МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Литвиненко О.Є.

«___» _____2021 р.

ДИПЛОМНИЙ ПРОЄКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬО-КВАЛІФІКАЦІЙНОГО РІВНЯ "БАКАЛАВР"

Тема: «*CRM*-система «Електронний журнал» викладача навчальних курсів на базі платформи *WPF*»

Виконавець: Бігняк О.Г.

Керівник: Кучерява О.М.

Нормоконтролер: Тупота Є.В.

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет Кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління

Освітньо-кваліфікаційний рівень бакалавр
Спеціальність 123 "Системне програмування"
(шифр, найменування)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Литвиненко О.Є.
«_____» _____2021 р.

ЗАВДАННЯ
на виконання дипломного проєкту
Бігняка Олександра Григоровича
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема проєкту (роботи): «*CRM*–система «Електронний журнал» викладача навчальних курсів на базі платформи *WPF*»
затверджена наказом ректора від "04" лютого 2021 року №135/ст.

2. Термін виконання проєкту (роботи): з 17.05.2021 до 20.06.2021

3. Вихідні дані до проєкту (роботи): Використання мови *C#* на базі платформи *WPF* для реалізації програмного забезпечення віддаленої системи обліку на основі СУБД *MySQL*

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):
1) аналіз засобів розробки сучасних додатків;
2) проєктування бази даних програмної системи;
3) розробка програмного забезпечення електронного журналу викладача.

5. Перелік обов'язкового графічного матеріалу:
1) діаграма бази даних ;
2) дизайн додатку;
3) схема взаємодії екранів додатку.

6. Календарний план

№ п/п	Етапи виконання дипломного проєкту	Термін виконання етапів	Примітка
1	Ознайомлення з постановкою задачідипломного проєктування	17.05.2021-18.05.2021	
2	Вивчення спеціальної літератури і технічної документації	18.05.2021-19.05.2021	
3	Розробка та налагодження програми	19.05.2021-30.05.2021	
4	Написання пояснювальної записки	31.05.2021-10.06.2021	
5	Підготовка графічного та демонстраційного матеріалів	11.06.2021-13.06.2021	
6	Захист дипломного проєкту	14.06.2021-20.06.2021	

7. Дата видачі завдання «17» травня 2021 р.

Керівник дипломного проєкту _____ Кучерява О.М.
(підпис)

Завдання прийняв до виконання _____ Бігняк О.Г.
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проєкту «*CRM*-система «Електронний журнал» викладача навчальних курсів на базі платформи *WPF*»: 40 с., 10 рис., 15 використаних джерел, 1 додаток.

КРОСПЛАТФОРМНІСТЬ, ЕЛЕКТРОННИЙ ЖУРНАЛ, ФРЕЙМВОРК.

Мета дослідження – розробка програмної системи електронного журналу.

Об’єкт дослідження – процес ведення звітності про відвідування та оплату навчальних курсів.

Предмет дослідження – програмна система електронного журналу.

Результатом виконання дипломного проєкту є розроблена програмна система «Електронний журнал», яка забезпечує швидкий і зручний доступ до даних про учнів, що відвідують навчальні курси, без використання паперових аналогів. Також дозволяє працювати з базами даних у віддаленому режимі роботи та оперативно оновлювати дані.

Розроблена система прийнята для практичного використання в діяльності ІТ академії робототехніки *«RoboLab»,* зокрема ведення обліку відвідування та оплати навчальних занять.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

WPF – *Windows presentation foundation* (фундамент презентації вікон)

GUI – *Graphical User Interface* (графічний інтерфейс користувача)

CRM – *Customer Relationship Management* (управління взаємовідносинами з клієнтами)

XAML – *eXtensible Application Markup Language* (розширювана мова арозмітки для додатків)

SQL – *Structured Query Language* (структурована мова запитів)

БД – база даних

СУБД – система управління базами даних

ВСТУП

Актуальність теми. Сучасне інформаційне суспільство спонукає людину до спрощення його життя, в чому йому допомагає діджиталізація – оцифровування різних видів інформації. Діджиталізація дозволяє обробляти величезні обсяги інформації, тим самим спрощуючи і прискорюючи роботу. Прикладів впровадження діджиталізація різні сфери суспільного життя безліч. Наприклад, сучасна освіта відчуває незворотні зміни, що пов'язано головним чином з розвитком і поширенням інформаційних технологій. В даний час швидкість обміну інформації та обсяги даних сприяють розвитку інформаційно–комунікаційної інфраструктури як всередині освітнього закладу, так і за його межами. Уже сьогодні сучасна школа повинна позиціонувати себе як цілісний організм у відкритому інформаційному просторі мережі Інтернет, створюючи тим самим сприятливі умови для формування позитивних відгуків про свою діяльність і високого рейтингу серед існуючих шкіл. Також електронний шкільний документооборот повинен знизити адміністративне навантаження на освітні заклади.

Мета і завдання дипломного проєкту. Метою дипломного проєкту є розробка системи «Електронний журнал». Для досягнення цілей дипломного проєктування було застосовано навички створення та роботи із базами даних, досвід створення *GUI* для додатків та навички роботи із мовою програмування *C#* . Виходячи з мети проєкту, було визначено такі завдання:

− розробка та опис предметної області програмного засобу;

− проєктування та розробка бази даних в *Microsoft SQL Server* 2019;

− розробка додатку на мові *C#* , розробка *GUI* на базі платформи *WPF* із використанням мови *XAML* ;

− демонстрація можливостей застосування системи.

Розробка програмного засобу передбачає реалізацію таких функцій:

1) Можливість додавання та редагування інформації про учнів, їх відвідування та оплату курсів.

2) Розгортання БД на серверах *Microsoft Azure*.

3) Захист даних: без введення правильних логіна та пароля адміністратора вхід до загальної БД буде блоковано.

Об'єкт і предмет проєктування. Об'єкт дослідження – процес ведення звітності про відвідування та оплату навчальних курсів. Предмет дослідження – програмна система електронного журналу. Оцифрування інформації значно пришвидшить час на адміністративні питання, також зробить комфортніші умови праці, оскільки доступ до інформації буде доступний із будь-якого комп'ютера, на котрому встановлений «електронний журнал», за допомогою мережі Інтернет. У результаті дипломного проєктування, відповідно до поставленого завдання, було розроблено програмний засіб, який дає змогу не тільки редагувати базу даних, а й здійснювати її адміністрування.

Результатом виконання дипломного проєкту є розроблена програмна система «Електронний журнал», яка забезпечує швидкий і зручний доступ до даних про учнів, що відвідують навчальні курси, без використання паперових аналогів. Також дозволяє працювати з базами даних у віддаленому режимі роботи та оперативно оновлювати дані.

Методи проєктування. В якості засобу створення та редагування бази даних використовується СУБД *Microsoft SQL Server Management Studio*, який є частиною *Microsoft SQL Server* 2019. Головним інструментом *SQL Server Management Studio* є *Object Explorer*, який дозволяє користувачеві переглядати, отримувати об'єкти сервера, а також повністю ними керувати.

В свою чергу сама БД розгорнута не сервері *Microsoft Azure*, аби забезпечити онлайн доступ до даних та їх синхронізацію із будь-якого комп'ютера , на якому встановлений «електронний журнал», за допомогою мережі Інтернет.

Оскільки існує необхідність тісної взаємодії програмної частини та бази

даних, потрібно було вибрати найбільш зручні засоби зі створення програмного забезпечення працюючого з цією базою.

Тому було обрано *Microsoft Visual Studio 2019* – середовище розробки, за допомогою якого розробляється програмне забезпечення.

Мовою програмування для реалізації даної програми було обрано *C#*, а платформою для *GUI* виступає *WPF* на мові *XAML*. Тим самим було забезпечено можливість використання у майбутньому розробленого прототипу для будь-яких інших БД.

Практичне значення отриманих результатів. Розроблена в даному дипломному проєкті система «Електронний журнал» дозволяє викладачу, за допомогою мережі Інтернет, переглядати та змінювати інформацію про учнів, їх відвідування уроків та оплати занять. Управління даною системою здійснює адміністратор, який може корегувати права доступу до БД.

Апробація отриманих результатів. Розроблена система прийнята для практичного використання в діяльності ІТ академії робототехніки *«RoboLab»,* зокрема ведення обліку відвідування та оплати навчальних занять.

РОЗДІЛ 1
АНАЛІЗ ЗАСОБІВ РОЗРОБКИ СУЧАСНИХ ДОДАТКІВ

Додаток для персонального комп’ютера – програма, що працює на вашому робочому столі, так званий «декстоп–додаток». Такі додатки зручні для користувачів, зазвичай інтегровані із різними офісними або іншими настільними додатками

Популярність розробки настільних додатків з кожним роком зростає. Збільшується попит на високоефективні та прості у користуванні додатки. Для полегшення роботи програмісти використовують платформи для розробки – фреймворки.

Фреймворк (*framework*) – програмна платформа, яка визначає структуру програмної системи; програмне забезпечення, що полегшує розробку і об'єднання різних компонентів великого програмного проєкту.

Вибір найкращої платформи для розробки таких додатків багато в чому залежить від операційної системи. Тож для реалізації додатку, було розглянуто 5 найпопулярніших платформ для розробки настільних додатків:

– *WPF;*
– універсальна платформа *Windows (UWP);*
– *Cocoa;*
– *Electron;*
– *Swing.*

1.1. Аналіз існуючих платформ програмування для розробки додатків

На даний момент існує велика кількість платформ програмування, які призначені для розробки додатків для ПК. Кожна з них має свої особливості використання, переваги та недоліки.

Список програм, які ви можете створити за допомогою платформи розробки програмного забезпечення для ПК:

- автономний бізнес-додаток;
- клієнт-серверний додаток;
- спільні програми;
- програми та плагіни;
- мультимедійні програми;
- мережеві додатки.

