НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук

/Голуб Б.Л., доц., к.т.н. /
підпис ПІБ, вчене звання і ступінь

«___»_____2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмне забезпечення з обліку графіків робочих змін персоналу»

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент / Вайганг Г.О./
Науковий ступень та вчене звання підпис ПІБ

Керівник бакалаврської кваліфікаційної роботи : / Бородкін Г.О./
підпис ПІБ

Виконав: / Тимошенко О.Є./
підпис ПІБ

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерних наук

_____ / Голуб Б.Л., доцент, к.т.н /
підпис

"___" _____ 2025 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студенту Тимошенку Олександру Євгеновичу

Спеціальність 121 «Інженерія програмного забезпечення»

1. Тема роботи: Програмне забезпечення з обліку графіків робочих змін персоналу

Затверджена наказом ректора НУБіП України № 2248 "С" від 16.12.2024

2. Термін подання завершеної роботи на кафедру 2025 . 05 . 21
рік, місяць, число

3. Вихідні дані до роботи: опис програмного забезпечення

4. Перелік питань що розглядаються:

1. Аналіз проблемної області.
2. Вибір та обґрунтування засобів для розробки системи.
3. Проектування інформаційної системи.
4. Висновки.

Керівник бакалаврської кваліфікаційної роботи _____ / Бородкін Г.О. /
підпис ініціали та прізвище

Завдання прийняла до виконання _____ / Тимошекно О.Є. /
підпис ініціали та прізвище

Дата отримання завдання 2025 . 03 . 17
рік, місяць, число

ЗМІСТ

ВСТУП

У сучасних умовах цифрової трансформації підприємств та впровадження автоматизованих систем управління особливої актуальності набуває проблема ефективного планування та обліку робочого часу персоналу, зокрема – позмінної роботи.

У більшості організацій, діяльність яких передбачає змінний режим роботи, облік змін і графіків залишається ручним або виконується за допомогою загального офісного програмного забезпечення, яке не пристосоване для гнучкого та надійного ведення змін. Це призводить до неузгодженості розкладів, конфліктів між працівниками, збоїв у виробничих і сервісних процесах, перевантаження персоналу, а також ускладнює контроль з боку керівництва.

Розробка програмного забезпечення для автоматизованого обліку графіків робочих змін покликана вирішити низку важливих завдань:

– забезпечити централізоване зберігання даних, прозорість змін, доступ різних категорій користувачів (адміністраторів, менеджерів змін, працівників) до відповідної інформації;

– автоматизувати процеси створення, редагування, підтвердження та перегляду змін, а також формувати звітність у зручному форматі;

– адаптуватися до змін у внутрішніх регламентах підприємства;

– мати розмежування прав доступу;

– забезпечувати захист персональних даних.

Метою дослідження є проєктування, розробка та апробація програмного продукту, призначеного для автоматизованого обліку графіків робочих змін персоналу. Система повинна охоплювати повний цикл життєдіяльності графіка змін: від створення і налаштування шаблонів змін до їх редагування, затвердження та архівації, а також передбачати зручний інтерфейс для всіх категорій користувачів. Реалізація поставленої мети вимагає аналізу предметної області, моделювання основних процесів, проєктування структури бази даних, розробки

архітектури програмної системи, вибору ефективних засобів розробки та побудови інтерфейсу.

Під час виконання даної роботи будуть використовуватися сучасні методи програмної інженерії, включаючи об’єктно-орієнтований підхід до проєктування, засоби UML-моделювання (діаграми варіантів використання, класів, активності, компонентів, розгортання), реляційне моделювання даних (ER-діаграми), методи нормалізації та створення реляційних структур. Вибір мови програмування C# з використанням середовища розробки Visual Studio зумовлено високим рівнем підтримки побудови клієнт-серверних архітектур, наявністю бібліотек для роботи з базами даних та створенням сучасного графічного інтерфейсу. Для управління базою даних обрано MySQL як надійну, продуктивну й безкоштовну СУБД.

Структура дипломної роботи включає вступ, у якому викладено мету, актуальність, обґрунтовано вибір технологій та описано загальну архітектуру системи. Основна частина містить системний аналіз предметної області, проєктування інформаційної та програмної складових, реалізацію окремих програмних модулів та рекомендації щодо впровадження. У завершальній частині наведено висновки, які підсумовують виконану роботу та визначають перспективи подальшого розвитку системи.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Система обліку графіків робочих змін персоналу повинна допомогти підприємствам, у яких працівники працюють позмінно. Це можуть бути лікарні, виробничі підприємства, охоронні агентства, чергові служби, магазини з цілодобовим режимом роботи тощо. У таких закладах важливо контролювати змінність, забезпечувати рівномірне навантаження між співробітниками, враховувати відпустки, лікарняні та інші фактори, що впливають на графік роботи.

Основні цілі програми:

– спростити роботу адміністраторам і менеджерам змін;

– дозволити легко створювати, змінювати та підтверджувати графіки змін;

– забезпечити доступ працівників до актуального графіка;

– автоматизувати ведення історії змін та формування звітів.

Робота з графіками змін – це постійна рутинна діяльність, що пов’язана з великою кількістю одноманітних операцій. У ручному режимі це вимагає багато часу, супроводжується помилками, та ускладнює контроль. До того ж, часто виникають ситуації, коли зміни потрібно терміново змінити або повідомити працівників про оновлення – без програмного забезпечення це робиться вручну або через телефонні дзвінки.

Хоча програмне забезпечення не автоматизує усі кадрові процеси повністю, воно значно полегшує керування робочими змінами. Система матиме зручний інтерфейс, простий у використанні навіть для користувачів без глибоких технічних знань, та ефективний інструментарій для менеджера змін, адміністратора та працівника.

Серед функцій, які виконуватиме розроблена інформаційна система, слід відзначити такі:

– створення нового графіка змін;

– редагування та підтвердження існуючих графіків;

– збереження історії змін для аудиту;

– розмежування доступу відповідно до ролі користувача;

– формування та експорт звітів;

– автоматичне сповіщення працівників про зміни у графіку.

Система має бути простою в експлуатації, зрозумілою та адаптованою під потреби організації. Уся інформація про графіки, працівників, зміни та історію зберігається у централізованій базі даних, що дозволяє отримувати актуальні відомості у реальному часі та гарантує збереження даних при віддаленому доступі або локальній роботі.

Система повинна містити відомості про працівників (ПІБ, посада, відділ), робочі зміни (дата, час початку/завершення, відповідальний), відомості про підтвердження змін і сповіщення та журнал змін графіка для перегляду історії.

Програма повинна бути інтуїтивно зрозумілою, підтримувати централізований або розподілений режим роботи, працювати швидко та стабільно, а також мати гнучкість для розширення або інтеграції з іншими внутрішніми системами підприємства.

У сучасних умовах ефективне управління персоналом є однією з ключових умов стабільної роботи підприємства. Особливої важливості набуває це завдання в організаціях із позмінним графіком, де кожне відхилення чи непередбачувана ситуація можуть призвести до зривів у роботі, простоїв або перевантаження окремих працівників. Впровадження автоматизованої системи для обліку змін дозволяє суттєво знизити ризики, пов’язані з людським фактором, а також забезпечити прозорість і оперативність при формуванні графіків.

Предметна область передбачає фіксацію присутності або запланованих змін та гнучке реагування на позаштатні ситуації, оптимізацію розподілу ресурсів, підтримку справедливого навантаження, а також можливість стратегічного планування. Зміни повинні бути адаптивними до реальних умов: вихідних днів, державних свят, форс-мажорів, сезонності, лікарняних і відпусток. Тому система

повинна враховувати великий набір вхідних параметрів і дозволяти оперативне оновлення без шкоди для загальної структури графіка.

Особливістю предметної області є динамічність: склад команди може змінюватися, функціональні ролі перерозподілятися, а робочі зміни мати плаваючий характер залежно від навантаження або типу робіт. Відповідно, інформаційна система повинна мати можливість адаптації до таких змін без потреби суттєвої переналаштування або втручання розробників. Автоматизоване керування змінами також забезпечує підвищення мотивації персоналу, адже кожен працівник має змогу бачити своє навантаження, заздалегідь планувати вільний час і своєчасно отримувати інформацію про будь-які зміни.

У рамках даної предметної області велике значення має логіка доступу до даних. Залежно від ролі, різні користувачі матимуть різний рівень доступу: адміністратори можуть створювати й редагувати графіки, керівники відділів – затверджувати або вносити корективи, а звичайні працівники – переглядати свої зміни й отримувати сповіщення.

1.2 Аналіз вимог до програмної системи

Аналіз вимог – важливий етап проєктування, оскільки саме на цьому етапі формується повне уявлення про функціональність, яку повинна забезпечувати система, її поведінку, обмеження та очікування користувачів [1]. Для системи обліку графіків робочих змін персоналу вимоги визначаються особливостями предметної області, специфікою управління змінами, а також ролями користувачів, які взаємодіють із системою.

Система повинна забезпечувати автоматизоване створення та облік робочих змін, дозволяти адміністратору керувати користувачами, а менеджеру змін – формувати, редагувати та затверджувати графіки. Пересічні користувачі повинні мати змогу переглядати свої графіки, отримувати сповіщення про зміни та бути впевненими у їхній актуальності. Таким чином, система повинна підтримувати

багаторівневий доступ з розмежуванням прав на основі ролей, що забезпечує інформаційну безпеку та захист персональних даних.

Функціональні вимоги до системи охоплюють низку основних можливостей. До них належать: авторизація користувачів, створення нового графіка змін, редагування існуючих змін, підтвердження змін менеджером, перегляд змін користувачем, ведення історії змін, створення звітів, сповіщення працівників про оновлення в розкладі. Система повинна підтримувати пошук, сортування та фільтрацію інформації за різними параметрами, такими як прізвище працівника, дата зміни, статус зміни тощо.

Нефункціональні вимоги передбачають зручність користування, інтуїтивно зрозумілий інтерфейс, мінімальний час на навчання користувачів, підтримку української мови інтерфейсу, а також кросплатформеність у разі потреби. Програма має забезпечити високу продуктивність навіть за великої кількості записів, масштабованість у разі розширення обсягу даних або кількості користувачів, а також можливість легкої інтеграції з іншими внутрішніми системами підприємства.

Технічні вимоги до реалізації системи передбачають використання клієнт-серверної архітектури. На стороні клієнта повинна функціонувати графічна оболонка, яка надає користувачеві доступ до всіх передбачених функцій, а на стороні сервера – модуль логіки та взаємодії з базою даних. База даних має бути реалізована на основі реляційної моделі з підтримкою нормалізації, щоб забезпечити структуроване зберігання інформації про користувачів, графіки, зміни, підтвердження, звіти та історію.

У процесі деталізації вимог до програмної системи особливу увагу слід приділити безпосередньо функціоналу та сценаріям використання системи в реальному середовищі. Ситуації, пов’язані з екстреними замінами персоналу, необхідністю термінового коригування змін у випадку непередбачуваних обставин, а також потребою в попередньому плануванні на місяці вперед – усе це повинно знайти відображення у функціональних сценаріях системи.

Критичною є вимога до забезпечення надійної ідентифікації користувачів і документування їхньої активності. Усі ключові дії – створення, редагування, підтвердження чи скасування змін – мають бути зафіксовані в системі із зазначенням дати, часу та ідентифікатора відповідального користувача.

Окремим викликом є забезпечення стійкої роботи системи при паралельному доступі великої кількості користувачів. Адже у великих установах чи корпораціях одночасна робота з графіками може відбуватись у десятках відділів, і система має гарантувати коректну синхронізацію змін без втрати даних.

Умови сучасної роботи вимагають, аби менеджери змін могли оперативно вносити корективи навіть із мобільних пристроїв або планшетів, тому система повинна мати адаптивну верстку, що зберігає повну функціональність незалежно від розміру екрана. Підтримка push-сповіщень або email-інформування також є доцільною функціональністю для забезпечення миттєвого зворотного зв’язку з персоналом.

Для підвищення масштабованості та продуктивності бажано передбачити можливість хмарного розгортання системи, що забезпечить безперебійний доступ до неї з будь-якої точки, надійне резервне копіювання та захист від локальних збоїв. При цьому конфіденційність персональних даних повинна гарантуватися шляхом шифрування критичних полів, використання захищених каналів передачі інформації (наприклад, HTTPS), а також регулярного оновлення політик безпеки відповідно до сучасних стандартів.

1.3 Огляд інформаційних джерел та існуючих рішень

Огляд інформаційних джерел та існуючих рішень є важливим етапом під час розробки будь-якого програмного забезпечення, оскільки він дозволяє вивчити сучасний стан розробок у вибраній предметній області, уникнути повторення помилок, виявити недоліки аналогів та сформувати конкурентні переваги власної системи. У процесі дослідження тематики обліку графіків робочих змін персоналу

було проаналізовано низку наукових публікацій, технічної документації, описів готових рішень, а також оглянуто реальні програмні продукти, що використовуються у різних організаціях для вирішення аналогічних задач.

Згідно з науковими дослідженнями, проблема оптимального планування змін належить до категорії задач, пов’язаних з управлінням ресурсами, зокрема людськими [2]. У наукових працях, присвячених інформаційним системам управління персоналом, часто наголошується на важливості автоматизації планування змін для підвищення ефективності роботи підприємств з позмінною організацією праці.

Серед програмних рішень, що існують на ринку, можна виокремити кілька найбільш поширених типів систем. По-перше, це великі ERP-системи, до складу яких входить модуль управління персоналом і планування графіків – такі як SAP, Oracle PeopleSoft, Microsoft Dynamics [3]. Ці продукти мають потужну функціональність, але вони дорогі, складні в налаштуванні, потребують окремої команди спеціалістів для впровадження і часто є надмірними для підприємств середнього або малого рівня. По-друге, існують хмарні SaaS-рішення, такі як When I Work, Deputy, Shiftboard, які надають користувачам інтерфейс для планування змін, комунікації з працівниками, перегляду графіків та інтеграції з мобільними пристроями. Основними перевагами цих систем є доступність з будь-якої точки світу, регулярні оновлення, гнучкість. Однак вони часто мають обмежений набір функцій у безкоштовних версіях, зберігають дані на сторонніх серверах, що може викликати занепокоєння з точки зору безпеки, а також не завжди дозволяють адаптацію під специфіку українського ринку чи окремого підприємства.

Також існує ціла низка вузькоспеціалізованих систем, що створюються під конкретні підприємства або галузі. Наприклад, системи обліку змін для лікарень дозволяють враховувати категорії медперсоналу, інтенсивність чергувань, облік нічних змін, ротацію між відділеннями. Системи для виробничих підприємств часто інтегруються з контролем доступу, табельним обліком, автоматизованими годинниками. Проте такі рішення зазвичай створюються як індивідуальні проєкти під конкретного замовника і недоступні для широкого використання.

Багато організацій користуються самописними системами або електронними таблицями, зокрема Microsoft Excel або Google Sheets, в яких вручну створюють графіки змін, додають формули, умовне форматування тощо [4]. Цей підхід є найдешевшим, однак він не забезпечує необхідного рівня автоматизації, не дозволяє контролювати історію змін, обмежує спільну роботу з даними та не гарантує захищеного доступу.

Під час вивчення рішень також було виявлено, що більшість сучасних систем не приділяють достатньої уваги локалізації, адаптації під законодавство конкретної країни, ігнорують питання інтеграції з електронними табелями обліку робочого часу, журналами змін або медичними довідками. Часто відсутня можливість налаштувати шаблони змін, змінювати тривалість зміни, призначати відповідальних за певні ділянки чи процеси. Ще однією поширеною проблемою є відсутність зручного, зрозумілого інтерфейсу – особливо для працівників, які не мають глибоких знань у сфері IT.

У численних відгуках користувачів готових платформ виявляється, що одним із ключових недоліків є негнучкість систем щодо змін у штатному розкладі або непередбачених подій. Зміна працівника на зміні, перенесення часу, погодження змін з керівництвом – усе це часто потребує багато ручної роботи або непропорційно складного процесу редагування. У деяких SaaS-продуктах спостерігається відсутність багаторівневої системи прав доступу або деталізованого логування дій, що унеможливлює контроль за змінами у графіках. Деякі системи взагалі не підтримують архівацію графіків або їх експорт у форматах, зручних для подальшої обробки, що знижує їхню практичну цінність у довгостроковому використанні.

Багато програмних рішень просто ігнорують нюанси, пов’язані з веденням обліку згідно з КЗпП України, наприклад – облік понаднормових годин, чергувань у святкові дні, нічної роботи тощо. Малопоширеним є врахування елементів соціального планування – наприклад, можливості встановлення пріоритетів або обмежень для окремих категорій працівників (осіб з інвалідністю, працівників із дітьми, студентів).

Ще одним слабким місцем багатьох рішень є відсутність адаптивного мобільного інтерфейсу. Все більше людей використовують смартфони для доступу до робочої інформації, це обмежує оперативність взаємодії з системою. Виявлено обмежену кількість систем, які дозволяють користувачу самостійно вносити побажання щодо змін, погоджувати чергування з іншими працівниками або використовувати систему як середовище для командної комунікації.

1.4 Постановка завдання

Головним завданням розробки є створення програмного продукту, що дозволяє підприємству або організації з позмінним режимом роботи ефективно формувати, редагувати, підтверджувати та вести облік графіків робочих змін персоналу. Система повинна забезпечити можливість централізованого управління змінами, надати користувачам інструменти для перегляду актуального графіка, сповіщати про зміни, зберігати історію внесених коригувань, а також генерувати звіти для управлінського аналізу.

Програма має включати інтерфейси для різних категорій користувачів – адміністратора, менеджера змін, працівника – з відповідним розмежуванням прав доступу. Адміністратор повинен мати змогу керувати обліковими записами, створювати резервні копії, переглядати загальну інформацію по всьому підприємству. Менеджер змін повинен отримати зручний інструмент для створення і зміни графіків, а працівник – мати доступ до персонального розкладу.

У рамках проєктування системи необхідно чітко визначити, які саме дані підлягають збереженню. До таких належать відомості про працівників (ПІБ, посада, відділ, ідентифікатор), дані про зміну (дата, час початку та завершення, тип зміни, відповідальний менеджер), відомості про підтвердження змін, сповіщення, історію редагування графіків, а також системні дані про користувачів і їхні ролі в системі. Система повинна мати можливість швидкого пошуку по базі даних, сортування записів, фільтрації за визначеними критеріями. Важливою є функція

резервного копіювання бази даних, можливість відновлення попередньої версії графіка, захист від несанкціонованого доступу до персональної інформації.

Програмна система повинна підтримувати функції авторизації користувача з перевіркою прав доступу до модулів і операцій, ведення журналу дій, що дасть змогу здійснювати аудит дій у системі. У випадку зміни розкладу система має надіслати повідомлення відповідному працівнику, що дозволить забезпечити актуальність отриманої інформації.

Оскільки багато підприємств не мають можливості використовувати дорогі комерційні рішення, розробка має бути орієнтована на простоту розгортання, мінімальні вимоги до апаратного забезпечення, інтуїтивно зрозумілий інтерфейс та універсальність. Система повинна працювати на настільному комп’ютері під управлінням Windows, не вимагати складної установки й мати можливість локального або мережевого використання. З урахуванням можливих обмежень у навичках користувачів, особлива увага має бути приділена юзабіліті інтерфейсу: зрозуміла структура меню, мінімізація ручного введення, використання підказок, кольорових позначень змін та фільтрації.

