)
        {
            dbSet.Add(entity);
        }
```

```
        public T Get(Expression<Func<T, bool>> filter, string? includeProperties =
null)
        {
            IQueryable<T> query = dbSet.Where(filter);
            if (!string.IsNullOrEmpty(includeProperties))
            {
                query = IncludeNavigationProperties(query, includeProperties);
            }
            return query.FirstOrDefault();
        }

        public IEnumerable<T> GetAll(string? includeProperties = null)
        {
            IQueryable<T> query = dbSet;
            if (!string.IsNullOrEmpty(includeProperties))
            {
                query = IncludeNavigationProperties(query, includeProperties);
            }
            return query.ToList();
        }

        public void Remove(T entity)
        {
            dbSet.Remove(entity);
        }

        public void RemoveRange(IEnumerable<T> entities)
        {
            dbSet.RemoveRange(entities);
        }

        private IQueryable<T> IncludeNavigationProperties (IQueryable<T> query,
string includeProperties)
        {
            string[] properties = includeProperties.Split(',',
StringSplitOptions.RemoveEmptyEntries);
            for (int i = 0; i < properties.Length; i++)
            {
                query = query.Include(properties[i]);
            }
            return query;
        }

    }
}


using HCM.DAL.Data;
using HCM.DAL.Models;
using HCM.DAL.Repository.IRepository;

namespace HCM.DAL.Repository
{
    public class EmployeeRepository : Repository<Employee>, IEmployeeRepository
    {
        private readonly ApplicationDbContext _db;

        public EmployeeRepository(ApplicationDbContext db) : base(db)
        {
            _db = db;
        }

        public void Update(Employee employee)
```

```
        {
            _db.Employees.Update(employee);
        }
    }
}



namespace HCM.DAL.Repository.IRepository
{
    public interface IUnitOfWork
    {
        IEmployeeRepository Employee { get; }
        IWorkStatusRepository WorkStatus { get; }
        IHealthStatusRepository HealthStatus { get; }
        IDepartmentRepository Department { get; }
        IAttendanceRepository Attendance { get; }
        IWorkAccidentRepository WorkAccident { get; }
        IEventLogRepository EventLog { get; }

        void Save();
    }
}


using HCM.DAL.Data;
using HCM.DAL.Repository.IRepository;

namespace HCM.DAL.Repository
{
    public class UnitOfWork : IUnitOfWork
    {
        public IEmployeeRepository Employee { get; private set; }
        public IWorkStatusRepository WorkStatus { get; private set; }
        public IHealthStatusRepository HealthStatus { get; private set; }
        public IDepartmentRepository Department { get; private set; }
        public IAttendanceRepository Attendance { get; private set; }
        public IWorkAccidentRepository WorkAccident { get; private set; }
        public IEventLogRepository EventLog { get; private set; }

        private readonly ApplicationDbContext _db;

        public UnitOfWork(ApplicationDbContext db)
        {
            _db = db;
            Employee = new EmployeeRepository(_db);
            WorkStatus = new WorkStatusRepository(_db);
            HealthStatus = new HealthStatusRepository(_db);
            Department = new DepartmentRepository(_db);
            Attendance = new AttendanceRepository(_db);
            WorkAccident = new WorkAccidentRepository(_db);
            EventLog = new EventLogRepository(_db);
        }

        public void Save()
        {
            _db.SaveChanges();
        }
    }
}


using HCM.DAL.Data;
```

```
using HCM.DAL.Repository;
using HCM.DAL.Repository.IRepository;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"))
);

builder.Services.AddDefaultIdentity<IdentityUser>().AddEntityFrameworkStores<Applica
tionDbContext>();
builder.Services.AddRazorPages();
builder.Services.AddScoped<IUnitOfWork, UnitOfWork>();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for production
scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

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

app.Run();


<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - HCM Web</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min-flatly.css"
/>
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
    <link rel="stylesheet" href="~/HCM_Web.styles.css" asp-append-version="true" />
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.11.3/font/bootstrap-icons.min.css">
    <link rel="stylesheet"
href="https://cdn.datatables.net/2.0.6/css/dataTables.dataTables.min.css" />
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/css/toastr.min.css" />
</head>
<body>
    <header>
```