1.1.1. Огляд фреймворку *WPF*

WPF – це одна з найпопулярніших платформ для розробки додатків *Windows. Windows Presentation Foundation* (*WPF*) – це платформа в середовищі *.NET*, яка в основному використовується для розробки графіки додатків для персонального комп'ютера. Ви можете використовувати її для створення призначеного для користувача інтерфейсу для програмного забезпечення. Динамічні бібліотеки *WPF* зазвичай вбудовані в операційну систему *Windows*. Ключовою характеристикою *WPF* є його здатність уніфікувати різні елементи призначеного для користувача інтерфейсу. Ці елементи включають в себе векторну графіку, адаптивні документи, попередньо візуалізовані медіа–об'єкти і рендеринг *2D* і *3D*.

1.1.2. Огляд фреймворку Універсальна платформа *Windows (UWP)*

UWP – ще одна важлива структура для розробки додатків для настільних комп'ютерів. Це також фреймворк, який високо цінується за популярність платформи *.NET*. Тому, що це дозволяє розробникам створювати кросплатформені настільні додатки. По суті, *UWP* дозволяє розробникам створювати додатки, які можуть працювати на різних платформах, що належать *Microsoft*. Це означає, що ваше програмне забезпечення зможе працювати на декількох пристроях. Платформа *UWP* має високу масштабованість. Якщо ви створили додаток для робочого столу, фреймворк дозволить масштабувати його для мобільних пристроїв. Дозволяє програмі без проблем працювати на мобільних пристроях, планшетах і навіть на *Xbox*. Це фреймворк, значно поліпшує функціональність *Visual Studio.*

1.1.3. Огляд фреймворку *Cocoa*

Cocoa є нативним середовищем для розробки власних *MacOs*. Це об'єктно–орієнтована структура для створення призначеного для користувача інтерфейсу для *MacOS, iOS* і *tvOS*. Вона не тільки додає функціональність призначеного для користувача інтерфейсу, але і робить інтерфейс більш цікавим. З платформою *Cocoa* ви можете додати функцію анімації, графічний елемент управління і функції розпізнавання жестів в настільний додаток. Всі інструменти розробки для *Cocoa* надаються *Apple*. Коли справа доходить до написання кодів, фреймворк можна використовувати з мовами *Python, Perl і Ruby*. Щоб використовувати ці мови, вам знадобляться мости, такі як *PyObjC, PasCocoa і RubyCocoa.*

1.1.4. Огляд фреймворку *Electron*

Electron – це кроссплатформенне середовище розробки, розроблене *GitHub*. Фреймворк використовує *Node.js*, і розробники можуть використовувати його для створення кроссплатформенних додатків.

Судячи з усього, досить багато великих компаній використовують цю платформу для розробки своїх додатків: *Facebook, Microsoft* і *Stack*. Фреймворк дозволяє розробникам зосередитися на основних функціях програмного забезпечення. Тому, що фреймворк обробляє найскладніші частини процесу розробки програмного забезпечення.

Плюси :

– використання напрацювань з *Web;*
– просто знайти (або "виховати") фахівця;
– якісна документація;
– підтримка спільноти і *GitHub*.

Мінуси :

– високе споживання пам'яті (фізичної і ОЗУ);
– легко написати поганий код;
– погана нативність.

1.1.5. Огляд фреймворку *Swing*

Swing – це заснована на *Java* інфраструктура, яку ви можете використовувати для розробки додатків. Будучи кросплатформеним фреймворком, додатки, створені *Swing*, можуть працювати на будь–якій платформі. Основною функцією *Swing* є створення кращого графічного інтерфейсу. Він здатний емулювати дизайн, зовнішній вигляд і стиль багатьох настільних додатків.

Фреймворк поставляється з різними компонентами користувальницького інтерфейсу, такими як кнопки, панелі прокрутки таблиць і прапорці. Як бачите, деякі фреймворки призначені для *Windows*, інші – *MacOS*, а деякі – кросплатформені фреймворки розробки.

Саме характер додатку для настільного комп'ютера, яке ви розробляєте, буде визначати середовище розробки.

1.2. Аналіз існуючих мов програмування для розробки додатків для

Програмування зараз використовується майже у будь–якій сфері людської діяльності. Це спонукає до виникнення великої кількості різноманітних мов програмування. Перша десятка найбільш популярних мов програмування покриває більше половини існуючого ринку. Велика кількість мов програмування розрахована на вузьку нішу, також існують експериментальні мови програмування, що не розраховані на масову авдиторію.

Для реалізації додатку, було розглянуто 7 найпопулярніших мов програмування для розробки настільних додатків:

- *JavaScript;*
- *Python;*
- *Java;*
- *C/C++;*
- *PHP;*
- *C#;*
- *SQL.*

1.2.1. Огляд мови програмування *JavaScript*

JavaScript – скриптова мова програмування. Він підтримує як об'єктно–орієнтований, так і функціональний спосіб програмування. *JavaScript* активно використовується для створення інтерактивних *web*–сторінок. Іншими словами, все те, що Ви бачите практично на всіх сайтах в інтернеті (*Front–end*), тобто клієнтська частина, що виконується на стороні користувача в браузері, реалізована саме на *JavaScript.* За версією рейтингу *GitHub і Stack Overflow* мову *JavaScript* є найпопулярнішим, за іншими показниками і індексам він також впевнено входить в десятку популярних мов програмування. А якщо подивитися на кількість вакансій на *HeadHunter,* в яких згадується *JavaScript*, то ніяких сумнівів не виникне, що *JavaScript* є найпопулярнішим і затребуваним мовою програмування.

1.2.2. Огляд мови програмування *Python*

Python – це універсальна скриптова мова програмування. Вона підходить для реалізації різноманітних завдань, і застосовується практично у всіх напрямках, починаючи від веб-розробки, і закінчуючи створенням десктопних і мобільних додатків, включаючи ігри, і, звичайно ж, для аналітики даних. *Python* орієнтований на підвищення продуктивності та читання коду. *Python* підтримує структурне, об'єктно–орієнтоване, аспектно-орієнтоване, функціональне і імперативне програмування. За версією рейтингу *PYPL і IEEE Spectrum Python* є найпопулярнішою мовою програмування, по іншим версіями він також входить в число лідерів.

1.2.3. Огляд мови програмування *Java*

Java – це строго типізована об'єктно–орієнтована мова програмування. Код *Java* зазвичай транслюється в спеціальний байт–код, і може виконуватися на будь–якій комп'ютернії архітектурі за допомогою віртуальної *Java*–машини. Сфер застосування *Java* дуже багато: це і веб–розробка, і розробка мобільних додатків, і, звичайно ж, розробка десктопних додатків. За версією індексу *TIOBE* мову програмування *Java* вже досить давно займає лідируюче місце. В інших рейтингах *Java* не менш популярний, так, наприклад, в рейтингу *IEEE Spectrum і PYPL Java* знаходиться на другому місці.

1.2.4. Огляд мов програмування *C / C ++*

C / C ++ – це компільовані мови програмування. Вони широко використовуються для розробки операційних систем, різних прикладних програм, драйверів, а також ігор.

Особливістю цих мов є те, що на них можна писати високопродуктивний код, таким чином, ці мови відмінно підходять для систем, від яких потрібно отримати максимум продуктивності.

Ці мови вже давно є одними з найбільш популярних і затребуваних, тому жоден рейтинг не обходиться без згадки цих мов.

1.2.5. Огляд мови програмування *PHP*

PHP – це скриптова мова програмування. Вона активно використовується для розробки веб–додатків. Переважна більшість сайтів в інтернеті, в частині функціональності (*Back–end*), реалізовано саме на *PHP*.

Всім відомі *CMS WordPress* і *Joomla* розроблені на мові *PHP*. Тому в популярності цієї мови сумніватися не доводиться. Більшість програмістів починають свій шлях саме з *PHP*, так як на фрілансі пропозицій роботи з мовою *PHP* величезна кількість.

1.2.6. Огляд мови програмування *C #*

C# – це об'єктно–орієнтована мова програмування, розроблена компанією *Microsoft*. *C #* відноситься до сімейства *C*–подібних мов. Для роботи з мовою *C #* потрібно платформа *.NET Framework. C #* застосовують для розробки додатків для ПК, створення веб–сервісів, а також мобільних додатків. У кожному з рейтингів *C #* впевнено входить в ТОП 10 найпопулярніших і затребуваних мов програмування.

1.2.7. Огляд мови програмування *SQL*

SQL (Structured Query Language) – це мова структурованих запитів, за допомогою неї пишуться спеціальні запити до бази даних. *SQL* – декларативна мова програмування. За допомогою саме мови *SQL* ми здійснюємо всі дії в базі даних, зокрема, створюємо дані, змінюємо їх, видаляємо і витягаємо ці дані, іншими словами

SQL – це мова для взаємодії користувача з базою даних. Мова *SQL* не займає лідируючі місця в авторитетних рейтингах, хоча в деяких рейтингах він входить в ТОП 10, а за версією *Stack Overflow* навіть займає 3 рядок. Однак в сучасному світі із зростанням обсягу інформації та даних, мова *SQL* набуває все більшої цінності. Так, практично у будь–якій вакансії, пов'язаної з розробкою, в більшості випадків Ви зустрінете вимоги про знання *SQL*. Що і робить цю мову дуже популярною і затребуваною.

1.3. Висновки до розділу

У даному розділі було проведено аналіз предметної області та поставлені задачі проєктування.

Порівнюючи переваги та недоліки існуючих мов програмування і фреймворків для розробки додатків для персональних комп’ютерів, було виявлено, що оптимальним рішенням для даного проєкту є вибір мови програмування *C#* у поєднанні із фреймворком *WPF*. Цільова ОС для використання є саме *Windows,* оскільки вона є найбільш поширеною серед працівників сфери освіти, що тільки закріпило вибір фреймворку *WPF*. Також важливим фактором вибору являється приємний візуальний стиль *GUI WPF* , бо одним із найважливіших завдань було створити максимально зручний та інтуїтивно зрозумілий інтерфейс для користувача.