1.5 Моделювання предметної області

Перед створенням програмного забезпечення для обліку графіків робочих змін персоналу, моделювання здійснювалося на основі сучасних підходів об’єктно-орієнтованого аналізу з використанням набору UML-діаграм, що дозволили наочно представити логіку роботи системи, її функціональність, структуру, поведінку та взаємодію компонентів [5].

Діаграма варіантів використання відіграє роль загальної карти взаємодії користувачів із системою (див. рис. 1.1). Вона відображає ключові функціональні сценарії, що виконуються різними ролями: адміністратором, менеджером змін та працівником.

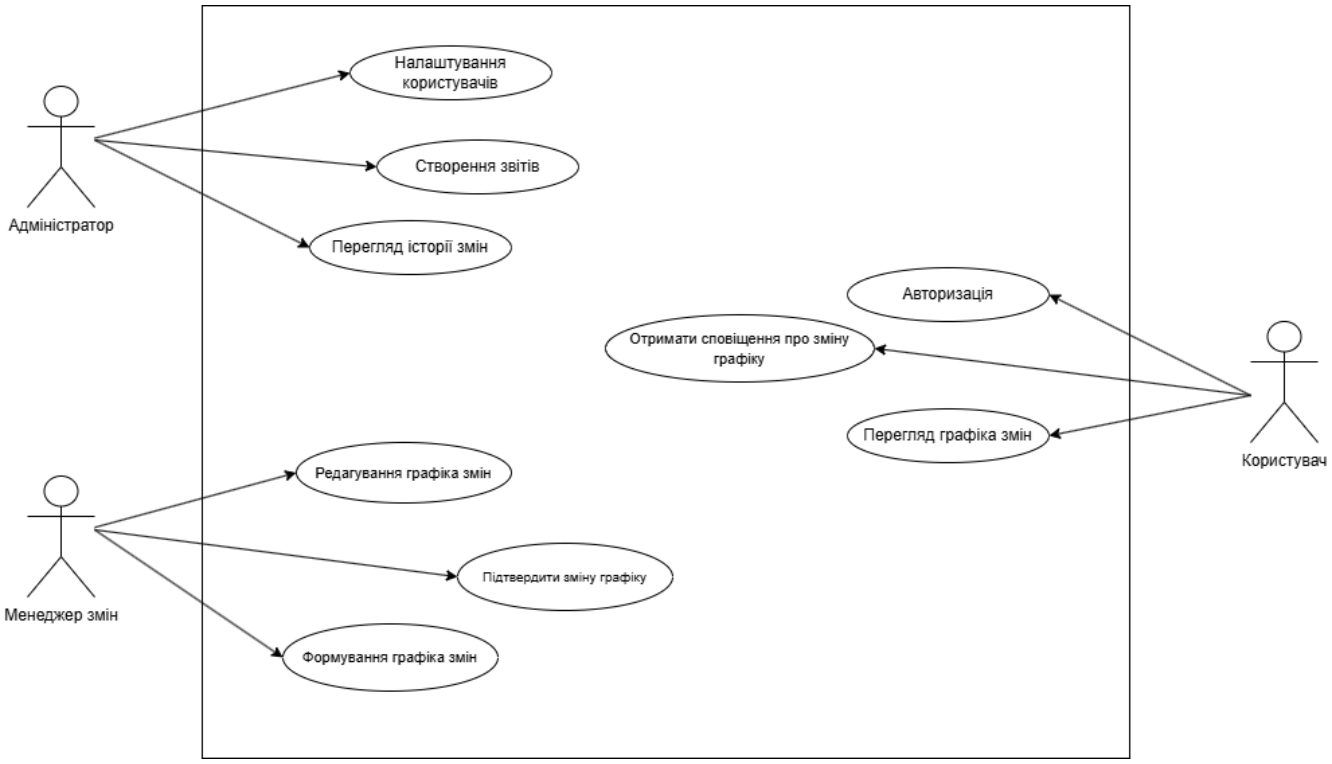

Рис. 1.1 Діаграма варіантів використання

Користувач має змогу авторизуватись, переглядати свій графік змін, отримувати повідомлення. Менеджер змін формує, редагує та затверджує графіки, створює звіти, переглядає історію змін. Адміністратор управляє обліковими записами та має доступ до всіх функціональних модулів. Використання включень і розширень у варіантах використання дозволяє відобразити зв’язки між сценаріями, які автоматично активуються в межах інших сценаріїв, наприклад, надсилання сповіщення працівнику при збереженні або підтвердженні зміни. Дана діаграма є основою для виявлення функцій, які реалізуються в системі, та визначення взаємодії між користувачами й програмним середовищем.

Діаграма послідовності описує динаміку обміну повідомленнями між об'єктами під час виконання конкретного варіанту використання (див. рис. 1.2) [6]. В межах проєкту змодельовано послідовність дій під час авторизації, перегляду графіка, редагування зміни або формування звіту. При перегляді графіка користувач ініціює запит, система перевіряє права доступу, надсилає запит до бази даних, отримує графік і передає його на відображення у користувацький інтерфейс.

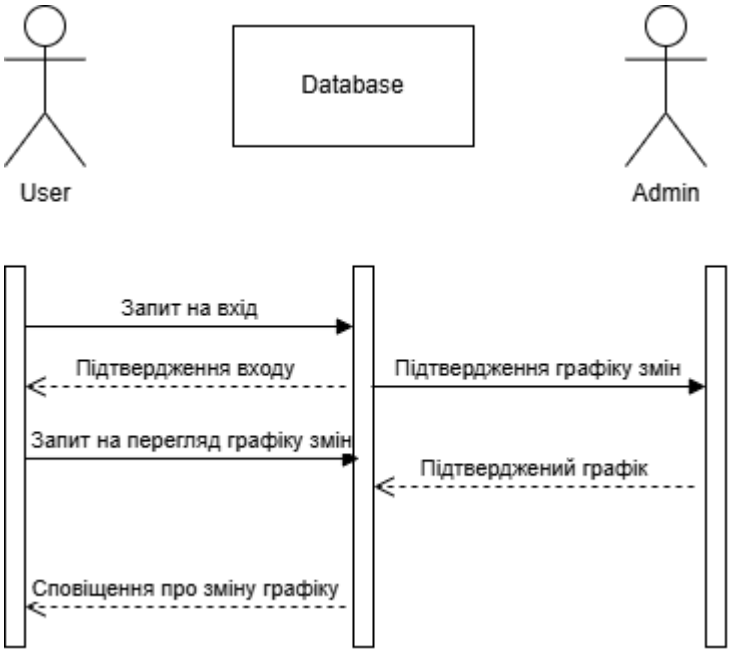

Рис. 1.2 Діаграма послідовності

Діаграма активності моделює логіку виконання процесу як потоку активностей. Вона демонструє алгоритм роботи користувача із системою: авторизація, вибір дії, перегляд або редагування графіка, формування звіту, вихід із системи (див. рис. 1.3).

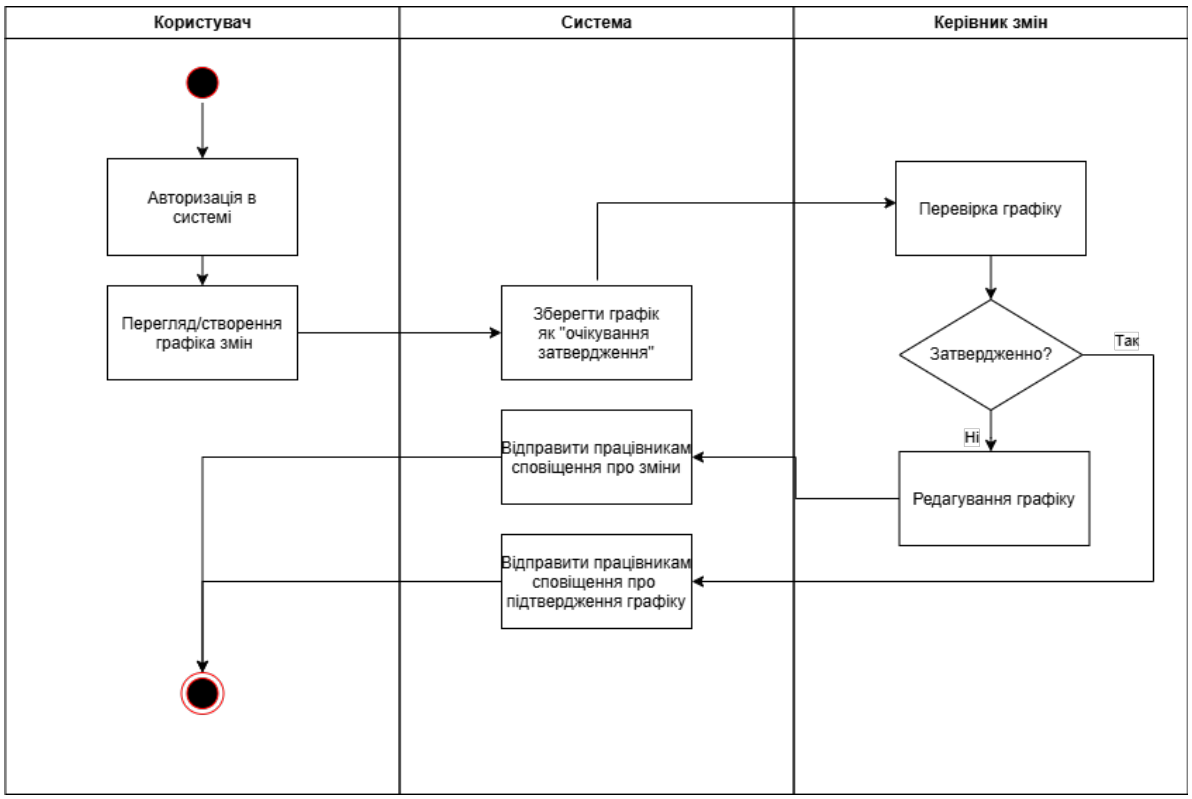

Рис. 1.3 Діаграма активності

Кожна гілка активності відображає окремий варіант поведінки: у разі успішної авторизації користувач переходить до вибору дій, після чого система реагує залежно від обраної опції. Візуалізована структура активностей дозволяє виявити можливі точки прийняття рішень, розгалуження і повернення до головного меню. Діаграма активності також дає змогу описати паралельність або послідовність виконання процесів, що є важливим для програмної реалізації.

Модель абстракцій предметної області подає ключові сутності, які є основою всієї системи, у вигляді узагальнень (див. рис. 1.4). Абстракції включають базові поняття, що характеризують працівника, зміну, графік, адміністратора, менеджера, історію змін. Наприклад, користувач представлений як загальна абстракція, яка ділиться на працівника, менеджера й адміністратора, кожен із яких має специфічні дії, але спільну структуру. Абстрагування дозволяє спростити проєктування класів і полегшити масштабування функціональності системи.

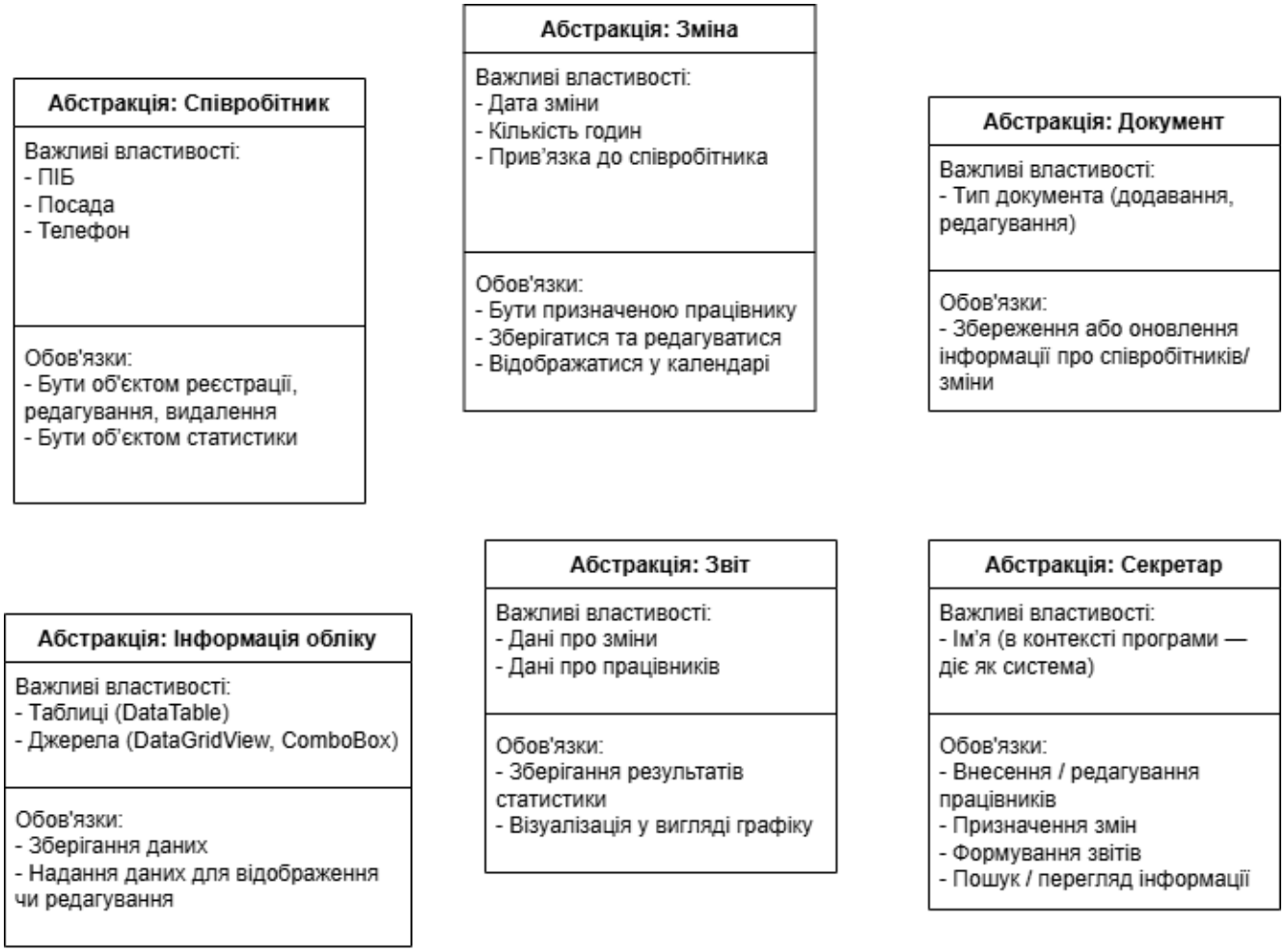

Рис. 1.4 Діаграма абстракції

Діаграма класів є центральною моделлю, що відображає структуру системи (див. рис. 1.5) [7]. У межах основної діаграми класів виділено сутності: Користувач, Графік, Зміна, Повідомлення, ІсторіяЗмін, Звіт, кожна з яких має свій набір атрибутів і методів. Між класами встановлені асоціації, наприклад, один графік містить багато змін, одна зміна пов’язана з одним працівником, зміна має історію редагувань. Клас «Користувач» пов'язаний з ролями, що визначають дозволені операції. Асоціації та зв’язки типу один-до-багатьох дозволяють змоделювати логіку взаємозалежності даних.

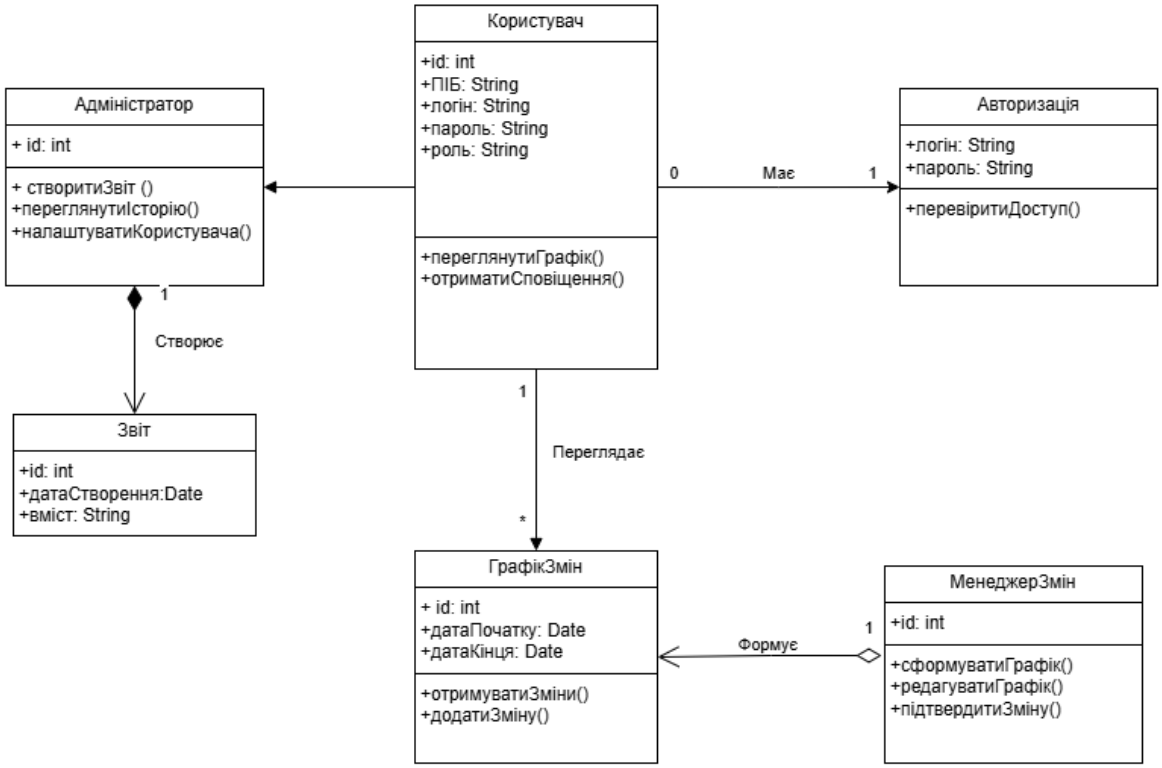

Рис. 1.5 Діаграма класів

Діаграма пакетів подає організаційну структуру програмного забезпечення на логічному рівні (див. рис. 1.6). Програмне забезпечення було розділено на модулі: інтерфейс користувача, управління графіками, керування користувачами, модуль історії змін, звітність, повідомлення та база даних. Така модульність забезпечує гнучкість, масштабованість та зручність супроводу. Відокремлення функціональних блоків у пакети дозволяє незалежно тестувати, розробляти та оновлювати окремі частини системи без ризику впливу на інші компоненти.