```
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm bg-primary border-
bottom box-shadow mb-3" data-bs-theme="dark">
            <div class="container-fluid">
                <a class="navbar-brand" asp-area="" asp-controller="Home" asp-
action="Index">HCM Web</a>
                <button class="navbar-toggler" type="button" data-bs-
toggle="collapse" data-bs-target=".navbar-collapse" aria-
controls="navbarSupportedContent"
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex justify-
content-between">
                    <ul class="navbar-nav me-auto">
                        <li class="nav-item">
                            <a class="nav-link" asp-area="" asp-
controller="Employee" asp-action="Index">Employee</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link" asp-area="" asp-
controller="Attendance" asp-action="Index">Attendance</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link" asp-area="" asp-
controller="WorkAccident" asp-action="Index">Work Accidents</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link" asp-area="" asp-
controller="Department" asp-action="Index">Departments</a>
                        </li>
                        <li class="nav-item dropdown">
                            <a class="nav-link dropdown-toggle" href="#"
role="button" data-bs-toggle="dropdown" aria-expanded="false">
                                Info Management
                            </a>
                            <ul class="dropdown-menu">
                                <li>
                                    <a class="dropdown-item" asp-area="" asp-
controller="" asp-action="Index">Work Status</a>
                                </li>
                                <li>
                                    <a class="dropdown-item" asp-area="" asp-
controller="" asp-action="Index">Health Status</a>
                                </li>
                            </ul>
                        </li>
                    </ul>
                    <partial name="_LoginPartial" />
                </div>
            </div>
        </nav>
    </header>
    <div class="container">
        <main role="main" class="pb-3">
            <partial name="_Notification" />
            @RenderBody()
        </main>
    </div>

    <footer class="border-top footer text-muted">
        <div class="container">
            &copy; 2024 - HCM for Manufacturing Enterprise - <a asp-area="" asp-
controller="Home" asp-action="Privacy">Privacy</a>
```

```
        </div>
    </footer>
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
    <script src="~/js/site.js" asp-append-version="true"></script>
    <script src="https://cdn.datatables.net/2.0.6/js/dataTables.min.js" asp-append-
version="true"></script>
    <script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>
    @await RenderSectionAsync("Scripts", required: false)
</body>
</html>


<div class="container">
    <div class="border p-3 mt-4 border-2 border-primary">
        <div class="row pb-2">
            <h3 class="text-primary">Employee List</h3>
            <hr style="border-color: #1a2530" />
            <div class="">
                <a asp-controller="Employee" asp-action="Upsert" class="btn btn-
primary">
                    <i class="bi bi-plus-circle"></i> Add New Employee
                </a>
            </div>
        </div>

        <table id="tblData" class="table table-bordered table-striped"
style="width:100%">
            <thead>
                <tr>
                    <th>Name</th>
                    <th>Surname</th>
                    <th>Position</th>
                    <th>Department</th>
                    <th>Work Status</th>
                    <th>Health Status</th>
                    <th></th>
                </tr>
            </thead>

        </table>

    </div>
</div>

@section Scripts {
    <script src="~/js/employee.js"></script>
}

@model EmployeeVM

<div class="container">
    <div class="border p-3 mt-4 border-2 border-primary">
        <div class="row pb-2">
            <div class="col-12 text-center">
                <h3 class="text-primary"> @(Model.Employee.Id != 0 ? "Update" :
"Add") Employee</h3>
                <hr style="border-color: #1a2530" />
            </div>
        </div>

        <form method="post" class="row">
            <input asp-for="Employee.Id" hidden />
```

```
            <input asp-for="Employee.HiredById" hidden />
            <input asp-for="Employee.HiringDate" hidden/>
            <div class="row">
                <div class="col-12">
                    <div class="form-floating col-12">
                        <input asp-for="Employee.Name" type="text" class="form-
control" placeholder="Type here" />
                        <label asp-for="Employee.Name"></label>
                        <span asp-validation-for="Employee.Name" class="text-
danger"></span>
                    </div>
                    <div class="form-floating mt-3 col-12">
                        <input asp-for="Employee.Surname" type="text" class="form-
control" placeholder="Type here" />
                        <label asp-for="Employee.Surname"></label>
                        <span asp-validation-for="Employee.Surname" class="text-
danger"></span>
                    </div>
                    <div class="form-floating mt-3 col-12">
                        <input asp-for="Employee.Patronym" type="text" class="form-
control" placeholder="Type here" />
                        <label asp-for="Employee.Patronym"></label>
                        <span asp-validation-for="Employee.Patronym" class="text-
danger"></span>
                    </div>
                    <div class="form-floating mt-3 col-12">
                        <input asp-for="Employee.BirthDate" type="date" class="form-
control" placeholder="Type here" />
                        <label asp-for="Employee.BirthDate"></label>
                        <span asp-validation-for="Employee.BirthDate" class="text-
danger"></span>
                    </div>
                    <div class="form-floating mt-3 col-12">
                        <input asp-for="Employee.PassportNumber" type="text"
class="form-control" placeholder="Type here" />
                        <label asp-for="Employee.PassportNumber"></label>
                        <span asp-validation-for="Employee.PassportNumber"
class="text-danger"></span>
                    </div>
                    <div class="form-floating mt-3 col-12">
                        <input asp-for="Employee.TIN" type="text" class="form-
control" placeholder="Type here" />
                        <label asp-for="Employee.TIN"></label>
                        <span asp-validation-for="Employee.TIN" class="text-
danger"></span>
                    </div>
                    <div class="form-floating mt-3 col-12">
                        <input asp-for="Employee.Position" type="text" class="form-
control" placeholder="Type here" />
                        <label asp-for="Employee.Position"></label>
                        <span asp-validation-for="Employee.Position" class="text-
danger"></span>
                    </div>
                    <div class="form-floating mt-3 col-12">
                        <select asp-for="Employee.DepartmentId" asp-
items="@Model.DepartmentList" class="form-select">
                            <option disabled selected>--Select Department--</option>
                        </select>
                        <label asp-for="Employee.DepartmentId"></label>
                        <span asp-validation-for="Employee.DepartmentId"
class="text-danger"></span>
                    </div>
                    <div class="form-floating mt-3 col-12">
```