РОЗДІЛ 2
РОЗРОБКА БАЗИ ДАНИХ ДОДАТКУ

2.1. Опис баз даних, що використовуються при розробці системи

Етапи процесу розробки додатків баз даних:

1) Визначення цілей і завдань програми;

2) Проєктування структури бази даних і прикладних процесів, необхідних для реалізації цих завдань;

3) Використання проєкту в додатку шляхом побудови необхідних об'єктів баз даних та об'єктів програми;

4) Тестування додатків на відповідність поставленим цілям;

5) Встановлення програм для експлуатації користувачем.

Перший крок у проєктуванні будь–якої програми – чітке визначення призначення і конкретних завдань програми. Другим кроком є розподіл завдань на операційні процеси.

Процес проєктування додатків тісно пов'язаний з процесом проєктування бази даних.

При побудові бази даних перш за все треба визначити основні операційні процеси і таблиці для обслуговування цих процесів, а також які поля вони повинні будуть містити. Цей процес називається моделюванням даних.

Існує кілька підходів до моделювання даних. Розглянемо два з них: моделювання центрального додатку і видове моделювання.

Моделювання центрального додатку відповідає потребам додатку в даних завдяки одній базі даних. Кожна програма має свою базу даних або свій набір таблиць.

У свою чергу, видове моделювання організовує таблиці в базах даних відповідно до видів даних, які ці таблиці представляють.

Розглянемо приклад. Таблиця, має список продукції, що випускається компанією, може перебувати в базі даних, пов'язаної з торговельною діяльністю, в той час як опис комплектуючих для виготовлення продукції, службовців, знаходиться у виробничій базі даних. Звіт, який відображає сумарний дохід по кожній одиниці продукції, повинен бути побудований на основі даних з обох таблиць, незважаючи на те, що вони розміщуються в абсолютно різних базах даних.

Ключовий елемент в конструюванні баз даних – таблиця. Кожну таблицю можна представити у вигляді площини, розкреслену на рядки і стовпці. Як і в електронній таблиці, рядки представляють записи в таблиці, а стовпчики – поля.

Запис – це головний зміст таблиці, вона зв'язується з реально діючим об'єктом даних. Це може бути номер рахунку банку або прізвище клієнта.

Поле – це елемент всередині записи, воно являє характеристику об'єкта, представленого запису. Значення даного поля для кожного запису називається атрибутом.

Поле або набір полів, який відрізняє один запис в таблиці від іншого, називається ідентифікатором елемента. Він унікальним чином визначає кожний запис у таблиці. Прикладом ідентифікатора елемента може служити поле номера рахунку в таблиці рахунків.

Первинний ключ – це поле або поля таблиці, які використовують як ідентифікатор елемента. Подібно ідентифікатору, значення первинного ключа таблиці завжди унікальне для кожного запису.

Поля, включаючи первинний ключ, використовуються для побудови індексу, призначеного для швидкого доступу до рядків таблиці.

Зовнішній ключ – це поле або поля таблиці, які, не будучи використаними в якості ідентифікатора, часто використовуються при об'єднанні з іншими таблицями.

Об'єднання – це логічні відносини однієї таблиці в іншу на основі загального ключа. Об'єднання повертають результуючий набір, який містить елементи з обох таблиць.

При проєктуванні бази даних, яка покликана обслуговувати намічені операційні процеси, необхідно подумати про елементи, які доведеться запам'ятовувати цих таблиць. За допомогою списку розроблених раніше таблиць треба скласти список інформаційних елементів (полів), з якими додаток буде функціонувати, визначити оптимальний тип даних і розмір для кожного поля, простеживши при цьому, щоб не було дублювання полів.

Розподілити поля з цього списку в структурі таблиць, які були намічені заздалегідь.

Розділити все таблиці на базові і таблиці транзакцій (змін). Базові таблиці містять елементи, які є унікальними за визначенням. Як приклад базової таблиці можна привести таблицю, в якій перераховані столиці районів.

Таблиці транзакцій отримують дані, що збираються від апаратних засобів або вводяться користувачами. Їх зміст часто підлягає змінам.

Дані повинні дублюватися на міжтабличному рівні. Це випливає з того, що первинний ключ однієї таблиці повинен бути продубльований первинним або зовнішнім ключем іншої таблиці з метою встановлення між ними взаємин. Ці взаємини виражаються в двох різновидах: один–до–одного і один–до–багатьох. Взаємовідносини один–до–багатьох утворюють основу для більшості додатків ведення баз даних. Таблиці, орієнтовані на взаємини один–до–багатьох, називаються головними і підлеглими таблицями. Усередині групи транзакційних таблиць визначить підлеглі таблиці, тобто ті, які залежать від інших при поданні значимої інформації, а також вирішити, які таблиці є головними.

Деякі таблиці по відношенню до однієї таблиці грають роль підлеглих, а по відношенню до інших є головними. Головні і підлеглі таблиці зв'язуються комбінацією первинного та зовнішнього ключів. Прикладом взаємин між головною і підлеглою таблицями є пара таблиць, одна з яких функціонує як головна і зберігає

заголовки інформацію для рахунку, а друга використовується як підпорядкована і запам'ятовує повні характеристики для рахунку.

Дуже корисно відобразити взаємини між таблицями графічно. Мета такої діаграми відносин полягає в візуальному представленні зв'язків між таблицями.

Також має сенс побудови схематичного зображення повного проєкту бази даних і побудови діаграм потоків даних для процесів, які утворюють додаток.

2.2. Опис системи управління базою даних

При побудові розподіленої системи було вирішено використовувати СУБД *MySQL*.

MySQL – дуже швидка, надійна система керування базами даних. База даних дозволяє ефективно зберігати, шукати, сортувати і отримувати дані. Сервер *MySQL* управляє доступом до даних, дозволяючи працювати з ними одночасно декільком користувачам, забезпечує швидкий доступ до даних, гарантуючи надання доступу тільки певним користувачам. Вона застосовує *SQL* – стандартна мова запитів бази даних, використовується по всьому світу. *MySQL* з'явився на ринку в 1996 році, але його розробка почалася ще в 1979 році.

На даний момент пакет *MySQL* доступний, як програмне забезпечення з відкритим вихідним кодом, але при необхідності можна отримати і комерційні ліцензії.

До конкурентам *MySQL*, крім інших відносяться *PostgreSQL, Microsoft SQL, Server і Oracle. MySQL* має багато переваг, в тому числі: високою продуктивністю, *MySQL*, безсумнівно, працює дуже швидко.

Багато з цих порівняльних тестів показують, що *MySQL* працює на порядок швидше за своїх конкурентів. Пакет *MySQL* доступний безкоштовно відповідно до ліцензії на програмне забезпечення з відкритим вихідним кодом, або, якщо це необхідно, за невелику суму можна придбати комерційну ліцензію.

У більшості сучасних баз даних використовується *SQL*. Якщо раніше ви працювали з іншими СУБД, перехід до цієї системи не викличе якихось труднощів.

Установка *MySQL* така ж проста, як і у інших аналогічних продуктів. *MySQL* може використовуватися в середовищі багатьох різних систем *UNIX*, а також в середовищі *Microsoft Windows*. Як і в випадку *PHP*, вихідний код *MySQL* можна вивантажувати і змінювати.

У наш час переважна більшість інтерактивних сайтів влаштована за трирівневою архітектурою "клієнт–сервер".

2.3. Структура бази даних

2.3.1. Структура довідників

Розроблена база даних містить ряд довідників, які дозволяють спростити і прискорити роботу з базою даних за рахунок розміщення в них інформації, рідко змінюється.

Для роботи з довідниками розроблені спеціальні форми, які передбачають при описі одного товару введення даних відразу в ряд довідників.

Основні зв'язку в даних таблицях відбуваються через ключові поля. Дані поля для наочності відмічені знаком ключ. Ключові поля введені для індексації таблиць, зменшує час звернення до таблиці, і для можливості використовувати інструмент Підстановки даних з інших таблиць.

Тип поля лічильник відповідає за автоматичне формування значення поля при додаванні нового запису. Значення мають цілий тип і збільшуються послідовно.

2.3.2. Структура бази даних

Загальну структуру бази даних можна представити и вигляді набору пов’язаних між собою таблиць (рис. 2.1)

workpls.workpls - dbo.users

	Column Name	Data Type	Allow Nulls
🔑	login	nvarchar(50)	☐
	password	nchar(10)	☑

а)

workpls.workpls - dbo.monday1400

	Column Name	Data Type	Allow Nulls
	countplus	nchar(10)	☑
🔑	names	nvarchar(50)	☐
	data0305	nchar(10)	☑
	data1005	nchar(10)	☑
	data1205	nchar(10)	☑
	data1405	nchar(10)	☑
	data2805	nchar(10)	☑

б)

Рис. 2.1. Структура довідників бази даних:

а) таблиця даних входу, б) таблиця даних про учнів

База даних містить у собі ряд довідників, що мають обмежений доступ, і ряд таблиць загального користування, серед них такі, як users (дана таблиця містить інформацію про користувачів програми для забезпечення доступу до БД із програми) і таблиці окремо на кожен робочий день (дані таблиці містять інформацію учнів , відвідування та оплату ними занять).