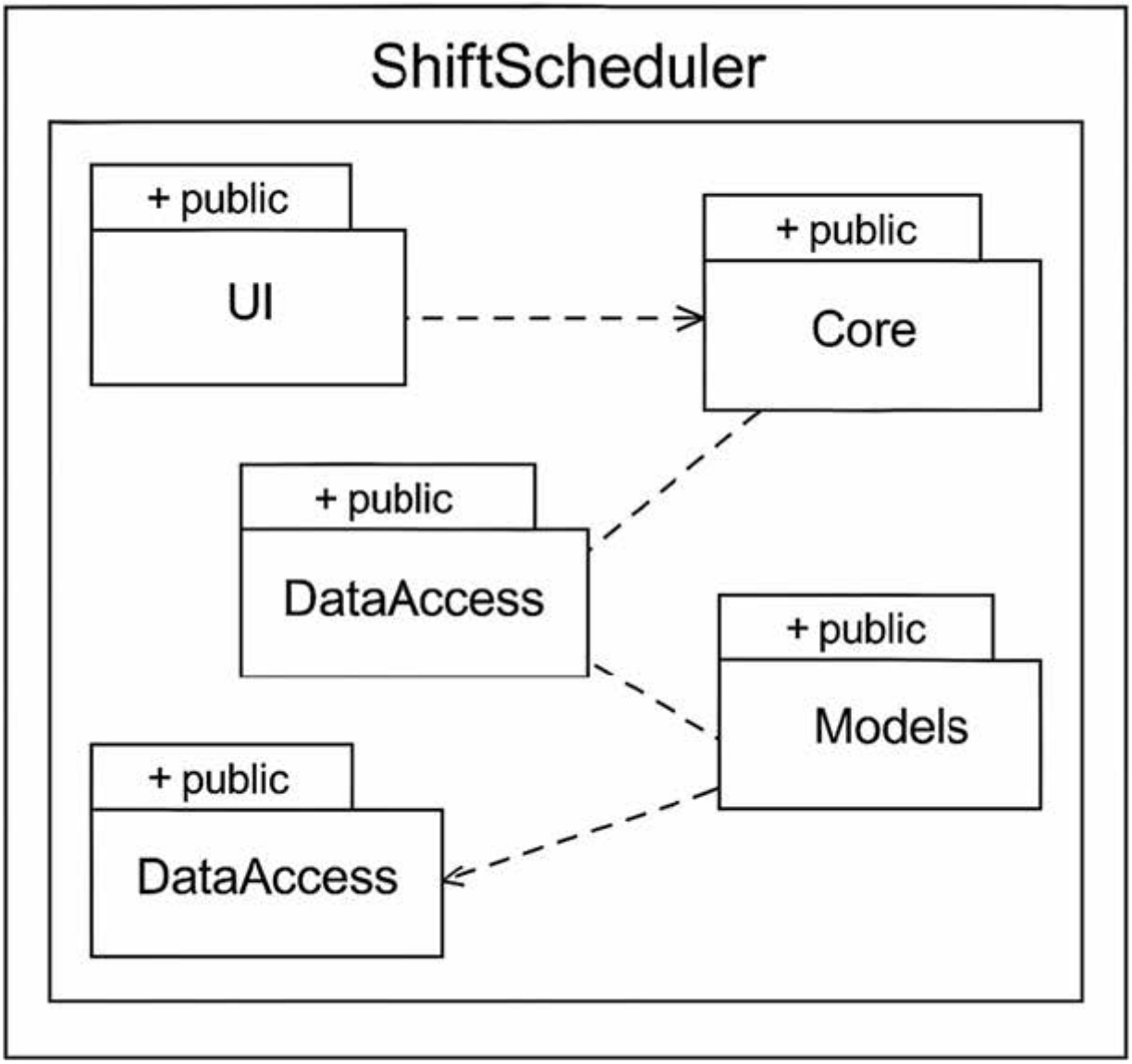

Рис. 1.6 Діаграма пакетів

Модель даних фізичного рівня реалізована у вигляді реляційної структури з нормалізованими таблицями, які зберігають дані про працівників, графіки, зміни, ролі користувачів, повідомлення, історію змін і звіти (див. рис. 1.7). Усі таблиці пов’язані зовнішніми ключами, що забезпечує цілісність даних і логіку відношень. Модель дотримується вимог третьої нормальної форми, що зменшує надмірність даних і мінімізує ймовірність аномалій при оновленні. Реалізація такої моделі у СУБД забезпечує високу продуктивність при виконанні запитів, надійне зберігання інформації та підтримку масштабованості.

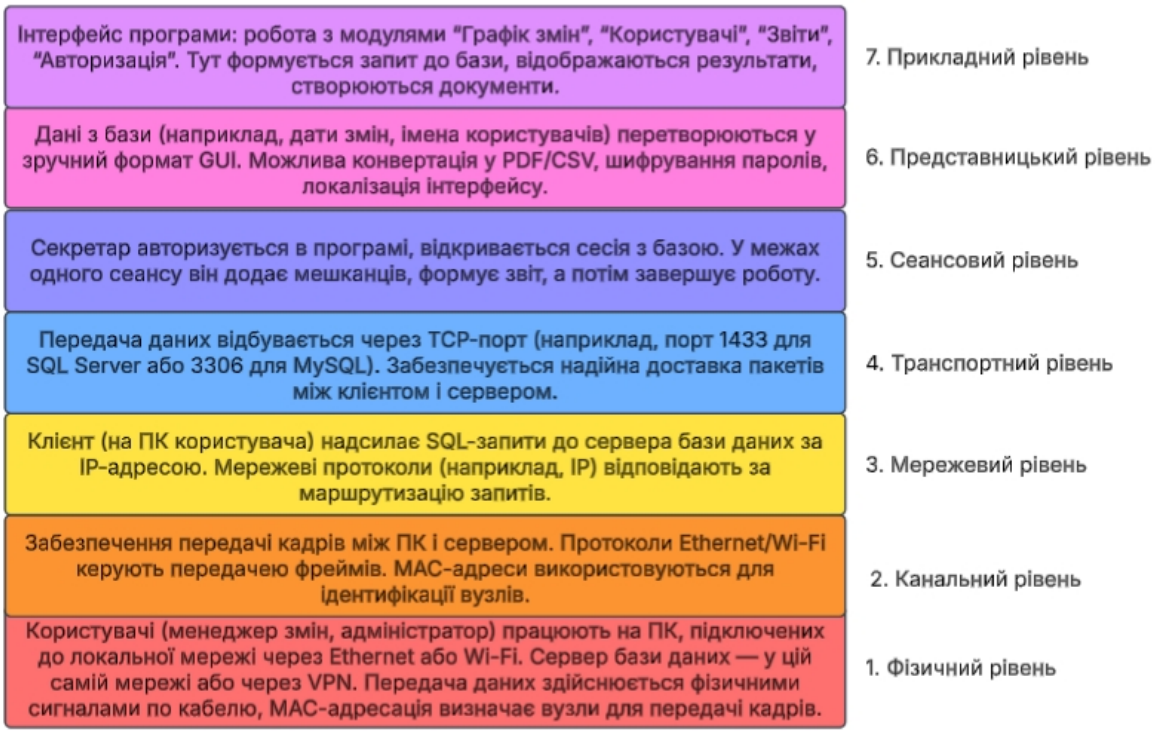

Рис. 1.7 Модель даних фізичного рівня

Діаграма компонентів моделює структуру вихідного коду системи, відображаючи взаємозв’язки між логічними модулями (див. рис. 1.8). Компоненти авторизації, управління графіками, обробки історії, сповіщення, формування звітів представлені як окремі частини, що взаємодіють із базою даних та між собою через визначені інтерфейси. Це дозволяє реалізувати принципи слабкого зв’язування та високої когезії між модулями. Також у діаграмі враховано процес компіляції та створення виконуваного файлу, який об’єднує всі компоненти в єдину програму.

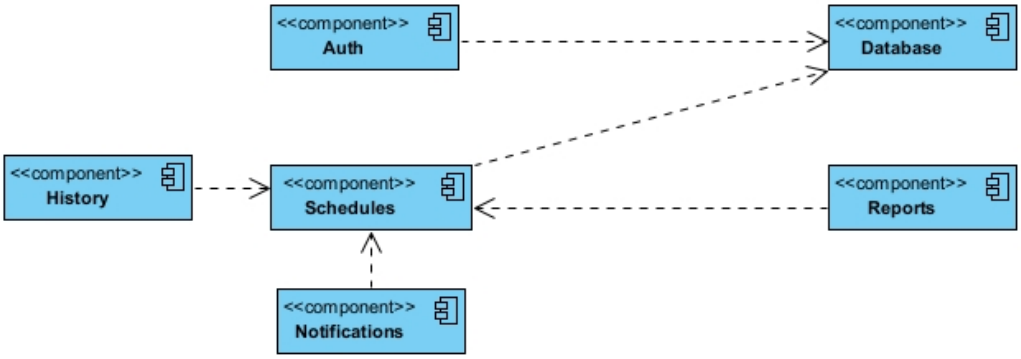

Рис. 1.8 Діаграма компонентів

Діаграма розгортання моделює фізичне середовище функціонування системи (див. рис. 1.9) [8]. Вона демонструє, що система складається з клієнтської частини, розгорнутої на комп'ютерах користувачів, та серверної частини, розміщеної на сервері, який містить базу даних та логіку обробки запитів. Клієнтська частина взаємодіє з сервером через протокол HTTP або TCP/IP, що забезпечує масштабованість і можливість використання системи у мережевому середовищі. Така структура дозволяє розподілити навантаження, централізовано адмініструвати систему та забезпечити безпеку даних.

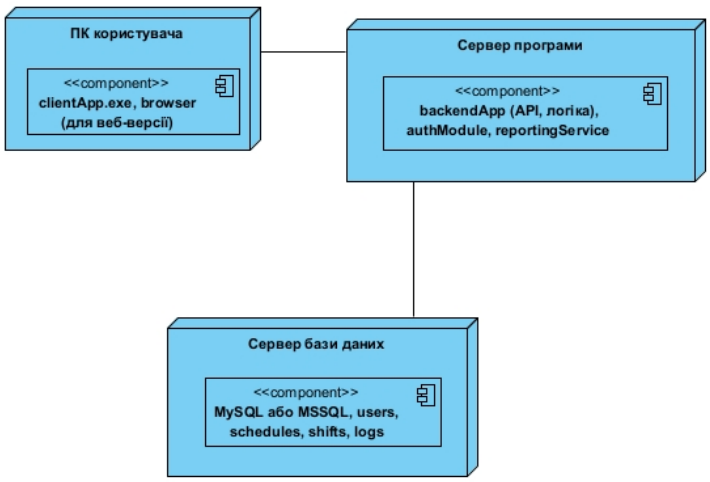

Рис. 1.9 Діаграма розгортання

Блок-схема алгоритму представляє логіку взаємодії користувача із системою в класичному вигляді: авторизація, вибір дії, обробка запиту, збереження даних, вивід результату (див. рис. 1.12). Вона візуально демонструє послідовність етапів роботи програми, включно з перевірками умов, що дозволяє однозначно

інтерпретувати алгоритмічну поведінку системи. З цією схемою узгоджується візуальна реалізація алгоритму, де кожна дія користувача відображена графічно, з використанням стандартних елементів – початок, дія, рішення, кінець.

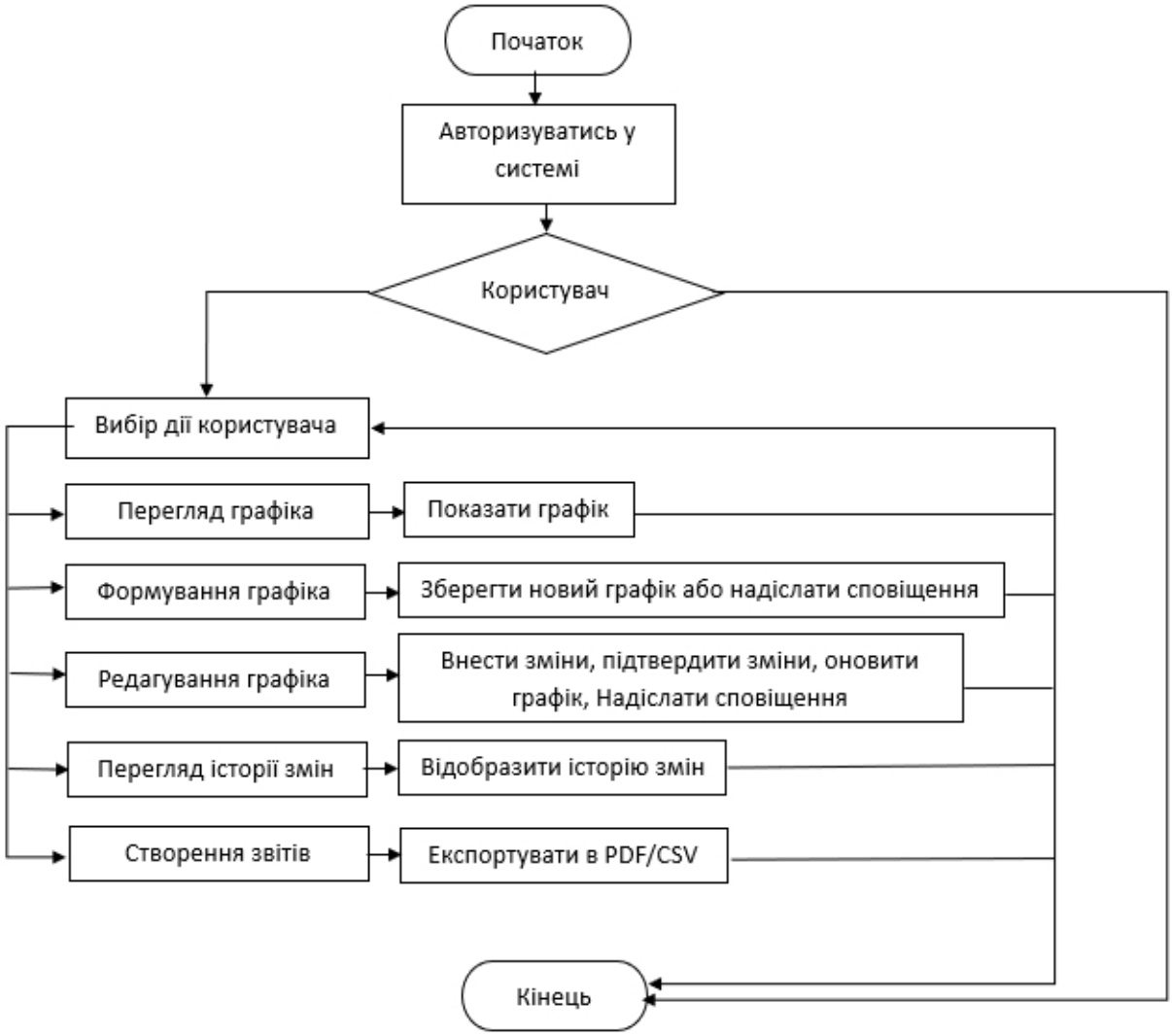

Рис. 1.10 Блок-схема алгоритму

Висновок

У результаті системного аналізу було чітко визначено актуальні проблеми обліку змін на підприємствах з позмінною роботою, що часто пов’язані з неефективністю ручного управління, неузгодженістю розкладів і браком гнучкості.

Було проведено глибоке дослідження вимог до майбутньої системи, сформовано основні функціональні блоки та варіанти використання. На основі UML-діаграм змодельовано ключові процеси взаємодії користувачів із системою.

Це дало змогу закласти надійне підґрунтя для проектування архітектури програмного забезпечення та логічної структури даних.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних

Логічна модель даних для програмного забезпечення з обліку графіків робочих змін персоналу відображає структуру зберігання та взаємозв'язків між сутностями, необхідними для організації ефективного управління робочими змінами [9]. В основі моделі знаходиться таблиця користувачів, яка містить інформацію про всіх працівників, їхні імена, електронні адреси, хешовані паролі та ролі, які вони виконують у системі. Кожен користувач може мати певну роль, що визначається через таблицю ролей, яка включає унікальні ідентифікатори ролей та їх назви (див. рис. 2.1). Це дозволяє гнучко керувати доступом і правами користувачів у системі.

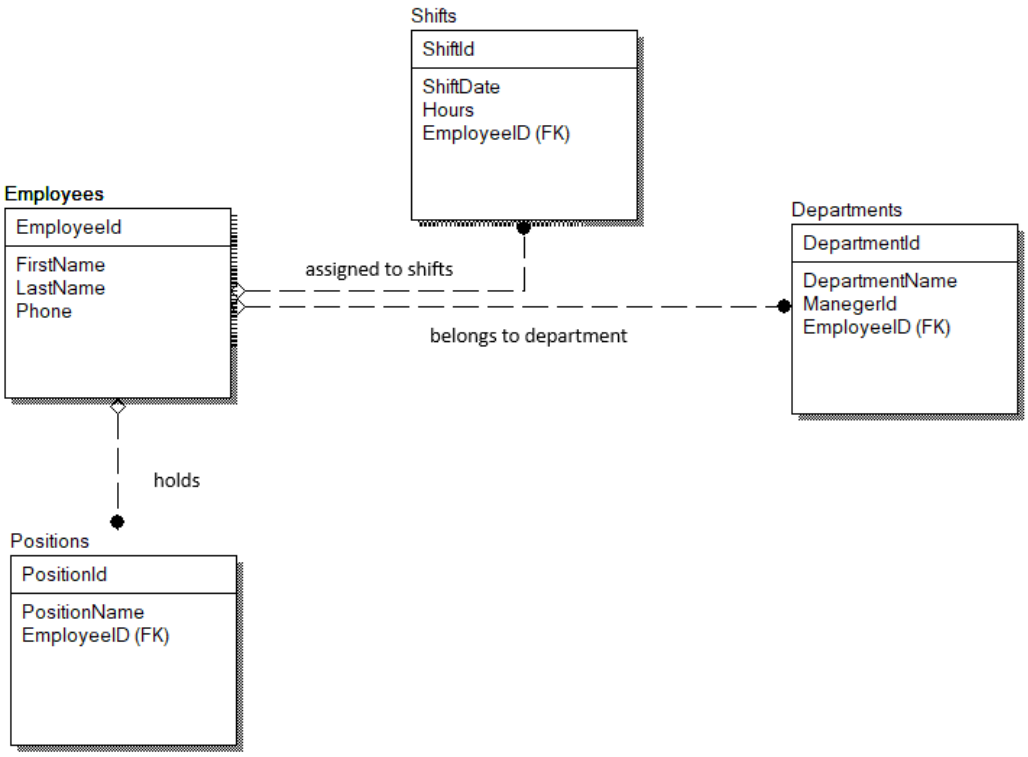

Рис. 2.1 Логічна модель даних

Робочі зміни користувачів фіксуються в таблиці змін, де зберігається інформація про розклад змін, їхню дату, час початку та закінчення, а також опис. Ці зміни пов’язані з таблицею розкладів, яка визначає часові рамки для користувачів та містить інформацію про затвердження змін адміністратором або відповідальною особою. Зміни користувачів можуть бути прийняті чи відхилені, що забезпечується полем затвердження у таблиці розкладів.

Для забезпечення прозорості і відстежуваності дій користувачів у системі, існує таблиця історії дій, де фіксуються всі виконані дії користувачами разом із часом їх здійснення.

Аналіз представленої логічної моделі підтверджує тенденцію до побудови гнучких, реляційно-орієнтованих архітектур, що легко масштабуються й інтегруються з іншими сервісами. На відміну від багатьох застарілих рішень, у яких дані розкидані між кількома неузгодженими файлами або аркушами, запропонована модель забезпечує централізоване управління інформацією з чіткими зв’язками між таблицями, що значно спрощує контроль доступу, змін і автентифікацію користувачів.

Особливістю цієї реалізації є використання логіки відокремлення контекстів – наприклад, розмежування між графіками (schedules) і змінами (shifts) дозволяє відслідковувати не лише поточний стан, а й історію складання графіка, відхилення від плану, та статус його затвердження. На відміну від комерційних систем, які часто не підтримують аудит змін на рівні окремих дій, у запропонованій моделі використовується таблиця history_logs, яка дозволяє зафіксувати будь-яку подію в системі, що має значення для безпеки та звітності.