```
                        <select asp-for="Employee.WorkStatusId" asp-
items="@Model.WorkStatusList" class="form-select">
                            <option disabled selected>--Select Work Status--
</option>
                        </select>
                        <label asp-for="Employee.WorkStatus"></label>
                        <span asp-validation-for="Employee.WorkStatus" class="text-
danger"></span>
                    </div>
                    <div class="form-floating mt-3 col-12">
                        <select asp-for="Employee.HealthStatusId" asp-
items="@Model.HealthStatusList" class="form-select">
                            <option disabled selected>--Select Health Status--
</option>
                        </select>
                        <label asp-for="Employee.HealthStatusId"></label>
                        <span asp-validation-for="Employee.HealthStatusId"
class="text-danger"></span>
                    </div>
                    <div class="form-floating mt-3 col-12">
                        <input asp-for="Employee.AccessLevel" type="text"
class="form-control" placeholder="Type here" />
                        <label asp-for="Employee.AccessLevel"></label>
                        <span asp-validation-for="Employee.AccessLevel" class="text-
danger"></span>
                    </div>

                    <div class="row pt-3">
                        <div class="col-6 col-md-3">
                            @if (Model.Employee.Id != 0)
                            {
                                <button type="submit" class="btn btn-primary form-
control">Update</button>
                            }
                            else
                            {
                                <button type="submit" class="btn btn-primary form-
control">Create</button>
                            }
                        </div>
                        <div class="col-6 col-md-3">
                            <a asp-controller="Employee" asp-action="Index"
class="btn btn-outline-primary form-control">
                                Back to List
                            </a>
                        </div>
                    </div>

                </div>
            </div>
        </form>

    </div>
</div>

@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}

@if (TempData["success"] != null)
{
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
```

```
    <script
src="//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.min.js"></script>
    <script type="text/javascript">
        toastr.success('@TempData["success"]');
    </script>
}
@if (TempData["error"] != null)
{
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script
src="//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.min.js"></script>
    <script type="text/javascript">
        toastr.error('@TempData["error"]');
    </script>
}


var dataTable;

$(document).ready(function () {
      loadDataTable();
});

function loadDataTable() {
      dataTable = $('#tblData').DataTable({
            "ajax": { url: '/employee/getall' },
            "columns": [
                  { data: 'name', "width": "15%" },
                  { data: 'surname', "width": "15%" },
                  { data: 'position', "width": "15%" },
                  { data: 'department.name', "width": "20%" },
                  { data: 'workStatus.status', "width": "10%" },
                  { data: 'healthStatus.status', "width": "10%" },
                  {
                        data: 'id',
                        "render": function (data) {
                              return `<div class="btn btn-group" role="group">
                                    <div class="mx-2" style="text-align:center">
                                    <a href="/employee/upsert?id=${data}"
class="btn btn-warning" >
                                          <i class="bi bi-pencil-square"></i>
                                    </a>
                                    </div>
                                    <div class="mx-2" style="text-align:center">
                                    <a onClick=Delete('/employee/delete/${data}')
class="btn btn-danger">
                                          <i class="bi bi-trash"></i>
                                    </a>
                                    </div>
                               </div>`
                        },
                        "width": "15%"
                  }
            ]
      });
}

function Delete(url) {
      Swal.fire({
            title: "Are you sure?",
            text: "You won't be able to revert this!",
            icon: "warning",
            showCancelButton: true,
```

```
            confirmButtonColor: "#3085d6",
            cancelButtonColor: "#d33",
            confirmButtonText: "Yes, delete it!"
      }).then((result) => {
            if (result.isConfirmed) {
                  $.ajax({
                        url: url,
                        type: 'DELETE',
                        success: function (data) {
                              dataTable.ajax.reload();
                              toastr.success(data.message);
                        }
                  })
            }
      });
}


{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection":
"Server=localhost;Database=HCM_System;Trusted_Connection=True;TrustServerCertificate
=True",
    "ApplicationDbContextConnection":
"Server=(localdb)\\mssqllocaldb;Database=HCM_Web;Trusted_Connection=True;MultipleAct
iveResultSets=true"
  }
}
```