2.3.3. Формування запитів до таблиць бази даних

Для формування звітів і поточної роботи з базою використовується механізм запитів, який дозволяє поєднати декілька таблиць і вибрати з них лише необхідні поля з записами, які відповідають заданим умовам. Так, для отримання даних для перевірки логіну та пароля користувача для входу використовується наступний запис у форматі мови *SQL:*

public static string sqlusers;

public static SqlConnection conusers = new SqlConnection();

public static SqlCommand cmdusers = new SqlCommand("", conusers);

public static DataTable dtusers;

public static SqlDataAdapter dausers;

*DbClass.sqlusers = "SELECT * FROM users;";*

DbClass.cmdusers.CommandType = CommandType.Text;

DbClass.cmdusers.CommandText = DbClass.sqlusers;

DbClass.dausers = new SqlDataAdapter(DbClass.cmdusers);

DbClass.dtusers = new DataTable();

DbClass.dausers.Fill(DbClass.dtusers);

SqlCommandBuilder comandbuilderusers = new SqlCommandBuilder(DbClass.dausers);

Для отримання даних про учнів конкретного уроку виконується наступний запит:

public static SqlCommand cmdMonday1400 = new SqlCommand("", conusers);

*DbClass.sqlusers = "SELECT * FROM monday1400;";*

DbClass.cmdMonday1400.CommandType = CommandType.Text;

DbClass.cmdMonday1400.CommandText = DbClass.sqlusers;

DbClass.daMonday1400 = new SqlDataAdapter(DbClass.cmdMonday1400);

DbClass.dtMonday1400 = new DataTable();

DbClass.daMonday1400.Fill(DbClass.dtMonday1400);

SqlCommandBuilder comandbuilderMonday1400 = new SqlCommandBuilder(DbClass.daMonday1400);

Щоб додати нове поле в базу даних із програми використовується наступний запит:

SqlCommandBuilder comandbuilderMonday1400 = new SqlCommandBuilder(DbClass.daMonday1400);

DbClass.cmdMonday1400.CommandText = "ALTER TABLE Monday1400 ADD [data" + data + "] nchar(10) NULL; ";

DbClass.cmdMonday1400.ExecuteNonQuery();

DbClass.cmdMonday1400.CommandText = "UPDATE Monday1400 SET [data" + data + "] = 0 WHERE [data" + data + "] IS NULL";

DbClass.cmdMonday1400.ExecuteNonQuery();

*DbClass.sqlusers = "SELECT * FROM Monday1400;";*

DbClass.cmdMonday1400.CommandText = DbClass.sqlusers;

DbClass.dtMonday1400 = new DataTable();

DbClass.daMonday1400.Fill(DbClass.dtMonday1400);

myDataGridMonday1400.ItemsSource = DbClass.dtMonday1400.DefaultView;

DbClass.daMonday1400.Update(DbClass.dtMonday1400);

Даний підхід дозволяє в повній мірі використовувати інструменти мови *SQL*, які реалізовано в СУБД *MySQL*. Саме на основі даної СУБД організовано редагування, збереження і обробка даних в програмі.

2.4. Висновки до розділу

В даному розділі розкрито, що при проєктуванні бази даних, по-перше, необхідно продумати елементи, котрі будуть оброблятися. Після цього за допомогою списку розроблених раніше таблиць треба скласти список усіх інформаційних елементів (полів), з котрими буде працювати програма та визначити для кожного поля оптимальний тип даних і розмір, відстежуючи при цьому, щоб не було дублювання полів.

Використання стандартних СУБД дозволяє значно спростити етапи проєктування, створення та налагодження зв’язків таблиць.

Так для реалізації бази даних програми було використано онлайн БД *Microsoft Azure*, яка в повному обсязі відповідає вимогам до СУБД і підтримає стандартну мову запитів *SQL*. До того ж розробники даної СУБД додали ряд спеціалізованих функцій для їх використання при розробці інтернет-програм.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЕЛЕКТРОННОГО ЖУРНАЛУ ВИКЛАДАЧА

3.1. Опис мови програмування

При написані програми була використана мова *C#. C#* – це сучасна, об'єктно–орієнтована та безпечна для програмування мова програмування для платформи *.NET*. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою *Microsoft Research* (належить *Microsoft*). *C#* дозволяє розробникам створювати багато типів безпечних та надійних додатків, що працюють в екосистемі *.NET*. *C#* сягає своїм корінням з сімейства мов *C* і буде одразу знайомий програмістам на *C, C, Java* та *JavaScript. C#* – це об'єктно–орієнтована, орієнтована на компоненти мова програмування. *C#* надає мовні конструкції для прямої підтримки цих концепцій, роблячи *C#* природною мовою, на якій можна створювати та використовувати програмні компоненти.

З моменту свого зародження *C#* додав функції для підтримки нових навантажень та нових практик дизайну програмного забезпечення. Кілька функцій *C#* допомагають створювати надійні та довговічні програми. Збір сміття автоматично вилучає пам’ять, зайняту недосяжними невикористаними об’єктами. Типи, що допускають відхилення, захищають від змінних, які не посилаються на виділені об'єкти. Обробка винятків забезпечує структурований та розширюваний підхід до виявлення та відновлення помилок. Лямбда–вирази підтримують методи функціонального програмування. Синтаксис інтегрованого мовного запиту *(LINQ)* створює загальний шаблон роботи з даними з будь–якого джерела.

Мовна підтримка асинхронних операцій забезпечує синтаксис для побудови розподілених систем. *C#* має уніфіковану систему типів. Усі типи *C#,* включаючи примітивні типи, такі як *int* і *double*, успадковуються від одного кореневого типу об'єкта. Усі типи мають спільний набір загальних операцій. Цінності будь–якого типу можна зберігати, транспортувати та використовувати послідовно. Крім того, *C#* підтримує як визначені користувачем типи посилань, так і типи значень. *C#* дозволяє динамічно розподіляти об'єкти та зберігати в лінійці легкі конструкції. *C#* підтримує загальні методи та типи, які забезпечують підвищену безпеку та продуктивність типу. *C#* надає ітератори, які дозволяють реалізаторам класів колекцій визначати власну поведінку для клієнтського коду. *C#* робить акцент на встановленні версій, щоб забезпечити розвиток програм та бібліотек з часом сумісним способом. Аспекти дизайну *C#,* на які безпосередньо вплинули міркування щодо версій, включають окремі модифікатори віртуальних та перевизначених, правила дозволу перевантаження методів та підтримку явних оголошень членів інтерфейсу.

Програми *C#* працюють на *.NET*, віртуальній системі виконання, що називається середовищем виконання загальної мови (*CLR*), і набором бібліотек класів. *CLR* – це впровадження корпорацією Майкрософт загальномовної інфраструктури (*CLI*), міжнародного стандарту. *CLI* – це основа для створення середовищ виконання та розробки, в яких мови та бібліотеки працюють безперебійно.

Вихідний код, написаний на *C#*, компілюється на проміжну мову (*IL*), що відповідає специфікації *CLI*. Код *IL* та ресурси, такі як растрові зображення та рядки, зберігаються у збірці, як правило, з розширенням *.dll*. Збірка містить маніфест, який надає інформацію про типи та версію збірки.

Мовна сумісність – ключова особливість *.NET*. Код *IL*, створений компілятором *C#*, відповідає загальній специфікації типу (*CTS*). Код *IL*, згенерований із *C#*, може взаємодіяти з кодом, створеним із версій *.NET F #, Visual Basic, C ++* або будь–якої з більш ніж 20 інших мов, сумісних із *CTS*. Одна збірка може містити кілька модулів, написаних різними мовами *.NET*, і типи можуть посилатися один на одного, так ніби вони були написані однією мовою.

На додаток до служб виконання, *.NET* також включає великі бібліотеки. Ці бібліотеки підтримують багато різних навантажень. Вони організовані в простори імен, що забезпечують широкий спектр корисних функцій для всього, від введення та виведення файлів до маніпуляцій рядками до аналізу *XML*, до фреймворків веб–додатків до елементів керування *Windows Forms*. Типовий додаток *C#* широко використовує бібліотеку класів *.NET* .

3.2. Опис платформи програмування

WPF (*Windows Presentation Foundation*) – одна з останніх моделей проєктування призначеного для користувача інтерфейсу (*GUI*), яка використовується в рамках *Microsoft .NET*.

GUI – це графічний інтерфейс користувача (*Graphical User Interface*). До складу *Windows* входять *GUI*, що функціонують на вашому комп'ютері або в браузері.

GUI фреймворк дозволяє створювати додатки з різними графічними елементами, такими як: лейбли, текстові поля і т.д. Якщо при проєктуванні графічного інтерфейсу не використовувати подібні фреймворки, то доведеться "вручну" малювати все графічні елементи і реагувати на дії користувачів (введення даних з елементів управління: клавіатура, миша і т.д.). Вищеописане завдання в підсумку є досить трудомістким. Як наслідок, переважна більшість розробників використовують *GUI* фреймворки для реалізації подібних базових задач, приділяючи більше часу логіці програми.

Існує безліч подібних фреймворків, але для *.NET* розробників найцікавішими *є WinForms* і *WPF*. Модель *WPF* є більш новою, але *Microsoft* не перестає підтримувати застарілу *WinForms*. Між даними фреймворками існує чимало відмінностей, але їх призначення є однаковим – спростити процес розробки додатків з відмінним графічним інтерфейсом.

3.3. Опис програми

Програма «електронний журнал» була створена для зручності роботи викладача курсів робототехніки у школі «*RoboLab*». Основне завдання програми: ведення звітності про відвідування учнями занять робототехніки та оплату цих же занять.

Програма містить 3 робочих вікна:

1) Вікно входу

Рис. 3.1. Вікно входу

Вікно входу служить для авторизації викладача у системі. Авторизація створена задля безпеки даних. Аби доступ та можливість зміни інформації була лише у допущених до цього осіб, в нашому випадку викладачів курсів робототехніки. Авторизація здійснюється шляхом введення логіну та паролю і відповідні поля.

При некоректному вході програма повідомить про помилку (рис. 3.2). При коректному вході ви увійдете в систему та перейдете до вікна вибору робочого дня.

Рис. 3.2. Некоректний вхід в систему

2) Вікно вибору робочого дня

Рис. 3.3. Вікно вибору робочого дня

Вікно вибору робочого дня містить 7 кнопок, по одній на кожен день тижня. При натисканні на якусь із них ми переходимо до вікна робочого дня із даними про відповідний день тижня.