Інтеграція таблиці notifications вказує на орієнтацію системи не лише на внутрішню адміністративну функціональність, а й на комунікацію з кінцевим користувачем. Це є суттєвим кроком уперед порівняно з рішеннями, де сповіщення реалізовані виключно через зовнішні сервіси без збереження історії. У той час як більшість SaaS-систем не дозволяє користувачеві отримувати деталізовану інформацію про дії або зміни у графіку без відкриття окремого інтерфейсу, модель

із внутрішніми сповіщеннями полегшує взаємодію та мінімізує ризик втрати важливих даних.

2.2 Вибір системи управління інформаційною базою

СУБД визначає те, як саме зберігатимуться, оброблятимуться та надаватимуться в користування дані, які є важливими для функціонування програми. Адже саме від цього залежить ефективність виконання запитів, масштабованість, надійність і безпека зберігання інформації, а також можливість інтеграції з іншими компонентами. У межах проєкту для зберігання даних про графіки робочих змін, працівників, ролі користувачів, історію змін, звіти та сповіщення було вирішено використовувати MySQL Server.

Вибір MySQL Server аргументований тим, що це один з найпоширеніших і найкраще підтримуваних реляційних СУБД з відкритим початковим кодом, яка активно розвивається як спільнотою, так і комерційною організацією Oracle [10]. Це дає змогу розраховувати на широкий спектр супутніх інструментів, бібліотек, документації та матеріалів, які допоможуть у розгортанні та супроводі системи. Практично для кожної популярної мови програмування існують зручні драйвери й ORM-фреймворки, які спрощують роботу з MySQL на рівні коду, що суттєво полегшує завдання з’єднання клієнтської або серверної частини з базою.

MySQL добре зарекомендувала себе в багатьох проєктах різних масштабів, починаючи від невеликих локальних систем до високонавантажених веб-додатків зі складними запитами й великими обсягами даних. У випадку програми з обліку графіків робочих змін, яка передбачає помірне або інтенсивне використання бази внаслідок регулярної взаємодії менеджерів змін, працівників і адміністраторів, MySQL забезпечує надійну роботу навіть на доступному апаратному рівні. Технологія підтримує ефективний пошук, швидке оновлення та внесення змін, можливість застосування індексів і різних типів полів, що оптимізує загальний час відгуку системи. MySQL, також, відома своїм досить простим налаштуванням і

доступністю безкоштовних інструментів адміністрування, як-от MySQL Workbench, що дає можливість проєктувати схеми баз даних, візуалізувати таблиці та зв’язки, створювати резервні копії, здійснювати моніторинг продуктивності й виконувати інші операції без складних додаткових витрат.

Додатковою перевагою є висока сумісність MySQL із сучасними хмарними платформами та можливість швидкого розгортання у вигляді контейнерів, наприклад з використанням Docker. MySQL підтримує транзакції (InnoDB) і механізми відновлення в разі збоїв, гарантує цілісність даних під час виконання багатьох паралельних операцій, передбачає створення резервних копій та реплікацію. У галузі обліку робочих змін подібні інструменти надзвичайно важливі, адже кожна зміна, затвердження чи редагування має відображатися коректно і надійно фіксуватися в базі. Можливість налаштування багаторівневого доступу до бази даних із виділеними привілеями для різних користувачів (наприклад, адміністратора БД, менеджера з обслуговування та ін.) також сприяє підтриманню безпечного середовища. Коли йдеться про зберігання персональних даних працівників або критичних журналів змін, важливо реалізовувати обмеження доступу на рівні бази, щоб унеможливити несанкціонований перегляд або модифікацію інформації.

Дана система управління дає змогу працювати з різними типами даних: датами, часовими мітками, числовими значеннями, текстовими полями. Це уможливлює легку реалізацію сценаріїв з обліку робочих змін, де важливими є дата, час початку, час закінчення, тривалість зміни, відомості про працівників чи керівників тощо. Розширена підтримка індексів і можливість створення складених індексів для полів часу, дат і ідентифікаторів працівників суттєво покращує швидкість виконання вибірок, фільтрації та формування звітів. Отримання інформації про робочу зміну з бази, що містить велику кількість записів, може відбуватися швидко й без надмірного навантаження на сервер.

У разі збільшення обсягів даних або кількості одночасних підключень MySQL дозволяє ефективно балансувати навантаження, розгортати кластери, реалізовувати реплікацію з кількох джерел або налаштовувати розподілену

інфраструктуру. Гнучка підтримка міграцій і резервного копіювання полегшує перехід між середовищами – від локального розгортання до хмарного. Безкоштовна ліцензія MySQL, сумісність з відкритими інструментами та підтримка великої спільноти розробників забезпечують ще одну суттєву перевагу: швидке реагування на проблеми, велика кількість навчальних ресурсів і бібліотек, що значно спрощує як розробку, так і подальший супровід програмного продукту.

2.3 Створення інформаційної бази

Для реалізації бази даних було обрано СУБД MySQL Server – потужну, надійну та гнучку платформу, яка підтримує реляційну модель даних, забезпечує високу швидкодію запитів, масштабованість та простоту адміністрування [11]. У даному випадку логіка створення бази починається зі створення окремої логічної одиниці даних – бази під назвою «rotapro», яка об’єднує в собі всі таблиці, що належать до предметної області управління робочими змінами. Саме вона виступає як контейнер, в межах якого визначається структура таблиць, встановлюються зв’язки та вводяться дані.

Після створення бази відбувається побудова двох основних таблиць: Employees та Shifts, кожна з яких виконує свою важливу функцію. Таблиця Employees слугує як основне джерело інформації про персонал, у ній зберігаються поля з унікальним ідентифікатором кожного співробітника, його ім’ям, прізвищем, посадою та контактним номером телефону. Ключове поле EmployeeId є первинним ключем таблиці і має властивість AUTO_INCREMENT, що дозволяє автоматично генерувати унікальні ідентифікатори при кожному новому додаванні працівника. Також для полів FirstName та LastName встановлено обмеження NOT NULL, що означає обов’язковість заповнення цих полів, оскільки вони є основними для ідентифікації співробітника в межах людського сприйняття. Поля Position та Phone є додатковими, однак їх присутність забезпечує зручний і повноцінний облік персоналу, зберігаючи критичні атрибути працівника.

Таблиця Shifts є підлеглою, і саме в ній фіксується інформація про змінний графік. Кожен запис у цій таблиці відповідає одній зміні одного працівника на конкретну дату. Тут так само використовується первинний ключ ShiftId з автоматичною інкрементацією, що забезпечує унікальність кожного запису. Ключовим елементом зв'язку є поле EmployeeId, яке оголошується як зовнішній ключ і посилається на поле EmployeeId з таблиці Employees. Завдяки цьому між таблицями встановлюється чіткий зв'язок типу один до багатьох: один працівник може мати багато змін, але кожна зміна належить лише одному працівнику. Цей зв'язок реалізується механізмом FOREIGN KEY, який не лише створює логічну залежність між таблицями, а й накладає обмеження на цілісність даних: не можна додати зміну, що посилається на неіснуючого працівника. Окрім того, додатково встановлено каскадне видалення (ON DELETE CASCADE), що означає, що у разі видалення працівника з таблиці Employees, усі пов'язані з ним зміни автоматично видаляються з таблиці Shifts. Це гарантує консистентність інформації: база не зберігає покинутих змін, які більше ні до кого не належать.

Поля ShiftDate і Hours є основними атрибутами зміни. Перше з них визначає, коли саме відбувається зміна, а друге – її тривалість у годинах. Обидва поля мають обмеження NOT NULL, адже кожна зміна має відбуватися в певну дату й мати конкретну тривалість. З такою структурою можна легко відображати графік працівника, формувати звіти про його навантаження, проводити аналіз ефективності роботи відділів, а також здійснювати планування на майбутні періоди. Завдяки простоті структури та її логічній ясності таблиця Shifts є надзвичайно ефективною при виконанні запитів, оскільки дозволяє виконувати фільтрацію за датою, групування за працівниками або підрахунок сумарних годин за певний період. Із застосуванням індексів ці операції виконуються ще швидше, особливо при роботі з великими обсягами даних.

Додавання даних до створеної бази rotapro відбувається за допомогою SQL-оператора INSERT, спочатку в таблицю Employees, щоб зафіксувати всіх працівників, а потім – у Shifts, щоб записати їхні зміни. Важливо дотримуватись цієї послідовності, адже при додаванні зміни в таблицю Shifts СУБД перевіряє

наявність відповідного працівника у таблиці Employees. У наведеному прикладі база одразу заповнюється реальними даними: чотири працівники з різними посадами та телефонами, і для кожного з них фіксуються одна або кілька змін на різні дні з різною тривалістю.

З точки зору логіки застосування, така структура підтримує всі ключові процеси в системі. Вона дозволяє гнучко масштабуватися – додавання нового працівника не потребує змін у структурі таблиць, як і додавання нової зміни. Система може працювати як у малому колективі, так і на великому підприємстві. MySQL Server забезпечує стабільність, контроль доступу, цілісність і швидкодію навіть при роботі з великими обсягами інформації, а завдяки своїй відкритості і підтримці стандарту SQL дозволяє легко інтегрувати базу з будь-яким інтерфейсом, побудованим на сучасних мовах програмування.

Висновок

У цьому розділі побудовано логічну модель даних, яка охоплює всі основні сутності, необхідні для повноцінного функціонування системи. Базу даних спроєктовано з дотриманням принципів нормалізації, що забезпечує структурованість і відсутність надмірностей.

Вибір СУБД MySQL обґрунтовано її надійністю, високою продуктивністю та підтримкою транзакцій. Створено таблиці з правильно визначеними зв’язками та ключами, реалізовано логіку збереження, підтвердження та історії змін. Такий підхід гарантує надійне та цілісне зберігання даних із можливістю масштабування системи у майбутньому.

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Організаційна структура програмного забезпечення

Організаційна структура програмного забезпечення (див. рис. 3.1) відображає чітке структурування логічних, функціональних та допоміжних компонентів, що забезпечують узгоджену взаємодію між елементами програми, підвищену масштабованість і підтримуваність рішення, а також гнучкість його адаптації до змін вимог або середовища експлуатації. Архітектура розподілена на декілька головних частин, кожна з яких виконує окрему роль у повному циклі життєдіяльності системи. Центральним осередком є основна директорія ROTAPRO, у якій розташовані всі компоненти, пов'язані як із виконуваним файлом, так і з розробкою, компіляцією, налагодженням та управлінням залежностями.

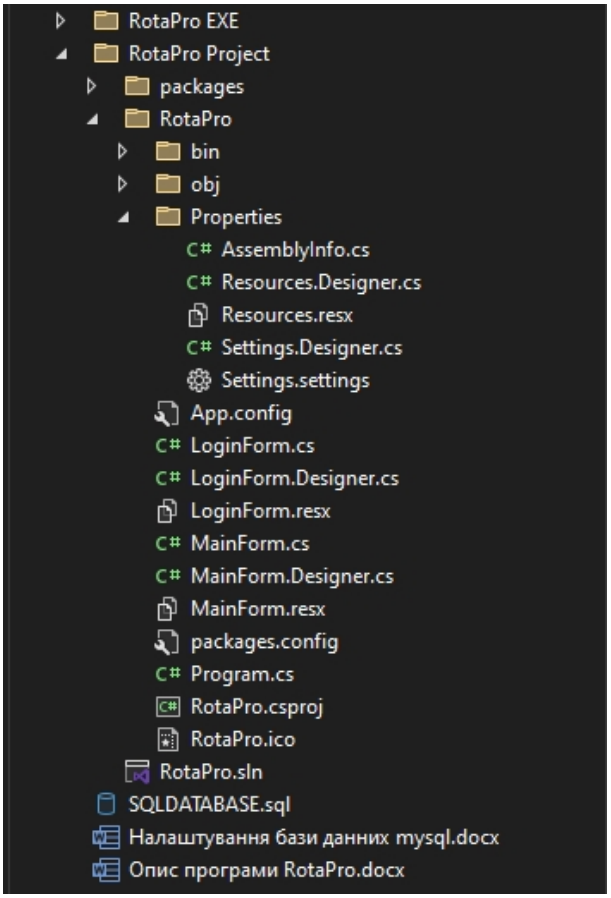

Рис. 3.1 Структура програмного забезпечення

Компонент RotaPro EXE вказує на підготовлений до запуску виконуваний файл, який є фінальним продуктом, доступним для кінцевого користувача. Цей файл є результатом багатьох внутрішніх процесів, що відбуваються всередині папки RotaPro Project, яка є центральним вузлом розробки. Усередині неї зберігаються важливі технічні каталоги, такі як .vs, що використовується середовищем розробки Visual Studio для збереження сесійних налаштувань, тимчасових файлів і метаданих проекту, а також каталог packages, який містить усі зовнішні бібліотеки та залежності, що підключаються через NuGet-пакетний менеджер для розширення функціональності додатку.

Серед переліку пакетів у директорії packages можна побачити широкий набір інструментів, які відповідають за різні аспекти програми: шифрування (BouncyCastle.Crytography), обробку даних (Google.Protobuf, System.IO.Pipelines, System.Memory, System.Buffers), роботу з потоками (System.Threading.Tasks.Extensions, System.Diagnostics.DiagnosticSource), а також специфічні бібліотеки для стискання даних (K4os.Compression.LZ4, ZstdSharp.Port) і логування (Microsoft.Extensions.Logging.Abstractions). Драйвери для роботи з базами даних MySql.Data і MySqlConnector – важливі для забезпечення зв’язку з сервером збереження графіків робочих змін персоналу.

Каталог RotaPro є безпосередньо тією частиною, в якій зосереджена вся вихідна кодова база розроблюваного додатку. У ньому, зокрема, знаходяться каталоги bin і obj, що генеруються автоматично під час компіляції та містять відповідно зібрані виконувані файли й проміжні об’єкти. Папка Properties містить файли конфігурацій ресурсів, таких як іконки, локалізація, параметри додатку тощо. Сюди ж входить і файл AssemblyInfo.cs, який містить метадані збірки, включаючи версію, авторські права та інше.

Файли LoginForm.cs, MainForm.cs та їх відповідні дизайнерські і ресурсні файли демонструють використання WinForms як технологічного підґрунтя для побудови графічного інтерфейсу користувача. Тут реалізуються функції авторизації, головного вікна управління змінами та інші критично важливі елементи інтерфейсу. Їх супроводжують файли .Designer.cs, які генеруються

автоматично середовищем розробки і містять декларації UI-компонентів, і файли .resx, що відповідають за збереження ресурсів, таких як текстові повідомлення, зображення чи налаштування для мультимовності.

Файл App.config забезпечує конфігурацію програми на рівні запуску, наприклад, вказуючи строки підключення до бази даних, параметри логування, таймаути тощо. Сам запуск програми ініціюється через Program.cs, у якому прописується вхідна точка виконання. Проєктне ядро представлено у вигляді файлу RotaPro.csproj, що описує структуру проекту, залежності та інструкції для компіляції. RotaPro.sln об'єднує всі пов’язані проєкти в один логічний простір, і файл SQLDATABASE.sql, що є основою для побудови або ініціалізації бази даних, структуру таблиць якої, найімовірніше, використовує система обліку графіків робочих змін.

Весь цей складний і багатошаровий набір компонентів забезпечує коректну та ефективну роботу програмного забезпечення та ілюструє фундаментальні принципи структурної організації проєктів у середовищі .NET: суворе розділення логіки, інтерфейсу, ресурсів і залежностей; підтримка життєвого циклу проєкту від розробки до релізу; а також наявність інструментарію для гнучкого управління компонентами, що значно полегшує підтримку, тестування та розвиток системи у майбутньому.

3.2 Вибір інструментарію для створення ППЗ

Вибір інструментарію для створення програмного забезпечення «Програмне забезпечення з обліку графіків робочих змін персоналу» є одним із визначальних факторів успіху всієї системи, адже саме від правильного поєднання мов програмування, баз даних, середовищ розробки та допоміжних бібліотек залежить, чи буде майбутній продукт не лише функціональним, а й зручним, гнучким, стабільним та масштабованим у довгостроковій перспективі. Під час розробки такого специфічного рішення було прийнято рішення використати C# як основну

мову програмування у поєднанні з системою управління базами даних MySQL, що створює синергію між потужністю об'єктно-орієнтованого підходу та продуктивністю перевіреної часом реляційної моделі даних [12].

C# – це сучасна, об'єктно-орієнтована мова програмування, створена корпорацією Microsoft у рамках платформи .NET. Вона була розроблена як відповідь на потребу у мові, яка поєднувала б продуктивність C++, зручність Java та гнучкість сучасних мов високого рівня. C# має строгий, чітко структурований синтаксис, який базується на принципах безпеки типів, модульності, інкапсуляції, успадкування та поліморфізму. У випадку даної розробки, ця мова дозволяє легко оперувати складними структурами даних, створювати логіку перевірок і розрахунків, реалізовувати інтерфейси взаємодії з користувачем, базами даних, файлами і мережею [13].

Одним з головних переваг C# є її інтеграція з екосистемою .NET – це величезна платформа, яка включає в себе середовище виконання (CLR – Common Language Runtime), бібліотеки класів, інструменти розробки, API для роботи з базами даних, файловими системами, мережею, шифруванням, інтерфейсами користувача, а також підтримку сучасних підходів до програмування, таких як реактивне програмування, тестування, розробка REST-сервісів і мікросервісів. У межах .NET Framework або .NET Core (а тепер уже й .NET 5/6/7/8) C# дозволяє створювати кросплатформенні застосунки, але в контексті десктопного застосунку, як у випадку «Програмного забезпечення з обліку графіків робочих змін персоналу», найчастіше використовується технологія Windows Forms.

Windows Forms (або WinForms) – одна з найстаріших, але досі актуальних технологій для створення графічного інтерфейсу користувача (GUI) у середовищі Windows [14]. Вона надає розробнику засоби створення форм, кнопок, списків, полів введення, вкладок, діалогових вікон та інших елементів інтерфейсу, які взаємодіють з користувачем у реальному часі. WinForms дозволяє швидко побудувати інтерфейс «що бачиш – те й отримаєш», використовуючи візуальний редактор у Visual Studio. Розробник має можливість обробляти події – наприклад, натискання кнопки або вибір елемента зі списку – за допомогою відповідних

методів, які чітко пов’язані з елементами на формі. У випадку даної розробки – це форма логіну, основна форма з розкладом, таблиці з інформацією про працівників, тощо.

У середовищі Windows, яке залишається домінантним на багатьох робочих місцях у державних і комерційних структурах, C# має надзвичайно високий рівень інтеграції завдяки середовищу розробки Visual Studio. Visual Studio – офіційне середовище розробки від Microsoft, яке тісно інтегрується з C# та платформою .NET. Visual Studio не просто текстовий редактор, а повноцінна IDE (Integrated Development Environment), яка надає інструменти для компіляції, налагодження, тестування, автоматичної генерації коду, підключення бібліотек, роботи з Git, інтеграції з базами даних і навіть створення інсталяторів. Розробник у Visual Studio отримує доступ до потужної системи підказок (IntelliSense), автоматичного виправлення помилок, відображення ієрархії класів і побудови діаграм залежностей. Дана середа підтримує створення проєктів різних типів – консольних, веб-застосунків, десктопних рішень, хмарних сервісів тощо.