3) Вікно робочого дня

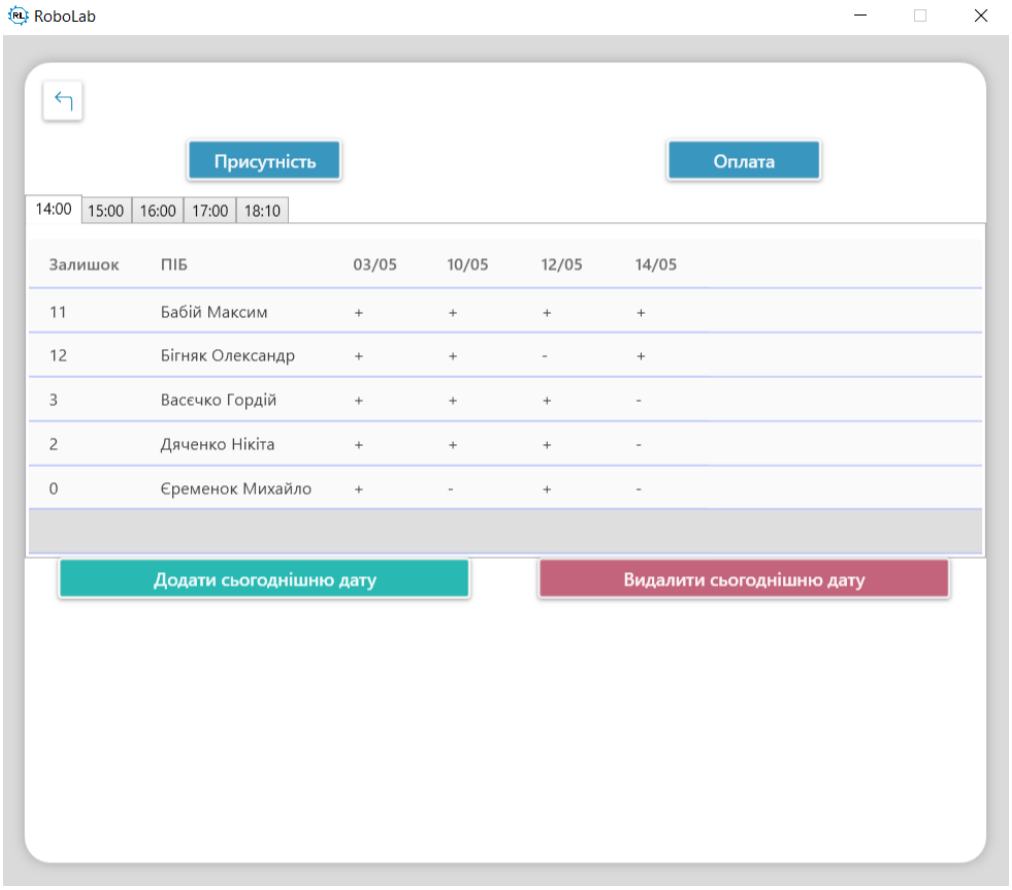

Рис. 3.4. Вікно робочого дня

Вікно робочого дня містить два можливих стани:

1. Присутність – відображає інформацію про учнів та відвідування ними занять.

2. Оплата – відображає інформацію про учнів та оплати ними занять.

Перехід між станами здійснюється за допомогою натискань відповідних кнопок (рис. 3.5)

Рис. 3.5. Кнопки зміни стану

Вікно робочого у кожному із вище вказаних станах містить вкладки, що відповідають конкретному уроку (рис. 3.6.)

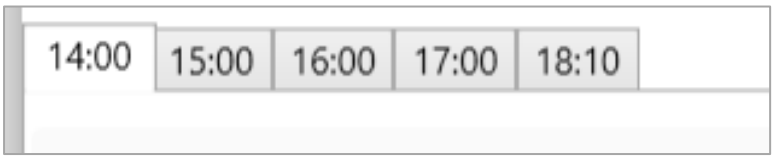

Рис. 3.6. Вкладки для вибору уроку

Вкладки реалізовані за допомогою елемента *TabControl*. Кожна вкладка це окремий *TabItem.*

При виборі конкретного часу уроку ви отримаєте таблицю із інформацією про дітей що відвітують саме цей урок (рис. 3.7)

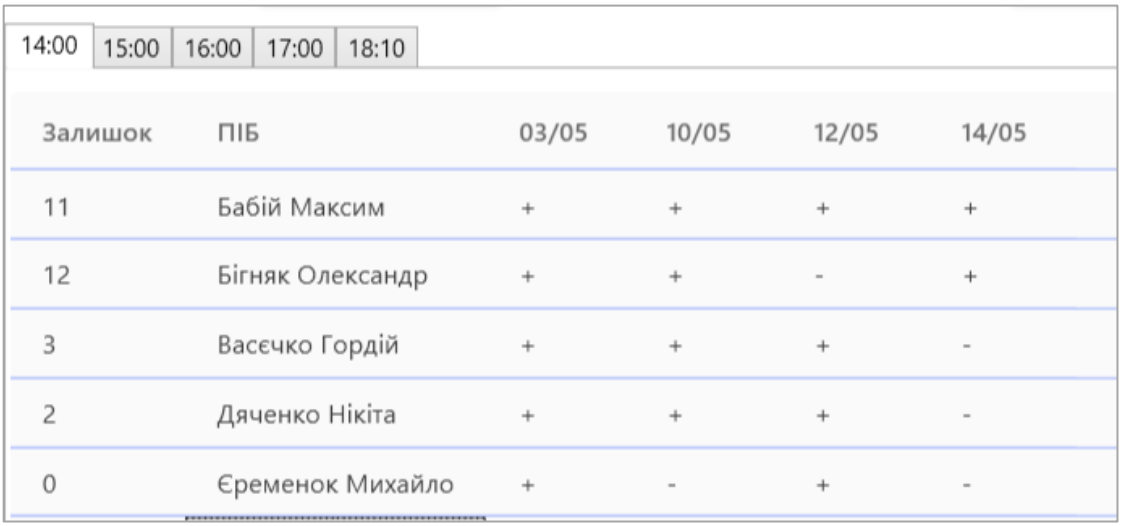

Рис. 3.7. Таблиця з інформацією про учнів

Перша колонка у таблиці «Залишок»– автоматично розраховує скільки зашилось оплачених занять у конкретного учня. Розрахунок іде за принципом сума усіх відвіданих занять мінус сума усіх оплачених занять.

```
public void count_Plus()
    {
     {
        for (int i = 0; i < myDataGridOplataMonday1400.Items.Count – 1; i++)
```

```
 {
 int oplataCount = 0;
 for (int j = myDataGridOplataMonday1400.Columns.Count – 1; j > 1; j—)
 {
oplataCount += int.Parse(DbClass.dtOplMonday1400.Rows[i][j].ToString());
DbClass.dtOplMonday1400.Rows[i][0] = oplataCount.ToString().Trim();
            }
          }

          for (int i = 0; i < myDataGridMonday1400.Items.Count – 1; i++)
          {
            int plusCount = 0;
            for (int j = myDataGridMonday1400.Columns.Count – 1; j > 1; j—)
            {
              if ((DbClass.dtMonday1400.Rows[i][j].ToString().Trim() == "+"))
              {
                plusCount++;
              }
            }
  int temp = int.Parse(DbClass.dtOplMonday1400.Rows[i][0].ToString()) – plusCount;
            DbClass.dtMonday1400.Rows[i][0] = temp.ToString().Trim();
          }
}
```

Друга колонка у таблиці вказує на прізвище та ім’я учня. Колонки далі містять дату уроку.

У стані «відвідування» – навпроти кожного учня у відповідній колонці дати ставимо «+» або «–» в залежності від того був присутнім учень на занятті чи ні.

У стані «оплата» – навпроти учня у відповідній колонці дати ставимо число, що відповідає кількості оплачених занять. Зміна значень відбувається дуже просто. Натискаємо на відповідну комірку та вписуємо значення із клавіатури (рис. 3.8).

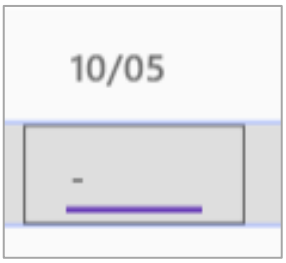

Рис. 3.8. Зміна значення комірки

Дані автоматично оновляться та збережуться після їх зміни. Ця функція працює завдяки обробнику подій *SelectedCellsChanged* , який спрацьовує відразу після зміни комірки в таблиці.

```
private void myDataGridMonday1400_SelectedCellsChanged(object sender, SelectedCellsChangedEventArgs e)
        {
            count_Plus();
            DbClass.daMonday1400.Update(DbClass.dtMonday1400);
        }
```

Внизу таблиці є дві кнопки для додавання нового стовпчика із поточною датою та видалення його (для випадку якщо випадково додали стовпчик) (рис. 3.9)

Рис. 3.9. Кнопки додавання та видалення стовпчика

При натисканні на кнопку «додати сьогоднішню дату» у таблиці створюється стовпчик із поточною датою заповнений «0». Стовпчик створюється для усіх уроків відкритого робочого дня і для стану «відвідування» і для стану «оплата».

```
DateTime date1 = DateTime.Today;
```

```
string data = date1.ToString("ddMM");
SqlCommandBuilder comandbuilderMonday1400 = new
SqlCommandBuilder(DbClass.daMonday1400);
DbClass.cmdMonday1400.CommandText = "ALTER TABLE
Monday1400 ADD [data" + data + "] nchar(10)  NULL; ";
DbClass.cmdMonday1400.ExecuteNonQuery();
DbClass.cmdMonday1400.CommandText = "UPDATE Monday1400 SET
[data" + data + "] = 0  WHERE [data" + data + "] IS NULL";
DbClass.cmdMonday1400.ExecuteNonQuery();
DbClass.sqlusers = "SELECT * FROM Monday1400;";
DbClass.cmdMonday1400.CommandText = DbClass.sqlusers;
DbClass.dtMonday1400 = new DataTable();
DbClass.daMonday1400.Fill(DbClass.dtMonday1400);
myDataGridMonday1400.ItemsSource =
DbClass.dtMonday1400.DefaultView;
DbClass.daMonday1400.Update(DbClass.dtMonday1400);
```

При натисканні на кнопку «видалити сьогоднішню дату» із таблиці видаляється стовпчик із поточною датою.

```
DateTime date1 = DateTime.Today;
string data = date1.ToString("ddMM");
SqlCommandBuilder comandbuilderMonday1400 = new
SqlCommandBuilder(DbClass.daMonday1400);
DbClass.cmdMonday1400.CommandText = "ALTER TABLE Monday1400 DROP
COLUMN [data" + data + "] ;";
DbClass.cmdMonday1400.ExecuteNonQuery();
DbClass.sqlusers = "SELECT * FROM Monday1400;";
DbClass.cmdMonday1400.CommandText = DbClass.sqlusers;
DbClass.dtMonday1400 = new DataTable();
DbClass.daMonday1400.Fill(DbClass.dtMonday1400);
myDataGridMonday1400.ItemsSource = DbClass.dtMonday1400.DefaultView;
```

DbClass.daMonday1400.Update(DbClass.dtMonday1400);

У стані «оплата» перший стовпчик таблиці вказує на кількість оплачених уроків. Дані обраховуються автоматично шляхом суми усіх оплат відповідного учня (рис. 3.10)

Оплачено	ПІБ	03/05	10/05	12/05	14/05	28/05
15	Бабій Максим	10	0	5	0	0

Рис. 3.10. Таблиця у стані «оплата»

3.4. Тестування обробки запитів до бази даних

Для перевірки швидкості обробки запитів було проведено тестування кожної із таблиць на вивід та зміну інформації. Тест проводився в умовах ідентичних до робочих, формування сторінки з результатами пошуку займає в середньому менше секунди (10 спроб на комп`ютері із наступною конфігурацією: *Intel(R) Core(TM) i5–6200U CPU 2.30GHz*, ОЗУ 8 *GB,* твердотільний накопичувач 512 *GB,* під керуванням операційної системи *Windows 10*).

3.5. Висновки до розділу

Даний розділ присвячено програмній реалізації проєкту. В даному розділі описано розроблені вікна програми та представлено програмні коди, які використовувались для організації даних форм. Всі елементи розроблено на мові *C#* на базі платформи *WPF*, для зв’язування із БД *Microsoft Azure* використані спеціальні функції доступу до таблиць бази даних. Також було проведено тестування системи на швидкість взаємодії з сервером. Дане тестування проводилось під час обробки запиту на пошук по базі даних.

ВИСНОВКИ

Під час виконання дипломного проєкту було проведено дослідження оцифрування паперового документообігу з метою спрощення доступу для перегляду і зміни данних та було створено додаток для ОС *Windows 10.*

Даний додаток вирішує такі задачі:

− оперативний доступ до інформації за допомогою мережі Інтернет;

− синхронізація даних про учнів на усіх пристроях , на яких встановлено даний додаток;

− пришвидшення робочогу процесу контролю відвідування та оплати занять;

− позбавлення від потреби використання паперових документів.

У дипломному проєкті було створено додаток «Електронний журнал» викладача навчальних курсів на базі платформи *WPF* для роботи з базою даних, яка відображає інформацію про учнів, відвідування занять та їх оплату.

Дозволяє відслідковувати та змінювати данні із будь-якої точки планети за допомогою глобальної мережі Інтернет.

Використання стандартних СУБД дозволяє значно спростити етапи проєктування, створення та налагодження зв’язків таблиць. З огляду на те, що проєкт носить інформативну, а не комерційну направленість, відсутня необхідність в закупівлі СУБД, які, хоча і мають трохи більше можливостей та обслуговуються професійними працівниками технічної підтримки, але не можуть конкурувати з умовно безкоштовними СУБД, користувачі яких отримують ті ж самі привілеї за умови не використання даних СУБД для отримання прибутку безпосередньо.

Так для реалізації бази даних програми було використано СУБД *MySQL* та пробний період у сервісі хостингу БД *Microsoft Azure*, яка в повному обсязі відповідає вимогам до СУБД і підтримає стандартну мову запитів *SQL*.

До того ж розробники даної СУБД додали ряд спеціалізованих функцій для їх використання при розробці настільних додатків.

Практична цінність результатів дослідження полягає у наданні можливості працівникам освітнього закладу вести звітність про учнів на заняттях робототехніки, а також відвідування та оплату цих занять.

Проведено тест програми, що показав виведення таблиць, доступних для редагування, в середньому менше секунди (10 спроб на комп`ютері із наступною конфігурацією: *Intel(R) Core(TM) i5–6200U CPU 2.30GHz,* ОЗУ 8 *GB,* твердотільний накопичувач 512 *GB,* під керуванням операційної системи *Windows 10*).

У рамках виконання дипломного проєкту був розроблений додаток для контролю відвідування та оплати курсів робототехніки на платформі ОС *Windows 10*.

Додаток є простим та інтуїтивно зрозумілим для користувача, має дружній та приємний інтерфейс, оскільки за основу для дизайну було використано *Material Design* від компанії *Google*

Основні результати дипломного проєкту:

1) Сформульовано переваги «Електронного журналу» перед паперовим аналогом, визначено особливості їх застосування при програмуванні додатків.

2) Проаналізовано технології доступу до даних з використанням мережі Інтернет.

3) Розроблено базу даних для збереження інформації про учнів, відвідування та оплату занять.

4) Розроблено систему «Електронний журнал» викладача навчальних курсів на базі платформи *WPF*.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проєкти) випускників Національного авіаційного університету. – К.: НАУ, 2017. – 63 с.

2. ДСТУ 3008–95 Документація. Звіти у сфері науки і техніки. Структура і правила оформлення.

3. Ульман, Д. Введение в системы баз данных /Д.Ульман, Д.Уидом; Пер. с англ. – М.: Лори, 2000. – 512 с

4. Проєктирование информационных систем: курс лекций: учеб. Пособие для студентов вузов, обучающихся по специальностям в области информ. технологий / В.И. Грекул, Г.Н. Денищенко, Н.Л. Коровкина. – М.: Интернет–Ун–т Информ. технологий, 2015. – 304с

5. К.Дж.Дейт. Введение в системы баз данных.: Пер. С англ. К.: Диалектика, 2008. – 784с

6. Базы данных: Учебник для ВУЗов / Под ред. А.Д.Хомоненко – СПб: Корона принт, 2012. – 416 с

7. Кузнєцов А.М. Мартинов В.В. Вимоги до графічного дизайну і юзабіліті освітніх порталів. – М .: Просвята, 2003. – С. 365–420.

8. Кузнецов А. Дизайн і юзабіліті освітніх Інтернет ресурсів. // *E LearningWorld,* 2004. – № 3. – С. 30–37.

9. Вимоги до сучасних інтерфейсів користувача. Демидов Дмитро Григорович. Журнал: Вісник Московського державного університету друку, 2016. [Електронний ресурс] – Режим доступу до ресурсу: *https://cyberleninka.ru/article/n/trebovaniya–k–sovremennym–polzovatelskim–interfeysam/viewer*

10. Створення архітектури програми. [Електронний ресурс] – Режим доступу до ресурсу: *https://habr.com/ru/post/276593/*

11. Шарп Д. *Microsoft Visual C#.* Подробное руководство / Дж. Шарп. –СПб: Питер, 2017. – 848с. – (8–е издание).

12. ТОП 7 популярных языков программирования, востребованных в 2020. [Електронний ресурс] – Режим доступу до ресурсу: *https://devsday.ru/blog/details/3627*

13. Рейтинг мов програмування 2021: частка *Python* зменшується, а *TypeScript* обійшов *C++* [Електронний ресурс] – Режим доступу до ресурсу: *https://dou.ua/lenta/articles/language–rating–jan–2021/*

14. Фреймворки и библиотеки для кроссплатформенной разработки десктопных программ [Електронний ресурс] – Режим доступу до ресурсу: *https://habr.com/ru/post/528614/*

15. Пять лучших фреймворков для разработки приложений для ПК [Електронний ресурс] – Режим доступу до ресурсу: *https://senior.ua/articles/5–luchshih–freymvorkov–dlya–razrabotki–prilozheniy–dlya–pk*

ДОДАТОК А

Лістинг коду програмних модулів

MainWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using first_try.classes;

namespace first_try
{
    public partial class MainWindow : Window
   {
       public MainWindow()
       {
           DbClass.openConnection();
```

```
        DbClass.sqlusers = "SELECT * FROM users;";
        DbClass.cmdusers.CommandType = CommandType.Text;
        DbClass.cmdusers.CommandText = DbClass.sqlusers;
        DbClass.dausers = new SqlDataAdapter(DbClass.cmdusers);
        DbClass.dtusers = new DataTable();
        DbClass.dausers.Fill(DbClass.dtusers);
        SqlCommandBuilder comandbilderusers = new
SqlCommandBuilder(DbClass.dausers);
    }
    private void Button_Login_Click(object sender, RoutedEventArgs e)
    {
      string login = LoginBox.Text.Trim();
        string pass = PasswordBox.Password.Trim();
        InitializeComponent();
        DbClass.openConnection();
        DbClass.sqlusers = "SELECT * FROM users;";
        DbClass.cmdusers.CommandType = CommandType.Text;
        DbClass.cmdusers.CommandText = DbClass.sqlusers;
        DbClass.dausers = new SqlDataAdapter(DbClass.cmdusers);
        DbClass.dtusers = new DataTable();
        DbClass.dausers.Fill(DbClass.dtusers);
        SqlCommandBuilder comandbilderusers = new
SqlCommandBuilder(DbClass.dausers);
    }
private void Button_Login_Click(object sender, RoutedEventArgs e)
    {
        string login = LoginBox.Text.Trim();
        string pass = PasswordBox.Password.Trim();
        int count = DbClass.dtusers.Rows.Count;
        while (count > 0) {
```