ADO.NET – це набір класів, які входять до складу .NET і забезпечують доступ до джерел даних, передусім до реляційних баз даних [15]. У проєкті обліку змін персоналу ADO.NET використовується для підключення до MySQL, виконання SQL-запитів, зчитування результатів, вставки, оновлення або видалення записів. Через класи MySqlConnection, MySqlCommand, MySqlDataReader, розробник має змогу виконати будь-яку необхідну операцію з даними, а також налаштувати параметризацію запитів, що забезпечує безпечну роботу з базою, захищаючи систему від SQL-ін’єкцій.

Entity Framework (EF) – це ORM (Object-Relational Mapping) для .NET, який дозволяє взаємодіяти з базою даних через об’єктну модель [16]. Замість ручного написання SQL-запитів, розробник оперує об’єктами, які представляють таблиці, і працює з ними, як зі звичайними колекціями у C#. EF автоматично генерує SQL-запити, синхронізує зміни, підтримує транзакції, lazy loading, кешування та інші функції, які значно спрощують доступ до даних. У складних системах, таких як

програми для обліку змін, EF дозволяє будувати складні зв’язки між об’єктами – працівники, зміни, графіки, історія змін – і обробляти їх як частину єдиної моделі.

NuGet – це менеджер пакетів для .NET, який дає змогу підключати зовнішні бібліотеки, не витрачаючи часу на ручну інсталяцію та налаштування. У проєкті використовуються такі NuGet-пакети, як Microsoft.Extensions.Logging, System.Configuration.ConfigurationManager, System.IO.Pipelines, BouncyCastle.Crytography, K4os.Compression.LZ4, ZstdSharp.Port – кожен із яких розширює функціональні можливості додатку. Наприклад, логування дозволяє зберігати помилки, події, попередження у файли чи базу, шифрування – забезпечує захист конфіденційних даних, стиснення – дозволяє працювати з архівами або зменшити обсяг резервного копіювання.

Обрана система управління базами даних MySQL – безкоштовне, продуктивне, гнучке та відкрите рішення, що дає змогу ефективно організовувати структуру даних у вигляді таблиць з жорстко визначеними типами, зовнішніми ключами, індексацією, тригерами, представленнями та процедурами. У випадку системи обліку змін це особливо важливо, оскільки ми маємо справу з великою кількістю взаємозалежних даних: співробітники, зміни, посади, графіки, підрозділи, відпустки, заміни, чергування тощо. Саме реляційна модель дає змогу зберігати ці взаємозв’язки у строго впорядкованому вигляді з підтримкою цілісності та унікальності. Додатково, MySQL чудово масштабується – за потреби її можна перенести у хмарну інфраструктуру або організувати реплікацію на декілька серверів.

Для зв’язку між C# та MySQL використовуються офіційні та сторонні драйвери, зокрема MySql.Data або MySqlConnector, які підтримують як класичний ADO.NET підхід з використанням об’єктів типу MySqlConnection, MySqlCommand, MySqlDataReader, так і роботу з ORM (Object-Relational Mapping), такими як Entity Framework, що спрощує взаємодію з базою даних, дозволяючи оперувати об’єктами замість ручного формування SQL-запитів.

Значну увагу при реалізації програмного продукту було приділено також розширенню функціоналу через використання додаткових NuGet-бібліотек.

Наприклад, для забезпечення додаткової безпеки й шифрування при зберіганні або передаванні облікових даних чи журналів активності можна використовувати бібліотеку BouncyCastle.Crytography. Для стиснення даних – K4os.Compression.LZ4, для обробки потоків і великої кількості даних у реальному часі – System.IO.Pipelines, для реалізації гнучких асинхронних сценаріїв – System.Threading.Tasks.Extensions. Ці бібліотеки органічно інтегруються з екосистемою .NET, розширюючи межі базового інструментарію та дозволяючи реалізувати більш складну архітектуру, у якій забезпечується гнучкість, продуктивність і захищеність.

У підсумку, обраний інструментарій забезпечує можливість створення функціонального додатку та формує потужну платформу для довгострокового розвитку системи, що охоплює можливості адміністрування користувачів, управління змінами, формування звітів, інтеграцію з іншими системами, збереження резервних копій, контроль доступу та інші життєво важливі функції, необхідні для цифровізації процесів управління персоналом. У поєднанні з грамотним проєктуванням бази даних, багатошаровою архітектурою програмного коду, активним використанням шаблонів проєктування та автоматизованим тестуванням, обраний інструментарій гарантує, що система буде не лише ефективною на момент запуску, але й легко адаптуватиметься до змін у майбутньому.

3.3 Алгоритмізація та програмування програмних модулів

Алгоритмізація та програмування програмних модулів у контексті створення даного програмного забезпечення, це один з найбільш відповідальних і технічно складних етапів розробки, який поєднує в собі логічне проєктування поведінки системи, структурування бізнес-логіки та її реалізацію за допомогою сучасних мов програмування і технологій.

Початковим етапом алгоритмізації є формалізація предметної області. У системі обліку змін персоналу потрібно враховувати безліч факторів: типи змін (денна, нічна, чергова, резервна), тривалість змін, кількість працівників на зміну, урахування вихідних, відпусток, лікарняних, замін, а також можливі конфлікти змін – наприклад, ситуації, коли працівник випадково призначений на дві зміни одночасно. Побудова алгоритмів має спиратися на набір суворих правил, обмежень та перевірок, які покликані підтримувати консистентність даних, забезпечити їх цілісність та логічну узгодженість.

У C# реалізація модулів здійснюється за допомогою класів, структур, інтерфейсів, методів і подій. Основні програмні модулі можна умовно поділити на кілька логічних груп. Перша – це модулі взаємодії з базою даних. Вони забезпечують підключення до MySQL через драйвер MySqlConnector, відкриття та закриття з'єднань, виконання запитів та обробку результатів. Для цього створюються спеціальні класи на зразок DatabaseManager, які інкапсулюють усю логіку доступу до даних і надають розробнику високорівневі методи, наприклад GetEmployeeById, GetShiftsByDateRange, AddNewShift, UpdateSchedule, RemoveEmployeeFromShift. Кожен метод всередині виконує алгоритм перевірки параметрів, формує SQL-запит, передає його у виконання, а після цього отримує результат і перетворює його у зручну для системи форму.

Наступна важлива категорія модулів – це модулі бізнес-логіки, які відповідають за правильну поведінку системи відповідно до заданих алгоритмів. При додаванні нового запису про зміну в графік система має перевірити, чи не зайнятий працівник у цей день іншою зміною, чи відповідає його посада типу зміни, чи не перевищує він допустимий ліміт годин на тиждень. У C# ці перевірки реалізуються через методи з вкладеними умовами, логічними операторами, циклами та обробкою виключень. Алгоритм визначення конфліктів змін може, наприклад, виглядати як послідовне сканування існуючих записів у базі даних і зіставлення дат, ідентифікаторів працівників і типів змін із новими даними, що вводяться.

Також, важлива валідація даних. Користувач, який заповнює форму в програмі, може помилитися або внести неповну інформацію. Для цього використовуються алгоритми валідації: перевірка обов’язкових полів, форматів дати, числових значень, меж допустимих значень. Наприклад, при введенні дати початку зміни необхідно перевірити, що вона не є минулою, або що кінець зміни не настає раніше, ніж її початок. У C# такі алгоритми реалізуються через умовні оператори if, switch, а також конструкції try-catch для перехоплення винятків під час обробки введених даних.

Реалізуються модулі генерації звітності та аналітики, де в основі лежать алгоритми сортування, фільтрації, агрегації та обчислення статистики. Наприклад, щоб створити звіт по заповненості змін за тиждень, програма формує SQL-запит з групуванням і підрахунком (GROUP BY, COUNT(*)), а після отримання результату виводить інформацію в табличному або графічному вигляді. У C# це реалізується або через прямі запити до бази, або через використання LINQ-запитів до локальних колекцій, що зберігають кешовані дані.

Особливої уваги потребує реалізація модулів інтерфейсу користувача, які взаємодіють із логікою програми. У Windows Forms кожна форма має відповідні події – натискання кнопок, зміна вибраного значення, введення тексту, тощо. У відповідь на ці події запускаються алгоритми, які виконують необхідні дії. Наприклад, при натисканні кнопки «Додати зміну», відбувається зчитування введених значень, їх перевірка, формування запиту до бази даних, оновлення відображення графіка. Тут алгоритмічне мислення особливо важливе – розробник має чітко уявляти, у якій послідовності і з якими умовами повинні виконуватись дії, які можливі помилки і як їх обробляти, щоб уникнути збоїв у системі.

Алгоритмізація включає написання юніт-тестів – модулів, які перевіряють коректність окремих алгоритмів. У C# для цього часто використовують фреймворки MSTest, xUnit, NUnit. До прикладу, створюються тести для перевірки алгоритму виявлення конфліктів змін: один тест перевіряє, чи правильно виявляється дублювання змін, інший – чи допускаються зміни у вихідні дні, ще

один – чи викидається помилка при спробі призначити працівника, якого не існує в системі.

У результаті, алгоритмізація та програмування модулів у системі для обліку змін персоналу – це складний, глибоко структурований процес, який охоплює всі аспекти життєвого циклу даних: від їх введення до обробки, зберігання, візуалізації і аналізу.

Важливим аспектом алгоритмізації є реалізація механізмів логування та відстеження дій користувачів, що забезпечується за допомогою модулів журналювання. Кожна значуща дія – додавання зміни, редагування графіка, підтвердження розкладу чи формування звіту – має фіксуватись у системі з прив’язкою до часу та ідентифікатора користувача. У C# це досягається шляхом виклику відповідного методу логування у ключових точках програми, де створюється об’єкт із відповідною інформацією і передається до методу, який формує запис у базі даних.

Щоб забезпечити миттєву інформованість працівників про зміни у графіку, система повинна генерувати повідомлення у відповідь на події, що стосуються користувача: затвердження графіка, додавання нової зміни, її скасування або перенесення. Алгоритм має включати перевірку типу події, формування змісту повідомлення, його збереження в таблиці notifications та маркування як непрочитаного. При наступному вході користувача в систему ці повідомлення відображаються в інтерфейсі, і після перегляду позначаються як прочитані.

При вході в систему користувач вводить електронну адресу та пароль, які передаються до модуля авторизації. Тут проводиться перевірка існування запису, розшифрування та звірка хешу пароля, а також визначення ролі користувача, яка впливає на рівень доступу. У разі успішної автентифікації створюється сесія або токен, який дозволяє системі ідентифікувати користувача протягом роботи. У випадку невдачі вхід блокується, а в модулі журналювання фіксується спроба доступу. Алгоритм додатково може включати обмеження кількості спроб входу для захисту від брутфорс-атак.

Реалізація механізмів перевірки узгодженості змін між різними користувачами також є критично важливою. У ситуаціях, коли кілька адміністраторів одночасно працюють над одним розкладом, система повинна або блокувати редагування на час внесення змін, або реалізовувати механізм «оптимістичної конкурентності» – порівнювати збережений запис з поточним станом і, в разі розбіжностей, попереджати про конфлікт із пропозицією оновити дані.

Додатково, особливу роль відіграють алгоритми генерації динамічних форм введення даних. Залежно від обраного типу зміни, посади користувача або дати, система формує інтерфейс з відповідним набором полів.

При побудові таких модулів велике значення має логічна декомпозиція коду. Замість того, щоб реалізовувати весь алгоритм в одному методі, код розбивається на серію підпроцедур із чітко визначеними обов’язками: валідація, обробка введення, формування запиту, обробка відповіді, оновлення інтерфейсу.

Загалом, алгоритмізація програмних модулів для системи обліку змін – це глибоко аналітичний процес, що поєднує математичну точність, продуману архітектуру, обробку виняткових ситуацій та взаємодію з користувачем на кожному етапі.

Висновок

У розділі описано архітектурну організацію системи, обґрунтовано вибір мови програмування C# і використання середовища Visual Studio, що забезпечує стабільну роботу клієнт-серверного застосунку. Реалізовано основні функціональні модулі, які охоплюють авторизацію, керування графіками, редагування змін, сповіщення, перегляд історії та генерацію звітів.

Використано бібліотеки та технології, які сприяють підвищенню зручності роботи користувача та підтримці безпеки. Побудована структура коду відповідає

принципам модульності, що полегшує підтримку та розширення функціоналу в майбутньому.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Запустивши програму, першим що бачить користувач, це сторінку управління співробітниками (див. рис. 4.1). Тут можна створювати нового співробітника, редагувати та видаляти дані про нього, або ж повністю його видалити.

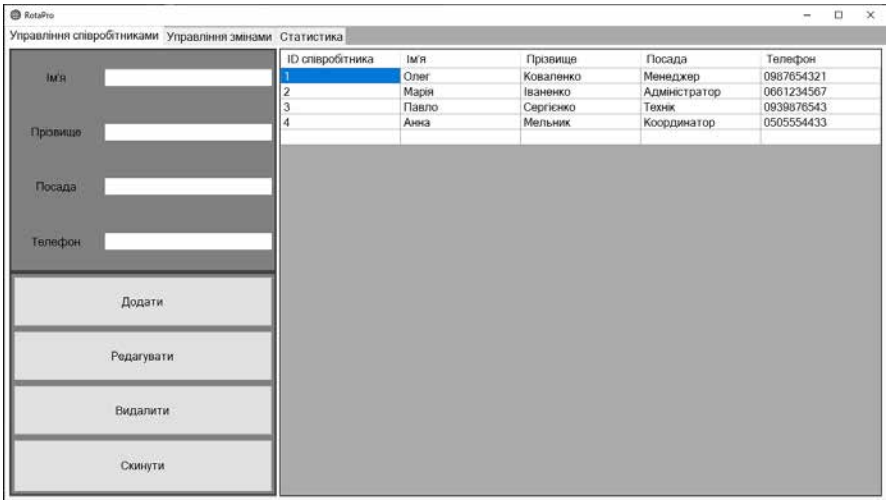

Рис. 4.1 Сторінка управління співробітниками

Додамо нового співробітника заповнюючи необхідні поля та натиснемо кнопку «Додати» знизу під формою введення даних (див. рис. 4.2).

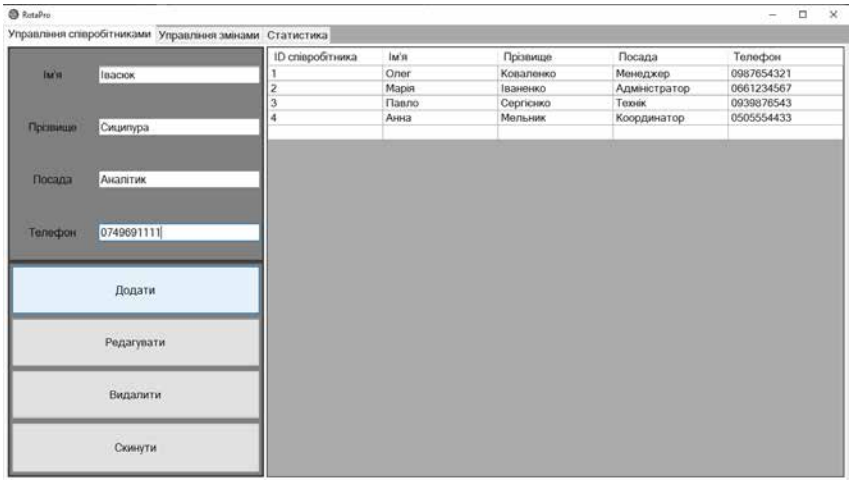

Рис. 4.2 Створення нового співробітника

На рисунку 4.3 зображено результат створення нового співробітника.

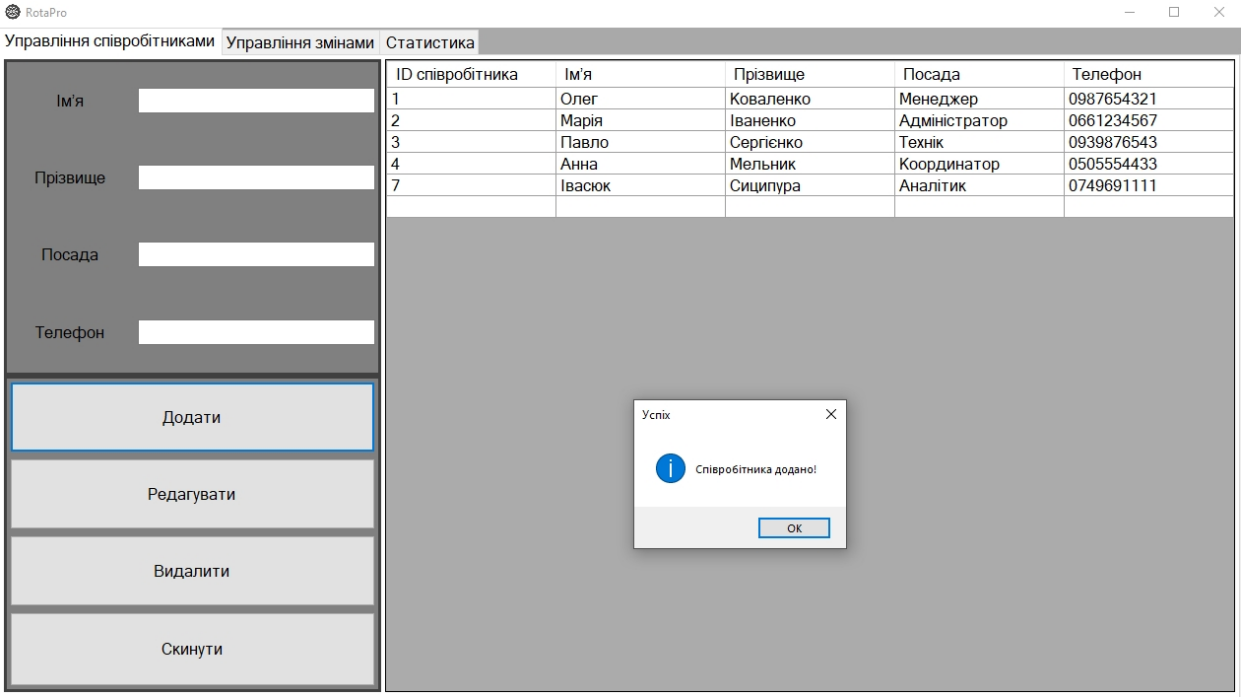

Рис. 4.3 Створення нового співробітника

Натиснувши лівою кнопкою миші на будь-якого співробітника та натиснувши на кнопку «Редагувати», у полях для введення даних, можна буде редагувати інформацію про нього (див. рис. 4.4).