```
            count—;
            if (login == DbClass.dtusers.Rows[count][0].ToString().Trim() && pass ==
DbClass.dtusers.Rows[count][1].ToString().Trim())
            {
                Workday workday = new Workday();
                workday.Show();
Close();
            }
            else if (login != DbClass.dtusers.Rows[count][0].ToString())
            {
                LoginBox.Background = Brushes.LightPink;
                LoginBox.BorderBrush = Brushes.DarkRed;
                PasswordBox.Clear();
            }
            else if (login == DbClass.dtusers.Rows[count][0].ToString() && pass !=
DbClass.dtusers.Rows[count][1].ToString())
            {
LoginBox.Background = Brushes.Transparent;
                LoginBox.BorderBrush = Brushes.DarkRed;
                PasswordBox.Clear();
            }
        }
    }
private void PasswordBox_KeyDown(object sender, KeyEventArgs e)
    {
        if (e.Key == Key.Enter)
        {
            Button_Login_Click(login_button, null);
        }
    }
```

MainWindow.xaml

```
<Window x:Class="first_try.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup–compatibility/2006"
    xmlns:local="clr–namespace:first_try"
    mc:Ignorable="d"
    xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"
    Title="RoboLab" Height="700" Width="800" Icon="minilogo.png"
ResizeMode="CanMinimize" WindowStartupLocation="CenterScreen"
    Loaded="Window_Loaded" WindowStyle="ThreeDBorderWindow" >
  <Grid  Background="#ffdadada">
    <Border MinWidth="400" Margin="20,10,20,10" Padding="40"
Background="White" VerticalAlignment="Center" CornerRadius="20" Height="620"  >
      <Border.Effect>
        <DropShadowEffect  BlurRadius="15"  Color="LightGray"  ShadowDepth="0"
/>
      </Border.Effect>
      <StackPanel Margin="–30" >
        <Image HorizontalAlignment="Center" Source="logo.png"
VerticalAlignment="Top" Height="345" Margin="101,0,144.6,0" Width="488"
OpacityMask="Black"  />
        <TextBox x:Name="LoginBox" FontSize="20" Padding="8 0 0 0"
materialDesign:HintAssist.Hint="Введіть логін" Style="{StaticResource
```

```
MaterialDesignFloatingHintTextBox}"
materialDesign:TextFieldAssist.UnderlineBrush="#0760ad"
    materialDesign:HintAssist.Foreground="#0760ad" Margin="150 40 150 0"
CaretBrush="#FF0096D5"/>
        <PasswordBox x:Name="PasswordBox" KeyDown="PasswordBox_KeyDown"
FontSize="20" materialDesign:HintAssist.Hint="Введіть пароль"
Style="{StaticResource MaterialDesignFilledPasswordFieldPasswordBox}"
materialDesign:TextFieldAssist.UnderlineBrush="#0760ad"
materialDesign:HintAssist.Foreground="#0760ad" Margin="150,0"
CaretBrush="#FF0096D5" />
        <Button x:Name="login_button" Click="Button_Login_Click"
HorizontalAlignment="Center" Margin="0 30 0 0" Background="#3594bd"
Content="ВхоД" BorderThickness="0" />
      </StackPanel>
    </Border>
  </Grid>
</Window>
```

Monday.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Data;
using System.Configuration;
using System.Data.SqlClient;
using System.Text;
```

```
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using first_try.classes;

namespace first_try
{
   public partial class Monday : Window
   {
public void Window_Loaded(object sender, RoutedEventArgs e)
      {
         if (DbClass.conusers.State == ConnectionState.Closed)
         {
            DbClass.openConnection();
            Console.WriteLine("open");
         }
         prusytnist_Click(sender, e);
```

```
    }
private void stolbik_Click(object sender, RoutedEventArgs e)
    {
        try
        {
DateTime date1 = DateTime.Today;
            string data = date1.ToString("ddMM");
SqlCommandBuilder comandbuilderMonday1400 = new SqlCommandBuilder(DbClass.daMonday1400);
            DbClass.cmdMonday1400.CommandText = "ALTER TABLE Monday1400 ADD [data" + data + "] nchar(10)  NULL; ";
            DbClass.cmdMonday1400.ExecuteNonQuery();
            DbClass.cmdMonday1400.CommandText = "UPDATE Monday1400 SET [data" + data + "] = 0  WHERE [data" + data + "] IS NULL";
            DbClass.cmdMonday1400.ExecuteNonQuery();
            DbClass.sqlusers = "SELECT * FROM Monday1400;";
            DbClass.cmdMonday1400.CommandText = DbClass.sqlusers;
            DbClass.dtMonday1400 = new DataTable();
            DbClass.daMonday1400.Fill(DbClass.dtMonday1400);
            myDataGridMonday1400.ItemsSource = DbClass.dtMonday1400.DefaultView;
            DbClass.daMonday1400.Update(DbClass.dtMonday1400);
    SqlCommandBuilder comandbuilderOplMonday1400 = new SqlCommandBuilder(DbClass.daOplMonday1400);
```

DbClass.cmdOplMonday1400.CommandText = "ALTER TABLE MondayOplata1400 ADD [data" + data + "] nchar(10) NULL; ";

DbClass.cmdOplMonday1400.ExecuteNonQuery();

DbClass.cmdOplMonday1400.CommandText = "UPDATE MondayOplata1400 SET [data" + data + "] = 0 WHERE [data" + data + "] IS NULL";

DbClass.cmdOplMonday1400.ExecuteNonQuery();

*DbClass.sqlusers = "SELECT * FROM MondayOplata1400;";*

DbClass.cmdOplMonday1400.CommandText = DbClass.sqlusers;

DbClass.dtOplMonday1400 = new DataTable();

DbClass.daOplMonday1400.Fill(DbClass.dtOplMonday1400);

myDataGridOplataMonday1400.ItemsSource =

SqlCommandBuilder comandbuilderMonday1500 = new SqlCommandBuilder(DbClass.daMonday1500);

DbClass.cmdMonday1500.CommandText = "ALTER TABLE Monday1500 ADD [data" + data + "] nchar(10) NULL; ";

DbClass.cmdMonday1500.ExecuteNonQuery();

DbClass.cmdMonday1500.CommandText = "UPDATE Monday1500 SET [data" + data + "] = 0 WHERE [data" + data + "] IS NULL";

DbClass.cmdMonday1500.ExecuteNonQuery();

*DbClass.sqlusers = "SELECT * FROM Monday1500;";*

DbClass.cmdMonday1500.CommandText = DbClass.sqlusers;

DbClass.dtMonday1500 = new DataTable();

DbClass.daMonday1500.Fill(DbClass.dtMonday1500);

myDataGridMonday1500.ItemsSource = DbClass.dtMonday1500.DefaultView;

DbClass.daMonday1500.Update(DbClass.dtMonday1500);

SqlCommandBuilder comandbuilderOplMonday1500 = new SqlCommandBuilder(DbClass.daOplMonday1500);

DbClass.cmdOplMonday1500.CommandText = "ALTER TABLE MondayOplata1500 ADD [data" + data + "] nchar(10) NULL; ";

DbClass.cmdOplMonday1500.ExecuteNonQuery();

DbClass.cmdOplMonday1500.CommandText = "UPDATE MondayOplata1500 SET [data" + data + "] = 0 WHERE [data" + data + "] IS NULL";

DbClass.cmdOplMonday1500.ExecuteNonQuery();

*DbClass.sqlusers = "SELECT * FROM MondayOplata1500;";*

DbClass.cmdOplMonday1500.CommandText = DbClass.sqlusers;

DbClass.dtOplMonday1500 = new DataTable();

DbClass.daOplMonday1500.Fill(DbClass.dtOplMonday1500);

myDataGridOplataMonday1500.ItemsSource = DbClass.dtOplMonday1500.DefaultView;

DbClass.daOplMonday1500.Update(DbClass.dtOplMonday1500);

SqlCommandBuilder comandbuilderMonday1600 = new SqlCommandBuilder(DbClass.daMonday1600);

DbClass.cmdMonday1600.CommandText = "ALTER TABLE Monday1600 ADD [data" + data + "] nchar(10) NULL; ";

DbClass.cmdMonday1600.ExecuteNonQuery();

DbClass.cmdMonday1600.CommandText = "UPDATE Monday1600 SET [data" + data + "] = 0 WHERE [data" + data + "] IS NULL";

DbClass.cmdMonday1600.ExecuteNonQuery();

*DbClass.sqlusers = "SELECT * FROM Monday1600;";*

DbClass.cmdMonday1600.CommandText = DbClass.sqlusers;

DbClass.dtMonday1600 = new DataTable();

DbClass.daMonday1600.Fill(DbClass.dtMonday1600);

myDataGridMonday1600.ItemsSource = DbClass.dtMonday1600.DefaultView;

DbClass.daMonday1600.Update(DbClass.dtMonday1600);

SqlCommandBuilder comandbuilderOplMonday1600 = new SqlCommandBuilder(DbClass.daOplMonday1600);

DbClass.cmdOplMonday1600.CommandText = "ALTER TABLE MondayOplata1600 ADD [data" + data + "] nchar(10) NULL; ";

DbClass.cmdOplMonday1600.ExecuteNonQuery();

DbClass.cmdOplMonday1600.CommandText = "UPDATE MondayOplata1600 SET [data" + data + "] = 0 WHERE [data" + data + "] IS