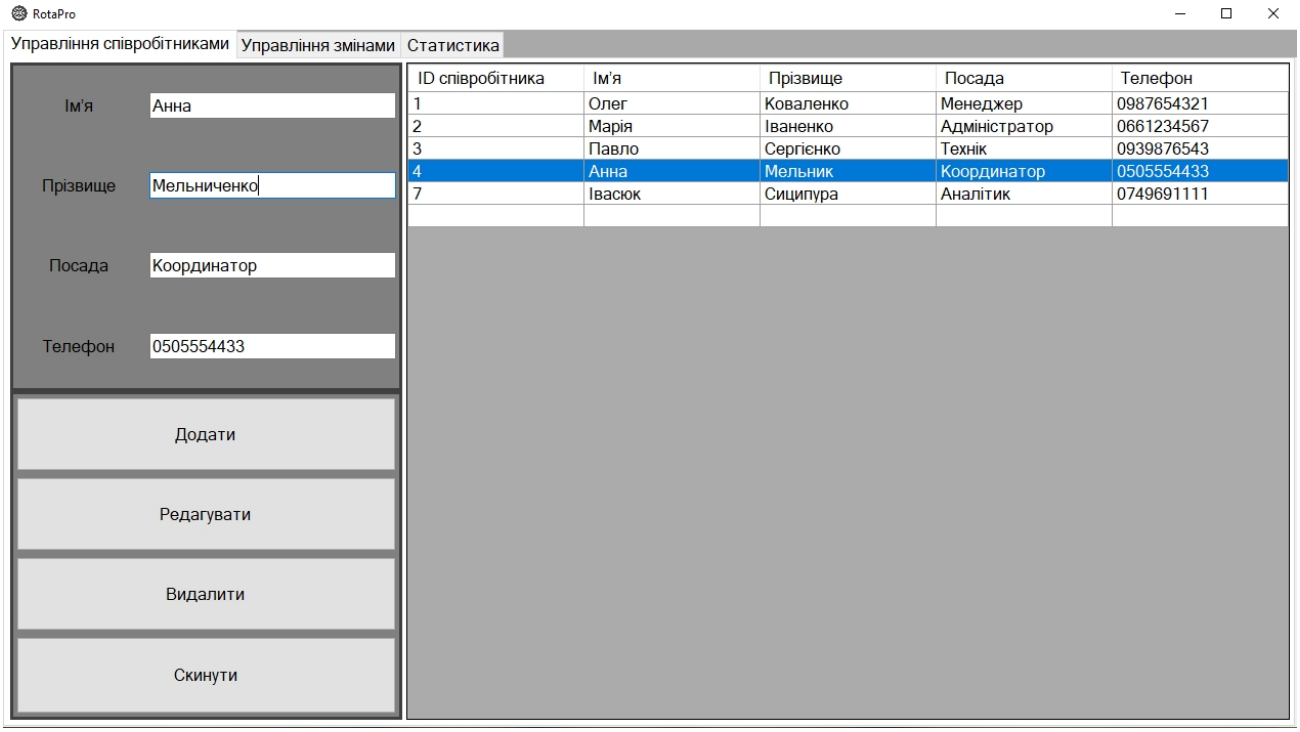

Рис. 4.4 Редагування даних співробітника

Змінивши дані, натиснемо на кнопку «Редагувати», після цього інформацію буде оновлено як на сторінці, так і в базі даних (див. рис. 4.5).

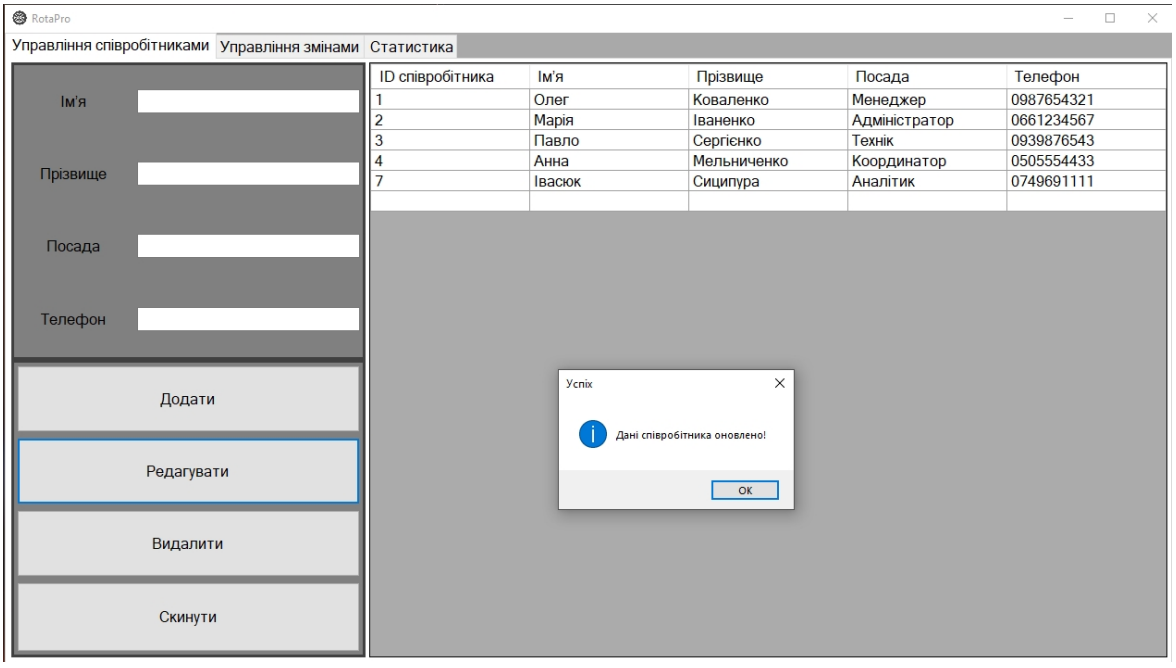

Рис. 4.5 Відредаговані дані співробітника

Щоб видалити співробітника потрібно натиснути на нього лівою кнопкою миші, потім на кнопку «Видалити», яка знаходиться з лівого боку сторінки (див. рис. 4.6).

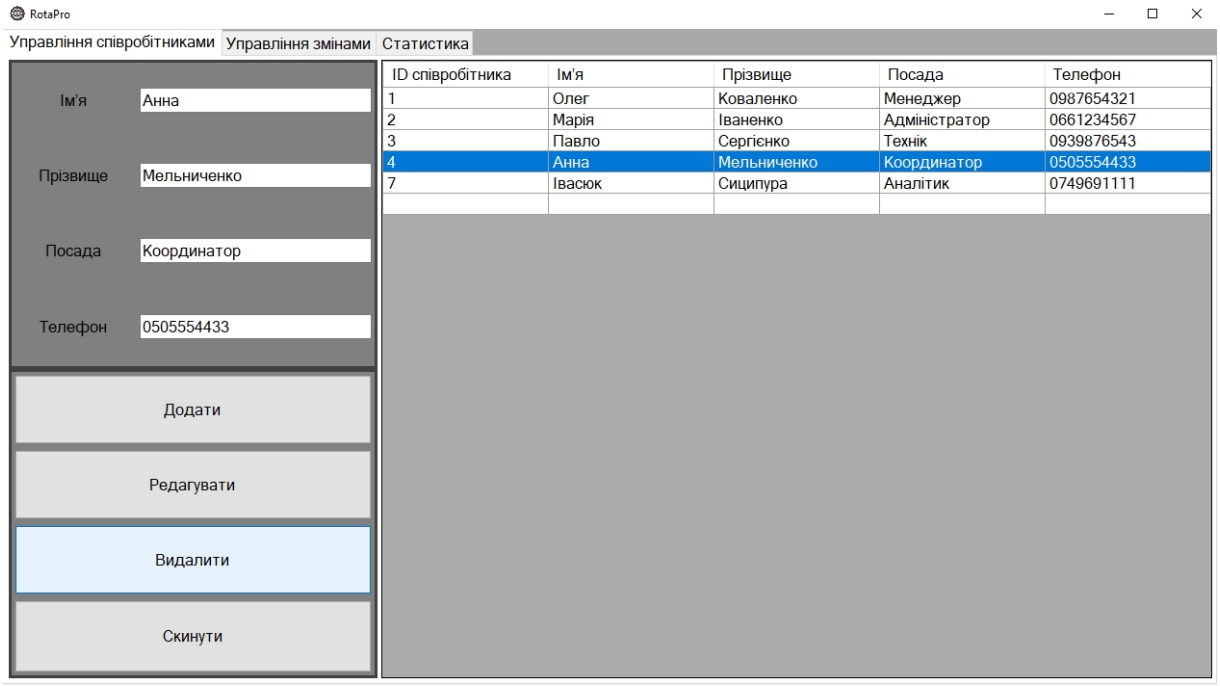

Рис. 4.6 Видалення співробітника «Анна Мельниченко»

Результат видалення співробітника зображено на рисунку 4.7.

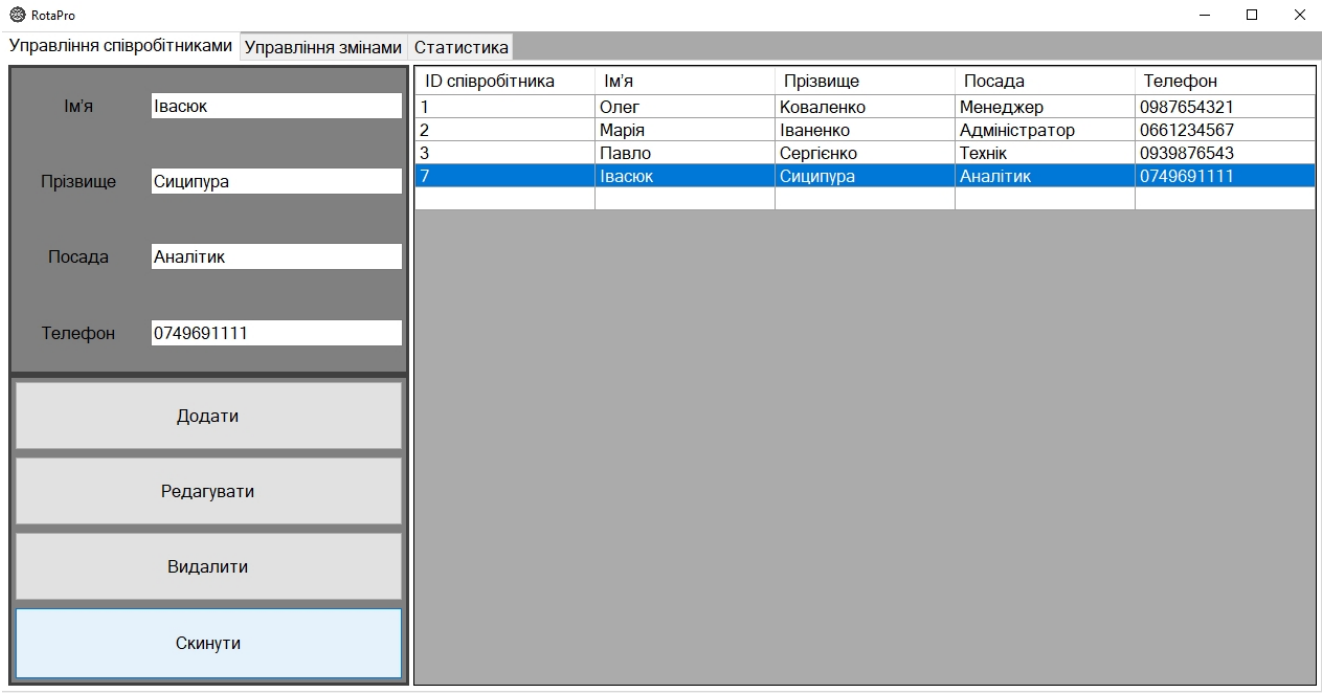

Рис. 4.7 Результат видалення співробітника «Анна Мельниченко»

Кнопка «Скинути» дозволяє очистити всі введені дані у полях, щоб заповнити їх наново. Результат натиску на кнопку «Скинути» зображено на рисунку 4.8.

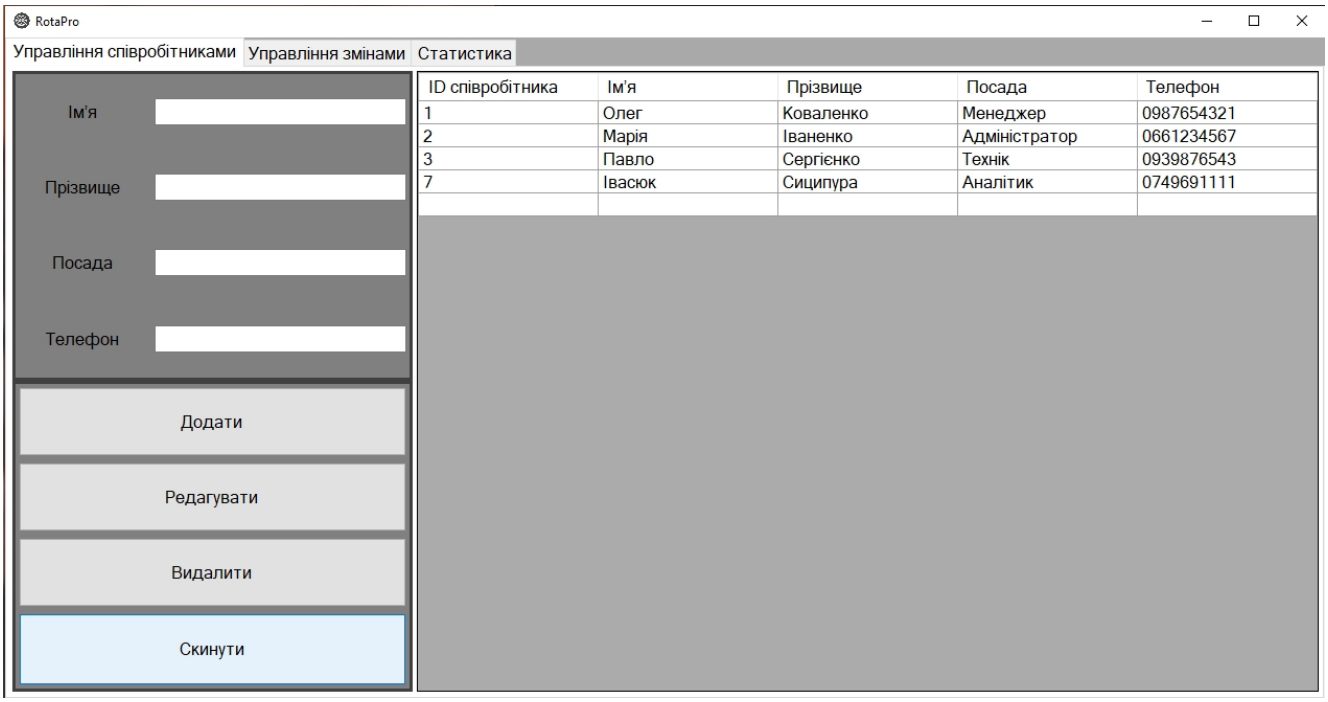

Рис. 4.8 Сторінка після натиску на кнопку «Скинути»

Перейдемо на вкладку «Управління змінами» (див. рис. 4.9). Тут можна побачити 3 секції:

– У першій секції можна обирати працівника, день для його зміни та кількість робочих годин. Також можна додати, редагувати та видалити зміну.

– Друга секція дозволяє переглядати усі зміни які є в базі даних.

– Третя секція дозволяє переглянути календар з днями зміни співробітника.

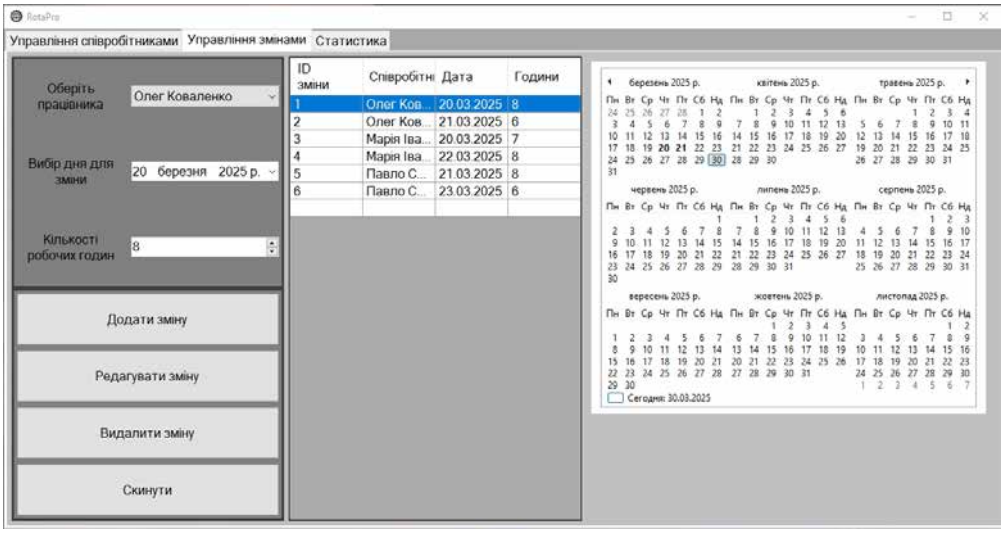

Рис. 4.9 Вкладка «Управління змінами»

На третій вкладці відображається статистика зі змінами працівників, тут можна обирати дати щоб переглядати хто буде на зміні у цей день, генерувати звіти, та переглядати гістограму, яка відображає статистику робочих днів співробітників (див. рис. 10).

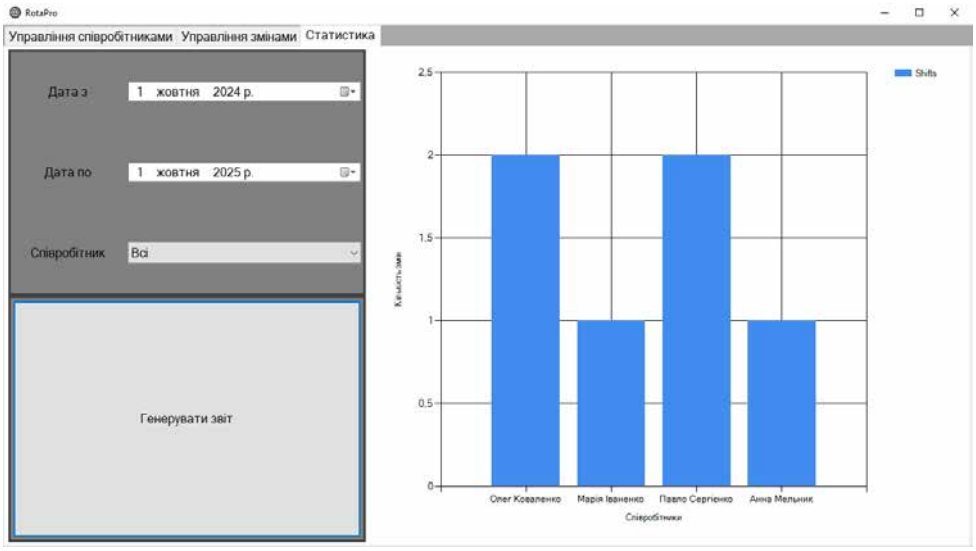

Рис. 4.10 Вкладка «Статистика»

4.2 Вимоги до апаратного та програмного забезпечення

Від правильного визначення мінімальних та рекомендованих характеристик комп’ютерного обладнання, операційного середовища, супутнього програмного забезпечення та мережевої інфраструктури залежить стабільність, швидкодія, та

безперебійність усіх бізнес-процесів, які ґрунтуються на точному, своєчасному та безпечному обліку змін персоналу.

Зважаючи на архітектуру програми, яка створена мовою програмування C# у середовищі .NET Framework із використанням Windows Forms, а також взаємодіє з реляційною системою управління базами даних MySQL, можна сформулювати чіткі вимоги до апаратного забезпечення комп’ютера, на якому ця система буде інсталюватися. Варто зазначити, що для повноцінного функціонування програми рекомендується використовувати настільний ПК або ноутбук із сучасною 64-розрядною архітектурою процесора, адже більшість компонентів .NET, включаючи останні бібліотеки, драйвери та графічні інтерфейси, оптимізовані саме для 64-бітних платформ.