DbClass.cmdOplMonday1600.ExecuteNonQuery();

*DbClass.sqlusers = "SELECT * FROM MondayOplata1600;";*

DbClass.cmdOplMonday1600.CommandText = DbClass.sqlusers;

DbClass.dtOplMonday1600 = new DataTable();

DbClass.daOplMonday1600.Fill(DbClass.dtOplMonday1600);

myDataGridOplataMonday1600.ItemsSource = DbClass.dtOplMonday1600.DefaultView;

DbClass.daOplMonday1600.Update(DbClass.dtOplMonday1600);

SqlCommandBuilder comandbuilderMonday1700 = new SqlCommandBuilder(DbClass.daMonday1700);

```
            DbClass.cmdMonday1700.CommandText = "ALTER TABLE Monday1700 ADD
[data" + data + "] nchar(10)  NULL; ";
            DbClass.cmdMonday1700.ExecuteNonQuery();
            DbClass.cmdMonday1700.CommandText = "UPDATE Monday1700 SET [data"
+ data + "] = 0  WHERE [data" + data + "] IS NULL";
            DbClass.cmdMonday1700.ExecuteNonQuery();
            DbClass.sqlusers = "SELECT * FROM Monday1700;";
            DbClass.cmdMonday1700.CommandText = DbClass.sqlusers;
            DbClass.dtMonday1700 = new DataTable();
DbClass.daMonday1700.Fill(DbClass.dtMonday1700);
            myDataGridMonday1700.ItemsSource = DbClass.dtMonday1700.DefaultView;
            DbClass.daMonday1700.Update(DbClass.dtMonday1700);
   rename_Columns();
        }
catch (Exception)
        {
            //
        }
    }
private void delete_Click(object sender, RoutedEventArgs e)
    {
        try
        {
```

```
        DateTime date1 = DateTime.Today;
SqlCommandBuilder comandbuilderMonday1400 = new SqlCommandBuilder(DbClass.daMonday1400);
        DbClass.cmdMonday1400.CommandText = "ALTER TABLE Monday1400 DROP COLUMN [data" + data + "] ;";
        DbClass.cmdMonday1400.ExecuteNonQuery();
        DbClass.sqlusers = "SELECT * FROM Monday1400;";
        DbClass.cmdMonday1400.CommandText = DbClass.sqlusers;
        DbClass.dtMonday1400 = new DataTable();
        DbClass.daMonday1400.Fill(DbClass.dtMonday1400);
        myDataGridMonday1400.ItemsSource = DbClass.dtMonday1400.DefaultView;
        DbClass.daMonday1400.Update(DbClass.dtMonday1400);
SqlCommandBuilder comandbuilderOplMonday1400 = new SqlCommandBuilder(DbClass.daOplMonday1400);
DbClass.cmdOplMonday1400.CommandText = "ALTER TABLE MondayOplata1400 DROP COLUMN [data" + data + "] ;";
        DbClass.cmdOplMonday1400.ExecuteNonQuery();
        DbClass.sqlusers = "SELECT * FROM MondayOplata1400;";
        DbClass.cmdOplMonday1400.CommandText = DbClass.sqlusers;
        DbClass.dtOplMonday1400 = new DataTable();
        DbClass.daOplMonday1400.Fill(DbClass.dtOplMonday1400);
        myDataGridOplataMonday1400.ItemsSource = DbClass.dtOplMonday1400.DefaultView;
        DbClass.daOplMonday1400.Update(DbClass.dtOplMonday1400);
```

```
    rename_Columns();
        }
        catch (Exception)
        {
            //
        }
    }
public void rename_Columns()
    {
      try
        {
            myDataGridMonday1400.Columns[0].Header = "Залишок";
            myDataGridMonday1400.Columns[1].Header = "ПІБ";
            for (int i = myDataGridMonday1400.Columns.Count – 1; i > 1; i—)
            {
         if (myDataGridMonday1400.Columns[i].Header.ToString().Length <= 5)
                continue;
            }
            String column_header =
myDataGridMonday1400.Columns[i].Header.ToString();
            column_header = column_header.Remove(0, 4);
            column_header = column_header.Insert(2, "/");
            myDataGridMonday1400.Columns[i].Header = column_header;
```

```
            }
        }
        catch (Exception)
        {
            //
        }
//оплата————————————————————————————————————————————————————
—
        try
        {
            myDataGridOplataMonday1400.Columns[0].Header = "Оплачено";
            myDataGridOplataMonday1400.Columns[1].Header = "ПІБ";
            for (int j = myDataGridOplataMonday1400.Columns.Count – 1; j > 1; j—)
            {
                if (myDataGridOplataMonday1400.Columns[j].Header.ToString().Length <=
5)
                {
                    continue;
                }
    string                                  column_header                                =
myDataGridOplataMonday1400.Columns[j].Header.ToString();
                column_header = column_header.Remove(0, 4);
                column_header = column_header.Insert(2, "/");
```

```
            myDataGridOplataMonday1400.Columns[j].Header = column_header;
        }
    }
    catch (Exception)
    {
        //
    }
public void count_Plus()
    {
    //14:00————————————————————————————————————————————————————
    try
    {
        for (int i = 0; i < myDataGridOplataMonday1400.Items.Count – 1; i++)
        {
            int oplataCount = 0;

            for (int j = myDataGridOplataMonday1400.Columns.Count – 1; j > 1; j—)
            {
                oplataCount += int.Parse(DbClass.dtOplMonday1400.Rows[i][j].ToString());
                DbClass.dtOplMonday1400.Rows[i][0] = oplataCount.ToString().Trim();
    }
```

```
            }
            for (int i = 0; i < myDataGridMonday1400.Items.Count – 1; i++)
            {
                int plusCount = 0;
                for (int j = myDataGridMonday1400.Columns.Count – 1; j > 1; j—)
                {
                    if ((DbClass.dtMonday1400.Rows[i][j].ToString().Trim() == "+"))
                    {
                        plusCount++;
                    }
                }
                int temp = int.Parse(DbClass.dtOplMonday1400.Rows[i][0].ToString()) –
plusCount;
                DbClass.dtMonday1400.Rows[i][0] = temp.ToString().Trim();
            }
catch (Exception)
        {
            //
        }
private void oplata_Click(object sender, RoutedEventArgs e)
    {
        TabControl.Visibility = Visibility.Collapsed;
        TabControlOplata.Visibility = Visibility.Visible;
```

```
        count_Plus();
        rename_Columns();
    }
    public void prusytnist_Click(object sender, RoutedEventArgs e)
    {
        TabControl.Visibility = Visibility.Visible;
        count_Plus();
        rename_Columns();
        TabControlOplata.Visibility = Visibility.Collapsed;
    }
public Monday()
    {
        InitializeComponent();
        // відвідування
        DbClass.sqlusers = "SELECT * FROM monday1400;";
        DbClass.cmdMonday1400.CommandType = CommandType.Text;
        DbClass.cmdMonday1400.CommandText = DbClass.sqlusers;
        DbClass.daMonday1400 = new SqlDataAdapter(DbClass.cmdMonday1400);
        DbClass.dtMonday1400 = new DataTable();
        DbClass.daMonday1400.Fill(DbClass.dtMonday1400);
        SqlCommandBuilder comandbuilderMonday1400 = new
SqlCommandBuilder(DbClass.daMonday1400);
        //оплата
```

```
        DbClass.sqlusers = "SELECT * FROM mondayOplata1400;";
        DbClass.cmdOplMonday1400.CommandType = CommandType.Text;
DbClass.cmdOplMonday1400.CommandText = DbClass.sqlusers;
        DbClass.daOplMonday1400 = new
SqlDataAdapter(DbClass.cmdOplMonday1400);
        DbClass.dtOplMonday1400 = new DataTable();
     DbClass.daOplMonday1400.Fill(DbClass.dtOplMonday1400);
        SqlCommandBuilder comandbuilderOplMonday1400 = new
SqlCommandBuilder(DbClass.daOplMonday1400);
private void Grid_KeyDown(object sender, KeyEventArgs e)
     {
        if (e.Key == Key.Escape)
        {
           back_Click(back, null);
        }
     }
private void back_Click(object sender, RoutedEventArgs e)
     {
        Workday workday = new Workday();
        workday.Show();
        Close();
     }
  private void myDataGrid_Loaded(object sender, RoutedEventArgs e)
```

```
    {
        myDataGridMonday1400.ItemsSource = DbClass.dtMonday1400.DefaultView;
        myDataGridOplataMonday1400.ItemsSource = DbClass.dtOplMonday1400.DefaultView;
            myDataGridMonday1500.ItemsSource = DbClass.dtMonday1500.DefaultView;
        myDataGridOplataMonday1500.ItemsSource = DbClass.dtOplMonday1500.DefaultView;
myDataGridMonday1600.ItemsSource = DbClass.dtMonday1600.DefaultView;
        myDataGridOplataMonday1600.ItemsSource = DbClass.dtOplMonday1600.DefaultView;
        myDataGridMonday1700.ItemsSource = DbClass.dtMonday1700.DefaultView;
        myDataGridOplataMonday1700.ItemsSource = DbClass.dtOplMonday1700.DefaultView;
        myDataGridMonday1810.ItemsSource = DbClass.dtMonday1810.DefaultView;
        myDataGridOplataMonday1810.ItemsSource = DbClass.dtOplMonday1810.DefaultView;
        count_Plus();
        rename_Columns();
    }
private void myDataGridMonday1400_SelectedCellsChanged(object sender, SelectedCellsChangedEventArgs e)
    {
        count_Plus();
```

```
        DbClass.daMonday1400.Update(DbClass.dtMonday1400);
    }
private void myDataGridOplataMonday1400_SelectedCellsChanged(object sender, SelectedCellsChangedEventArgs e)
    {
        count_Plus();
        DbClass.daOplMonday1400.Update(DbClass.dtOplMonday1400);
    }
  }
   }
}
```