Мінімальні апаратні вимоги можуть включати процесор типу Intel Core i3 або AMD Ryzen 3 з тактовою частотою не нижче 2.0 ГГц. Проте для більш комфортної роботи, особливо якщо програмне забезпечення використовується паралельно з іншими застосунками (наприклад, текстовими редакторами, таблицями, браузерами), рекомендовано використовувати процесори рівня Intel Core i5/i7 або AMD Ryzen 5/7 з частотою 2.5 ГГц і вище. Обсяг оперативної пам’яті також відіграє важливу роль – мінімально допустимим є 4 ГБ, однак для забезпечення швидкої роботи з базами даних, графічним інтерфейсом, формами, таблицями та іншими вікнами, бажано мати принаймні 8 ГБ, а ще краще – 16 ГБ оперативної пам’яті, особливо якщо система використовується в мережевому середовищі з доступом кількох користувачів.

Жорсткий диск або твердотільний накопичувач має забезпечувати достатній обсяг вільного простору як для самої програми, так і для бази даних, резервного копіювання та лог-файлів. Мінімально слід мати принаймні 10 ГБ вільного місця, але з урахуванням розширення бази даних, додавання звітів, збереження архівів змін, скриншотів, документів чи експорту в Excel/CSV, доцільно передбачити диск обсягом від 128 ГБ і більше. При цьому бажано, щоб носій був SSD, адже це істотно пришвидшує завантаження програми, обробку запитів до бази, запуск звітів і загальну реакцію інтерфейсу.

Що стосується графіки, то програма не є ресурсоємною у плані графічного рендерингу, тому вбудованої графіки типу Intel UHD Graphics або AMD Radeon Vega цілком достатньо. При використанні великого монітору або кількох екранів для перегляду розширених графіків змін, таблиць або діаграм, доцільно забезпечити підтримку роздільної здатності Full HD (1920x1080) або навіть вище. Дисплей має бути досить великим – від 15,6 дюймів для ноутбуків і від 21 дюйма для настільних систем, що знижує навантаження на очі та дозволяє комфортно переглядати великі розкладки зміни.

Серед інших апаратних вимог варто згадати необхідність наявності стабільного інтернет-з’єднання (у випадку використання програми в клієнт-серверному режимі або при віддаленій роботі з базою даних), мережевого адаптера (LAN або Wi-Fi).

На рівні програмного забезпечення обов’язковою умовою є встановлена операційна система Windows не нижче версії Windows 10 (64-bit). У деяких випадках система може коректно працювати на Windows 8.1 або Windows 11, однак рекомендованим варіантом залишається саме Windows 10 Pro, яка забезпечує стабільну сумісність із .NET Framework, драйверами MySQL і середовищем Visual Studio. Також має бути встановлений .NET Framework 4.7.2 або новіший, а у випадку використання новітніх бібліотек – повноцінна платформа .NET 6 або .NET 8, що встановлюється окремо.

Для повноцінної роботи з базою даних на серверній або клієнтській машині має бути встановлений сервер MySQL (наприклад, MySQL Community Server 8.0), утиліта для адміністрування (MySQL Workbench, DBeaver або HeidiSQL), а також відповідний драйвер для взаємодії C# із базою (MySql.Data або MySqlConnector). Якщо база даних працює в локальному середовищі – на одному ПК, тоді сервер і клієнт можуть бути розгорнуті на одній машині. Якщо ж йдеться про мережеве середовище, тоді на серверному ПК має бути встановлений MySQL Server з відкритим доступом для підключення з інших машин, налаштовані права доступу, мережевий порт (3306) і захист.

Для запуску, тестування і подальшої модифікації програми необхідно мати встановлене середовище розробки Visual Studio 2019 або новіше (можна безкоштовну версію Community Edition), в якому мають бути встановлені відповідні робочі навантаження: «.NET desktop development», «.NET Core cross-platform development» тощо. Також рекомендується встановити пакет NuGet CLI або використовувати вбудований менеджер пакетів для підключення необхідних бібліотек (наприклад, Microsoft.Extensions.Logging, System.Configuration.ConfigurationManager, K4os.Compression тощо).

Для захисту та збереження даних обов’язковою умовою є налагоджена система резервного копіювання – або автоматична, реалізована на рівні програми, або зовнішня – за допомогою спеціального програмного забезпечення, що копіює базу даних щоденно або щотижнево у захищене місце (хмарне сховище, зовнішній диск, сервер бекапів).

У разі масштабування або інтеграції з іншими системами (наприклад, обліком кадрів, системою табелювання або ERP-системою) необхідно врахувати можливість експорту/імпорту даних, API-інтерфейс або використання формату JSON/XML. Для цього також можуть бути потрібні додаткові бібліотеки або середовище інтеграції.

Окрім базових технічних характеристик і середовищ виконання, важливу роль відіграє стабільність локальної або корпоративної мережі, якщо система використовується кількома користувачами одночасно. У випадку розгортання програми у багатокористувацькому середовищі необхідно передбачити використання внутрішнього файлового або серверного хостингу, що гарантуватиме швидкий доступ до бази даних, обмеження затримок у взаємодії з інтерфейсом та уникнення конфліктів при одночасному доступі. У таких умовах доцільно використовувати виділений сервер із захищеним мережевим протоколом, наприклад, SSL/TLS для шифрування даних, або навіть впровадження VPN-тунелю для захищеної передачі інформації між віддаленими офісами.

Якщо підприємство використовує централізовану систему адміністрування (наприклад, Active Directory), рекомендується інтеграція із засобами авторизації

Windows для забезпечення єдиного входу (Single Sign-On), що, у свою чергу, потребує певного рівня підтримки в операційній системі та налаштувань домену.

При реалізації інтерфейсів для мобільного доступу або браузерного перегляду графіків, що може бути корисним для працівників без постійного доступу до ПК, слід враховувати специфіку роботи з адаптивними вебтехнологіями. У такому випадку сервер повинен мати можливість обробки HTTP-запитів, і необхідне розширення програмної частини за допомогою ASP.NET, REST API або інших відповідних інструментів, що потребує встановлення додаткових компонентів IIS або Apache Tomcat, у разі міжплатформової інтеграції.

З технічної точки зору важливою є й підтримка регулярного оновлення системного і прикладного ПЗ. Для цього доцільно мати централізовану систему керування оновленнями або хоча б можливість встановлення нових версій вручну без втрати даних. Програма повинна мати чітко структуровану логіку збереження конфігурацій, щоб уникнути конфліктів під час оновлення, а резервні копії мають бути автоматично збережені перед кожним критичним оновленням.

4.3 Склад інсталяційного пакету

Склад інсталяційного пакету – важлива складова частина загальної архітектури рішення, оскільки саме від його структури залежить правильність розгортання, сумісність компонентів, можливість запуску на кінцевому комп’ютері користувача та забезпечення повного функціонального набору програми. На підставі візуалізованого вмісту директорії RotaPro EXE, яка виконує роль зібраного інсталяційного пакету або підготовленої до запуску версії програми, можна проаналізувати детальну структуру і призначення кожного з елементів, які входять до складу розгортання.

Фактично, інсталяційний пакет або збірка програми складається з головного виконуваного файлу RotaPro.exe, конфігураційного файлу RotaPro.exe.config,

бібліотек динамічного компонування (DLL), файлів документації у форматі XML та допоміжних налагоджувальних файлів, таких як PDB (RotaPro.pdb). Сам файл RotaPro.exe є ядром всієї програми – саме він містить компільований код, який запускається у середовищі .NET CLR і починає виконання з методу Main() у класі Program.cs. Це головний файл, який ініціалізує завантаження форм, підключення до бази даних, обробку інтерфейсних подій та інші ключові процеси.

Файл RotaPro.exe.config виконує роль конфігураційного центру, у якому зберігаються важливі параметри, такі як строки підключення до MySQL-сервера, шляхи до ресурсів, рівень логування, налаштування середовища виконання. Це XML-файл, який дозволяє змінювати поведінку програми без перекомпіляції, що є надзвичайно зручно для адміністраторів.

Найбільш об’ємну частину інсталяційного пакету становлять динамічні бібліотеки (.dll), які фактично реалізують модульність системи. Наприклад, бібліотеки MySql.Data.dll та MySqlConnector.dll – це драйвери, що забезпечують прямий зв’язок між програмою і базою даних MySQL. Без них неможливо виконати жодного SQL-запиту, жодної транзакції або операції із зчитування, додавання, редагування чи видалення інформації.

Бібліотеки з префіксом System. (наприклад, System.Buffers.dll, System.Memory.dll, System.IO.Pipelines.dll, System.Numerics.Vectors.dll) – це частина стандартних компонентів платформи .NET, які відповідають за роботу з пам’яттю, потоками, буферами, математичними операціями, управлінням асинхронністю. Вони гарантують, що застосунок може обробляти великі обсяги даних, запускати паралельні задачі, реалізовувати затримки, маніпуляції з потоками, тощо.

Такі бібліотеки як Microsoft.Extensions.Logging.Abstractions.dll і Microsoft.Extensions.DependencyInjection.Abstractions.dll вказують на використання сучасних підходів у програмуванні – зокрема, залежні впровадження (Dependency Injection) та централізоване логування. Це означає, що програма має внутрішню структуру, яка дозволяє зручно керувати залежностями між об’єктами, що спрощує тестування, підтримку, розширення функціоналу. А логування подій – це

невід’ємна частина будь-якої зрілої системи, що дозволяє фіксувати дії користувачів, помилки, збої, час виконання задач тощо.

У пакеті також наявні бібліотеки стиснення і шифрування: K4os.Compression.LZ4.dll, ZstdSharp.dll, BouncyCastle.Cryptography.dll. Їх присутність свідчить про обробку конфіденційних або об’ємних даних у системі – наприклад, для шифрування облікових даних, захищеного збереження логів або резервних копій, а також для стиснення даних перед архівацією або пересиланням. Бібліотека Google.Protobuf.dll натякає на можливе використання протоколу Protocol Buffers – це технологія серіалізації даних, яка застосовується для ефективного обміну структурованою інформацією між компонентами системи або для експорту в машиночитабельному форматі.

Файли .xml, які мають такі ж назви як відповідні .dll, є документацією до бібліотек. Вони використовуються середовищем розробки (наприклад, Visual Studio) для відображення підказок (IntelliSense), коментарів до методів, опису параметрів. У кінцевому користуванні вони не є обов’язковими, однак можуть бути корисними під час підтримки системи чи її доопрацювання.

Файл RotaPro.pdb – це файл налагодження (debug symbols), який дозволяє при виникненні помилок точно визначити, на якій стрічці коду сталася помилка, у якому класі, який був стан змінних тощо. Він необхідний у період тестування і налагодження, але в релізній версії може бути вилучений для підвищення безпеки і зменшення розміру пакету.

Висновок

Розділ містить детальний опис вимог до апаратного та програмного забезпечення, підготовку інсталяційного пакету та рекомендації щодо впровадження. Результати тестування свідчать про коректну роботу всіх основних функцій, відсутність критичних помилок і стабільність програми при типових навантаженнях. Описано порядок запуску програми, встановлення та типові сценарії використання. Система адаптована для застосування у підприємствах різного масштабу, має низькі вимоги до ресурсів і може бути використана як у локальному, так і в мережевому середовищі.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було спроєктовано та реалізовано повноцінну клієнт-серверну програмну систему, що охоплює всі основні етапи життєвого циклу графіка змін – від його створення до архівації. Основною метою проєкту було забезпечення автоматизації процесів управління позмінною роботою, мінімізація людського чинника, підвищення прозорості, керованості та ефективності планування трудових ресурсів. Результати роботи підтверджують досягнення поставленої мети в повному обсязі: розроблена система дозволяє створювати та змінювати розклади, уникати конфліктів змін, призначати працівників відповідно до посад та кваліфікації, формувати звіти, а також забезпечує безпечне зберігання та доступ до персональних даних.

На відміну від більшості універсальних офісних застосунків (Microsoft Excel, Google Sheets), що традиційно використовуються для ведення змін, реалізоване програмне забезпечення орієнтоване саме на специфіку графіків персоналу і враховує динаміку змін, типи змін, права доступу, внутрішні обмеження та логіку підтвердження. На відміну від комерційних рішень, таких як Kronos чи Humanity, які часто потребують ліцензійної оплати, наша система базується на вільному стеку технологій (C#, MySQL), що робить її більш економічно доцільною для впровадження в невеликих та середніх організаціях, де фінансові ресурси обмежені.

З технічної точки зору, у роботі було реалізовано низку інноваційних підходів. Основою рішення став чітко структурований об’єктно-орієнтований підхід до моделювання предметної області. Було використано сучасні засоби UML-моделювання для побудови діаграм варіантів використання, класів, активності, компонентів і розгортання, що дозволило всебічно описати архітектуру та поведінку системи. Створена реляційна модель даних з використанням MySQL відповідає всім вимогам нормалізації та забезпечує цілісність, узгодженість і швидкодію при роботі з великими обсягами записів. На рівні програмної реалізації впроваджено модульну архітектуру з чітким розділенням бізнес-логіки, доступу до

даних та інтерфейсу користувача, що підвищує підтримуваність і масштабованість системи.

Одним із нових технічних рішень, реалізованих у межах проєкту, є алгоритм перевірки конфліктів змін, який дозволяє виявляти дублювання змін, перевищення тижневих норм годин, призначення на несумісні зміни та інші ситуації, що порушують регламент. Цей алгоритм розроблений з урахуванням реальних сценаріїв експлуатації системи та є унікальним внеском автора.

Оцінюючи техніко-економічну ефективність розробки, можна стверджувати, що її впровадження дозволяє значно скоротити часові витрати на формування графіків, знизити ймовірність помилок при розрахунках, зменшити навантаження на кадрову службу та керівників підрозділів. Система також має позитивний вплив на дисципліну праці та управління персоналом загалом, оскільки забезпечує прозорість розкладів і підвищує рівень відповідальності користувачів. Оскільки застосунок не потребує ліцензійних відрахувань і має незначні вимоги до апаратного забезпечення, витрати на впровадження є мінімальними, а період окупності – коротким.

Отримані результати дозволяють рекомендувати розроблене програмне забезпечення до впровадження в організаціях з позмінною формою праці: виробничих підприємствах, медичних установах, об’єктах енергетики, логістичних компаніях, охоронних фірмах, закладах торгівлі та інших структурах. За умови подальшого доопрацювання можливе створення мобільної версії, розширення API для інтеграції з іншими обліковими або ERP-системами, реалізація модулів обліку годин, обліку відпусток та синхронізації з табельними системами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Життєвий цикл програмного забезпечення. URL: https://javarush.com/ua/quests/lectures/ua.questservlets.level15.lecture00(дата звернення: 25.03.2025).
2. Основи менеджменту: Конспект лекцій [Електронний ресурс] : навч. посіб. для студентів спеціальності 073 «Менеджмент» освітньо-професійної програми «Менеджмент і бізнес-адміністрування» / КПІ ім. Ігоря Сікорського ; укладачі: Т.В. Лазоренко, С.О. Пермінова.– Електронні текстові дані (1 файл: 560 КБ). Київ : КПІ ім. Ігоря Сікорського. 2021.166 с.
3. Що таке ERP-система управління підприємством та де її краще розмістити? URL: https://www.sim-networks.com/ukr/blog/enterprise-resource-planning-systems-and-cloud-infrastructure (дата звернення: 25.03.2025).
4. Excel чи Google Таблиці? У чому різниця та яку програму вибрати для роботи. URL: https://www.rbc.ua/rus/styler/excel-chi-google-tablitsi-chomu-riznitsya-1719579413.html (дата звернення: 25.03.2025).
5. Для чого потрібні UML діаграми? URL: https://foxminded.ua/uml-diagramy/ (дата звернення: 25.03.2025).
6. Діаграма послідовності (Sequence Diagrams). URL: https://www.maxzosim.com/sequence-diagrams/ (дата звернення: 26.03.2025).
7. Елементи UML. URL: https://docs.kde.org/trunk5/uk/umbrello/umbrello/uml-elements.html (дата звернення: 26.03.2025).
8. Що таке діаграма розгортання? URL: https://www.guru99.com/uk/deployment-diagram-uml-example.html (дата звернення: 26.03.2025).
9. Логічна модель даних. URL: https://data-life-ua.com/db/lohichna-model-danykh/ (дата звернення: 26.03.2025).
10. Робота з MySQL, MS SQL Server та Oracle у прикладах. Практичний посібник для випробувачів. Куликов С., Куликов Святослав.

11. MySQL: визначення, основні характеристики та компоненти. URL: https://simplentrec.com/mysql-introduction/ (дата звернення: 26.03.2025).
12. Коноваленко І.В. Платформа .NET та мова програмування C# 8.0: навчальний посібник / Коноваленко І.В., Марущак П.О. – Тернопіль: ФОП Паляниця В. А., 2020 – 320 с.
13. Про середовище програмування C#. URL: https://foxminded.ua/seredovyshche-prohramuvannia-si-sharp/ (дата звернення: 26.03.2025).
14. Про графічний інтерфейс користувача GUI. URL: https://foxminded.ua/gui-tse/ (дата звернення: 26.03.2025).
15. Основи ADO.NET. URL: https://itvdn.com/ua/blog/article/basics-ado (дата звернення: 26.03.2025).
16. Entity Framework Core 7. URL: https://abitap.com/1-1-entity-framework-core-6/ (дата звернення: 26.03.2025).

ДОДАТОК А

База даних

Сторінок 1

Київ – 2025

```
CREATE DATABASE rotapro;
USE rotapro;

CREATE TABLE Employees (
   EmployeeId INT PRIMARY KEY AUTO_INCREMENT,
   FirstName VARCHAR(50) NOT NULL,
   LastName VARCHAR(50) NOT NULL,
   Position VARCHAR(50),
   Phone VARCHAR(15)
);

CREATE TABLE Shifts (
   ShiftId INT PRIMARY KEY AUTO_INCREMENT,
   EmployeeId INT NOT NULL,
   ShiftDate DATE NOT NULL,
   Hours INT NOT NULL,
   FOREIGN KEY (EmployeeId) REFERENCES Employees(EmployeeId) ON
DELETE CASCADE
);

INSERT INTO Employees (FirstName, LastName, Position, Phone)
VALUES
   ('Олег', 'Коваленко', 'Менеджер', '0987654321'),
   ('Марія', 'Іваненко', 'Адміністратор', '0661234567'),
   ('Павло', 'Сергієнко', 'Технік', '0939876543'),
   ('Анна', 'Мельник', 'Координатор', '0505554433');

INSERT INTO Shifts (EmployeeId, ShiftDate, Hours)
VALUES
   (1, '2025-03-20', 8),  -- Олег, 20 березня, 8 годин
   (1, '2025-03-21', 6),  -- Олег, 21 березня, 6 годин
   (2, '2025-03-20', 7),  -- Марія, 20 березня, 7 годин
   (2, '2025-03-22', 8),  -- Марія, 22 березня, 8 годин
   (3, '2025-03-21', 8),  -- Павло, 21 березня, 8 годин
   (3, '2025-03-23', 6),  -- Павло, 23 березня, 6 годин
   (4, '2025-03-20', 6),  -- Анна, 20 березня, 6 годин
   (4, '2025-03-22', 7);  -- Анна, 22 березня, 7 годин
```

ДОДАТОК Б

Код програми

Сторінок 15

Київ – 2025


```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace RotaPro
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
   }
}

namespace RotaPro
{
   partial class Form1
   {
      /// <summary>
      /// Обязательная переменная конструктора.
      /// </summary>
      private System.ComponentModel.IContainer components = null;

      /// <summary>
      /// Освободить все используемые ресурсы.
      /// </summary>
      /// <param name="disposing">истинно, если управляемый ресурс должен быть
удален; иначе ложно.</param>
      protected override void Dispose(bool disposing)
      {
         if (disposing && (components != null))
         {
```

```
            components.Dispose();
        }
        base.Dispose(disposing);
    }
```



```
    #region Код, автоматически созданный конструктором форм Windows

    /// <summary>
    /// Требуемый метод для поддержки конструктора — не изменяйте
    /// содержимое этого метода с помощью редактора кода.
    /// </summary>
    private void InitializeComponent()
    {
        this.components = new System.ComponentModel.Container();
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.ClientSize = new System.Drawing.Size(800, 450);
        this.Text = "Form1";
    }

    #endregion
  }
}
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MySql.Data.MySqlClient;
using System.Configuration;
using System.Windows.Forms.DataVisualization;
using System.Windows.Forms.DataVisualization.Charting;

namespace RotaPro
{
  public partial class MainForm : Form
  {

    private string connectionString =
ConfigurationManager.ConnectionStrings["rotapro"].ConnectionString;
```

```
        private MySqlConnection connection; // Підключення як поле класу

        private MySqlDataAdapter employeesAdapter;
        private DataTable employeesTable;
```



```
        private DataTable comboBoxEmployeesTable;

        private MySqlDataAdapter shiftsAdapter;
        private DataTable shiftsTable;

        private BindingSource employeesBindingSource = new BindingSource();
        private BindingSource comboBoxBindingSource = new BindingSource();

        public MainForm()
        {
            InitializeComponent();

            connection = new MySqlConnection(connectionString);
            connection.Open();

            dataGridView1.Rows.Clear();
            dataGridView1.Columns.Clear();

            LoadEmployeesToGrid();
            LoadShiftsToGrid();

            LoadEmployeesToComboBox();

            InitializeChart();

            LoadEmployeesToStatsComboBox();
        }

        private void InitializeChart()
        {
            chart1.Series.Clear();
            chart1.Series.Add("Shifts");
            chart1.Series["Shifts"].ChartType = SeriesChartType.Column; // Тип
гістограми
            chart1.ChartAreas[0].AxisX.Title = "Співробітники";
            chart1.ChartAreas[0].AxisY.Title = "Кількість змін";
        }
```

```
        private void LoadEmployeesToStatsComboBox()
        {
            string query = "SELECT EmployeeId, CONCAT(FirstName, ' ', LastName) AS
FullName FROM Employees";
            using (var adapter = new MySqlDataAdapter(query, connection))
            {
```



```
var statsEmployeesTable = new DataTable();
                adapter.Fill(statsEmployeesTable);
                // Додаємо опцію "Всі" для відображення всіх працівників
                DataRow allRow = statsEmployeesTable.NewRow();
                allRow["EmployeeId"] = 0; // 0 означає "Всі"
                allRow["FullName"] = "Всі";
                statsEmployeesTable.Rows.InsertAt(allRow, 0);

                comboBox2.DataSource = statsEmployeesTable;
                comboBox2.DisplayMember = "FullName";
                comboBox2.ValueMember = "EmployeeId";
                comboBox2.SelectedIndex = 0; // За замовчуванням "Всі"
            }
        }


        private void LoadEmployeesToGrid()
        {
            string query = @"
        SELECT EmployeeId, FirstName, LastName, Position, Phone
        FROM Employees";

            employeesAdapter = new MySqlDataAdapter(query, connection);
            employeesTable = new System.Data.DataTable();
            employeesAdapter.Fill(employeesTable);

            employeesBindingSource.DataSource = employeesTable;
            dataGridView1.DataSource = employeesBindingSource;

            dataGridView1.Columns["EmployeeId"].HeaderText = "ID співробітника";
            dataGridView1.Columns["FirstName"].HeaderText = "Ім’я";
            dataGridView1.Columns["LastName"].HeaderText = "Прізвище";
            dataGridView1.Columns["Position"].HeaderText = "Посада";
            dataGridView1.Columns["Phone"].HeaderText = "Телефон";

            foreach (DataGridViewColumn column in dataGridView1.Columns)
```

```
        {
            column.AutoSizeMode = DataGridViewAutoSizeColumnMode.Fill;
        }
    }

    private void LoadShiftsToGrid()
    {
```



```
 dataGridView2.Rows.Clear();
        dataGridView2.Columns.Clear();

        //using (var connection = new MySqlConnection(connectionString))
        //{
        //    connection.Open();
            // SQL-запит із JOIN для відображення імені працівника
            string query = @"
                SELECT s.ShiftId, CONCAT(e.FirstName, ' ', e.LastName) AS
EmployeeName,
                       s.ShiftDate, s.Hours
                FROM Shifts s
                JOIN Employees e ON s.EmployeeId = e.EmployeeId";

            shiftsAdapter = new MySqlDataAdapter(query, connection);
            shiftsTable = new System.Data.DataTable();
            shiftsAdapter.Fill(shiftsTable);

            // Прив’язка DataTable до DataGridView
            dataGridView2.DataSource = shiftsTable;
   // Налаштування колонок (опціонально)
            dataGridView2.Columns["ShiftId"].HeaderText = "ID зміни";
            dataGridView2.Columns["EmployeeName"].HeaderText = "Співробітник";
            dataGridView2.Columns["ShiftDate"].HeaderText = "Дата";
            dataGridView2.Columns["Hours"].HeaderText = "Години";
        //}

        foreach (DataGridViewColumn column in dataGridView2.Columns)
        {
            column.AutoSizeMode = DataGridViewAutoSizeColumnMode.Fill;
        }
    }

    private void LoadEmployeesToComboBox()
    {
```

```
        string query = "SELECT EmployeeId, CONCAT(FirstName, ' ', LastName) AS
FullName FROM Employees";
        using (var adapter = new MySqlDataAdapter(query, connection))
        {
            comboBoxEmployeesTable = new DataTable();
            adapter.Fill(comboBoxEmployeesTable);
        }
        comboBoxBindingSource.DataSource = comboBoxEmployeesTable;
```



```
  comboBox1.DataSource = comboBoxBindingSource;
        comboBox1.DisplayMember = "FullName";
        comboBox1.ValueMember = "EmployeeId";
        comboBox1.SelectedIndex = -1;
    }

    private void dataGridView1_CellClick(object sender,
DataGridViewCellEventArgs e)
    {
        if (e.RowIndex >= 0)
        {
            dataGridView1.Rows[e.RowIndex].Selected = true;

            // Отримуємо вибраний рядок
            DataGridViewRow row = dataGridView1.Rows[e.RowIndex];

            // Переносимо дані з рядка в текстові поля
            textBox2.Text = row.Cells["LastName"].Value.ToString();
            textBox1.Text = row.Cells["FirstName"].Value.ToString();
            textBox3.Text = row.Cells["Position"].Value?.ToString() ?? ""; // Обробка
NULL
            textBox4.Text = row.Cells["Phone"].Value?.ToString() ?? "";      // Обробка
NULL
        }
    }


    private void button1_Click(object sender, EventArgs e)
    {
        if (string.IsNullOrWhiteSpace(textBox1.Text) ||
string.IsNullOrWhiteSpace(textBox2.Text))
        {
            MessageBox.Show("Ім’я та прізвище є обов’язковими полями!",
"Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
            return;
        }

        //using (var connection = new MySqlConnection(connectionString))
        //{
            //connection.Open();
            string query = "INSERT INTO Employees (FirstName, LastName, Position,
Phone) " +
                           "VALUES (@FirstName, @LastName, @Position, @Phone)";
```



```
using (var cmd = new MySqlCommand(query, connection))
            {
                cmd.Parameters.AddWithValue("@FirstName", textBox1.Text);
                cmd.Parameters.AddWithValue("@LastName", textBox2.Text);
                cmd.Parameters.AddWithValue("@Position",
string.IsNullOrWhiteSpace(textBox3.Text) ? (object)DBNull.Value : textBox3.Text);
                cmd.Parameters.AddWithValue("@Phone",
string.IsNullOrWhiteSpace(textBox4.Text) ? (object)DBNull.Value : textBox4.Text);
                cmd.ExecuteNonQuery();
            }
        //}
        RefreshEmployeesGrid();
        RefreshComboBoxEmployees();
        ClearEmployeesFields();
        MessageBox.Show("Співробітника додано!", "Успіх",
MessageBoxButtons.OK, MessageBoxIcon.Information);
    }

    private void button2_Click(object sender, EventArgs e)
    {
        if (dataGridView1.SelectedRows.Count == 0)
        {
            MessageBox.Show("Виберіть співробітника для редагування!",
"Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }
        if (string.IsNullOrWhiteSpace(textBox1.Text) ||
string.IsNullOrWhiteSpace(textBox2.Text))
        {
            MessageBox.Show("Ім’я та прізвище є обов’язковими полями!",
"Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }
```

```
        int employeeId =
Convert.ToInt32(dataGridView1.SelectedRows[0].Cells["EmployeeId"].Value);
        //using (var connection = new MySqlConnection(connectionString))
        //{
            //connection.Open();
            string query = "UPDATE Employees SET FirstName = @FirstName,
LastName = @LastName, " +
                           "Position = @Position, Phone = @Phone WHERE EmployeeId =
@EmployeeId";
```



```
using (var cmd = new MySqlCommand(query, connection))
            {
                cmd.Parameters.AddWithValue("@FirstName", textBox1.Text);
                cmd.Parameters.AddWithValue("@LastName", textBox2.Text);
                cmd.Parameters.AddWithValue("@Position",
string.IsNullOrWhiteSpace(textBox3.Text) ? (object)DBNull.Value : textBox3.Text);
                cmd.Parameters.AddWithValue("@Phone",
string.IsNullOrWhiteSpace(textBox4.Text) ? (object)DBNull.Value : textBox4.Text);
                cmd.Parameters.AddWithValue("@EmployeeId", employeeId);
                cmd.ExecuteNonQuery();
            }
        //}
        RefreshEmployeesGrid();
        RefreshComboBoxEmployees();
        ClearEmployeesFields();
        MessageBox.Show("Дані співробітника оновлено!", "Успіх",
MessageBoxButtons.OK, MessageBoxIcon.Information);
    }

    private void button3_Click(object sender, EventArgs e)
    {
        if (dataGridView1.SelectedRows.Count == 0)
        {
            MessageBox.Show("Виберіть співробітника для видалення!", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }

        var result = MessageBox.Show("Ви впевнені, що хочете видалити цього
співробітника?", "Підтвердження",
                                     MessageBoxButtons.YesNo, MessageBoxIcon.Warning);
        if (result == DialogResult.Yes)
```

```
        {
            int employeeId =
Convert.ToInt32(dataGridView1.SelectedRows[0].Cells["EmployeeId"].Value);
            //using (var connection = new MySqlConnection(connectionString))
            //{
                //connection.Open();
                string query = "DELETE FROM Employees WHERE EmployeeId =
@EmployeeId";
                using (var cmd = new MySqlCommand(query, connection))
                {
                    cmd.Parameters.AddWithValue("@EmployeeId", employeeId);
```



```
cmd.ExecuteNonQuery();
                }
            //}
            RefreshEmployeesGrid();
            RefreshShiftsGrid();
            RefreshComboBoxEmployees();
            ClearEmployeesFields();
            MessageBox.Show("Співробітника видалено!", "Успіх",
MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }

    private void button4_Click(object sender, EventArgs e)
    {
        ClearEmployeesFields();
    }

    private void ClearEmployeesFields()
    {
        textBox1.Text = "";
        textBox2.Text = "";
        textBox3.Text = "";
        textBox4.Text = "";
        dataGridView1.ClearSelection(); // Знімаємо виділення з рядка
    }

    private void ClearShiftFields()
    {
        comboBox1.SelectedIndex = -1;
        dateTimePicker1.Value = DateTime.Today;
        numericUpDown1.Value = 0;
```

```
            dataGridView2.ClearSelection();
        }

        private void RefreshEmployeesGrid()
        {
            employeesTable.Clear();
            employeesAdapter.Fill(employeesTable);
            employeesBindingSource.ResetBindings(false);
        }
        private void RefreshShiftsGrid()
        {
            shiftsTable.Clear();
```



```
shiftsAdapter.Fill(shiftsTable);
            UpdateCalendar();
        }

        private void RefreshComboBoxEmployees()
        {
            comboBoxEmployeesTable.Clear();
            using (var adapter = new MySqlDataAdapter("SELECT EmployeeId,
CONCAT(FirstName, ' ', LastName) AS FullName FROM Employees", connection))
            {
                adapter.Fill(comboBoxEmployeesTable);
            }
            comboBoxBindingSource.ResetBindings(false);
        }


        private void button8_Click(object sender, EventArgs e)
        {
            if (comboBox1.SelectedIndex == -1)
            {
                MessageBox.Show("Виберіть співробітника!", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            if (numericUpDown1.Value <= 0)
            {
                MessageBox.Show("Кількість годин має бути більше 0!", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
```

```
        string query = "INSERT INTO Shifts (EmployeeId, ShiftDate, Hours) " +
                       "VALUES (@EmployeeId, @ShiftDate, @Hours)";
        using (var cmd = new MySqlCommand(query, connection))
        {
            cmd.Parameters.AddWithValue("@EmployeeId",
comboBox1.SelectedValue);
            cmd.Parameters.AddWithValue("@ShiftDate", dateTimePicker1.Value.Date);
            cmd.Parameters.AddWithValue("@Hours", numericUpDown1.Value);
            cmd.ExecuteNonQuery();
        }
        RefreshShiftsGrid();
        ClearShiftFields();
```



```
MessageBox.Show("Зміну додано!", "Успіх", MessageBoxButtons.OK,
MessageBoxIcon.Information);
    }

    private void button7_Click(object sender, EventArgs e)
    {
        if (dataGridView2.SelectedRows.Count == 0)
        {
            MessageBox.Show("Виберіть зміну для редагування!", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }
        if (comboBox1.SelectedIndex == -1)
        {
            MessageBox.Show("Виберіть співробітника!", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }
        if (numericUpDown1.Value <= 0)
        {
            MessageBox.Show("Кількість годин має бути більше 0!", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }

        int shiftId =
Convert.ToInt32(dataGridView2.SelectedRows[0].Cells["ShiftId"].Value);
        string query = "UPDATE Shifts SET EmployeeId = @EmployeeId, ShiftDate =
@ShiftDate, Hours = @Hours " +
```

```
                    "WHERE ShiftId = @ShiftId";
        using (var cmd = new MySqlCommand(query, connection))
        {
            cmd.Parameters.AddWithValue("@EmployeeId",
comboBox1.SelectedValue);
            cmd.Parameters.AddWithValue("@ShiftDate", dateTimePicker1.Value.Date);
            cmd.Parameters.AddWithValue("@Hours", numericUpDown1.Value);
            cmd.Parameters.AddWithValue("@ShiftId", shiftId);
            cmd.ExecuteNonQuery();
        }
        RefreshShiftsGrid();
        ClearShiftFields();
        MessageBox.Show("Зміну оновлено!", "Успіх", MessageBoxButtons.OK,
MessageBoxIcon.Information);
```



```
}

    private void button6_Click(object sender, EventArgs e)
    {
        if (dataGridView2.SelectedRows.Count == 0)
        {
            MessageBox.Show("Виберіть зміну для видалення!", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }

        var result = MessageBox.Show("Ви впевнені, що хочете видалити цю
зміну?", "Підтвердження",
                                     MessageBoxButtons.YesNo, MessageBoxIcon.Warning);
        if (result == DialogResult.Yes)
        {
            int shiftId =
Convert.ToInt32(dataGridView2.SelectedRows[0].Cells["ShiftId"].Value);
            string query = "DELETE FROM Shifts WHERE ShiftId = @ShiftId";
            using (var cmd = new MySqlCommand(query, connection))
            {
                cmd.Parameters.AddWithValue("@ShiftId", shiftId);
                cmd.ExecuteNonQuery();
            }
            RefreshShiftsGrid();
            ClearShiftFields();
            MessageBox.Show("Зміну видалено!", "Успіх", MessageBoxButtons.OK,
MessageBoxIcon.Information);
```

```
        }
    }

    private void button5_Click(object sender, EventArgs e)
    {
        ClearShiftFields();
    }

    private void dataGridView2_CellClick(object sender,
DataGridViewCellEventArgs e)
    {
        if (e.RowIndex >= 0)
        {
            dataGridView2.Rows[e.RowIndex].Selected = true;
            DataGridViewRow row = dataGridView2.Rows[e.RowIndex];
```



```
string employeeName = row.Cells["EmployeeName"].Value.ToString();
            DataRow employeeRow = comboBoxEmployeesTable.AsEnumerable()
                .FirstOrDefault(r => r.Field<string>("FullName") == employeeName);
            if (employeeRow != null)
            {
                comboBox1.SelectedValue = employeeRow.Field<int>("EmployeeId");
            }

            dateTimePicker1.Value =
Convert.ToDateTime(row.Cells["ShiftDate"].Value);
            numericUpDown1.Value = Convert.ToInt32(row.Cells["Hours"].Value);
        }
    }

    private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
    {
        UpdateCalendar();
    }

    private void UpdateCalendar()
    {
        // Очищаємо попередні підсвічені дати
        monthCalendar1.RemoveAllBoldedDates();

        if (comboBox1.SelectedIndex != -1)
        {
```

```
            try
            {
                int selectedEmployeeId = Convert.ToInt32(comboBox1.SelectedValue);
                string query = "SELECT ShiftDate FROM Shifts WHERE EmployeeId =
@EmployeeId";

                using (var cmd = new MySqlCommand(query, connection))
                {
                    cmd.Parameters.AddWithValue("@EmployeeId", selectedEmployeeId);
                    using (var reader = cmd.ExecuteReader())
                    {
                        while (reader.Read())
                        {
                            DateTime shiftDate = reader.GetDateTime("ShiftDate");
                            monthCalendar1.AddBoldedDate(shiftDate); // Підсвічуємо дату
                        }
```

Сторінка 14

```
}
                }
            }
            catch { }
        }
        monthCalendar1.UpdateBoldedDates(); // Оновлюємо відображення
    }

    private void button12_Click(object sender, EventArgs e)
    {
        chart1.Series["Shifts"].Points.Clear();

        string query = @"
            SELECT e.EmployeeId, CONCAT(e.FirstName, ' ', e.LastName) AS
EmployeeName,
                   COUNT(s.ShiftId) AS ShiftCount
            FROM Employees e
            LEFT JOIN Shifts s ON e.EmployeeId = s.EmployeeId
                AND s.ShiftDate BETWEEN @StartDate AND @EndDate
            GROUP BY e.EmployeeId, e.FirstName, e.LastName";

        // Якщо вибрано конкретного працівника
        int selectedEmployeeId = Convert.ToInt32(comboBox2.SelectedValue);
        if (selectedEmployeeId != 0) // 0 означає "Всі"
        {
            query += " HAVING e.EmployeeId = @EmployeeId";
```

```
        }

        using (var cmd = new MySqlCommand(query, connection))
        {
            cmd.Parameters.AddWithValue("@StartDate", dateTimePicker3.Value.Date);
            cmd.Parameters.AddWithValue("@EndDate", dateTimePicker2.Value.Date);
            if (selectedEmployeeId != 0)
            {
                cmd.Parameters.AddWithValue("@EmployeeId", selectedEmployeeId);
            }

            using (var reader = cmd.ExecuteReader())
            {
                while (reader.Read())
                {
                    string employeeName = reader.GetString("EmployeeName");
                    int shiftCount = reader.GetInt32("ShiftCount");
```



```
chart1.Series["Shifts"].Points.AddXY(employeeName, shiftCount);
                }
            }
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace RotaPro
{
    internal static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
```

```
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new MainForm());
        }
    }
}
```