**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ОСТРОЗЬКА АКАДЕМІЯ”  
Кафедра економіко-математичного моделювання та інформаційних  
технологій**

**ЗВІТ   
ПРО ПРОХОДЖЕННЯ**

**Навчальної практики з дисципліни “Програмування на C# ”**

здобувача вищої освіти

першого (бакалаврського) рівня

другого року навчання групи Кн-22

спеціальності 122 Комп’ютерні науки

ОПП **«**Комп’ютерні науки**»**

Жарчинського Маріана Сергійовича

Термін практики з 05.02.2024 по 19.02.2024р.

База практики – кафедра економіко-математичного моделювання

та інформаційних технологій

Керівник практики від університету

Кандидат технічних наук, доцент

Наталія ЖУКОВСЬКА

Члени комісії :

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Острог - 2024 р.

**Зміст**

[ВСТУП 1](#_Toc159002738)

[ПРОБЛЕМА 2](#_Toc159002739)

[РОБОТА З БАЗОЮ ДАНИХ 3](#_Toc159002742)

[АРХІТЕКТУРА 4](#_Toc159002743)

[ОПИС РЕЗУЛЬТАТУ 5](#_Toc159002744)

[Користувацька частина 5](#_Toc159002745)

[Сервісна частина 7](#_Toc159002746)

[Репозиторій 8](#_Toc159002747)

[ЗОВНІШНІ ПАКЕТИ 10](#_Toc159002748)

[ВИСНОВОК 11](#_Toc159002749)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 12](#_Toc159002750)

# **ВСТУП**

Наразі автоматизовані системи управління складом і відстеження руху товарів стають все більш важливими для успішної роботи великих роздрібних магазинів. З розвитком технологій і зростанням обсягів даних використання баз даних стає ключовим аспектом для забезпечення ефективного управління та аналізу інформації. Розробка програмного забезпечення, що забезпечує автоматизацію процесів управління складом і відстеження руху товарів, є актуальним завданням для торгівельно-логістичної галузі.

У цьому звіті буде детально розглянуто процес розробки автоматизованої системи управління складом і відстеження руху товарів у великому роздрібному магазині. Зокрема, будуть описані етапи проектування та впровадження бази даних, архітектура програмного забезпечення, розробка функціональних можливостей додатків та використання пакетів сторонніх програм для оптимізації роботи з базами даних та підвищення продуктивності.

Під час реалізації проекту я мав можливість отримати практичні навички створення та використання баз даних з використанням різних технік програмування та отримати нові знання у сфері розробки програмного забезпечення для управління складськими процесами та відстеження руху товарів.

# **ПРОБЛЕМА**



**Тема: Система автоматизованого управління складом та відстеження руху товарів у великому роздрібному магазині**

Студент повинен розробити WinForms додаток на мові програмування C#, який взаємодіє з базою даних для управління інформацією. Додаток повинен включати такі функціональні можливості.

**База даних:**

Використання **Entity Framework або іншу ORM** для взаємодії з базою даних. Додати можливість зберігати файли або картинки. Створіть базу даних з трьома або більше таблицями.

**Архітектура:**

* Розробіть додаток на основі трирівневої архітектури з використанням WinForms для інтерфейсу, бізнес-логіки та доступу до даних.
* Використовуйте репозиторій для забезпечення взаємодії з базою даних, забезпечуючи розділення логіки доступу до даних від решти додатку.

**Функціональність додатку:**

1. Використання NuGet для підключення сторонніх бібліотек, для спрощення розробки.
2. Створіть докладну документацію з описом структури бази даних, функціональності додатку та інструкціями користувача.
3. Додавання та управління товарами:
   1. Можливість додавання нових товарів до бази даних, включаючи інформацію про назву, категорію, ціну, кількість тощо.
   2. Реалізація функції редагування та видалення існуючих товарів.
4. Управління складом:
   1. Система відстеження залишків товарів на складі.
   2. Автоматичне оновлення кількості товарів при продажу або отриманні нового товару.
5. Оформлення та відстеження замовлень:
   1. Функціонал для створення нових замовлень, включаючи вибір товарів та їх кількість.
   2. Відображення історії замовлень та їх статусів.
6. Робота з файлами та папками:
   1. Збереження та завантаження даних у файли (наприклад, JSON) для забезпечення персистентності.
   2. Можливість експорту та імпорту списку товарів та замовлень.
7. Використання сторонніх пакетів:
   1. Використання NuGet пакетів для оптимізації роботи з базою даних або для реалізації додаткових функціональностей.
8. WinForms:
   1. Розробка графічного інтерфейсу з використанням WinForms для зручного користування додатком.
   2. Забезпечення можливості експорту звітів у різні формати (PDF, Excel).

# **РОБОТА З БАЗОЮ ДАНИХ**

Під час проектування бази даних було створено 7 основних сутностей, але пізніше я додав сутність “User” для збереження та використання інформації про акаунти користувачів(див. Рис. *1*).

У результаті, кінцева база даних містила 8 таблиць. Таблиці поділені , щоб було зручно працювати з товаром і замовленням з певних товарів (див. Рис. *2*). Майже всі таблиці мали зв’язок “Один до багатьох”, окрім таблиці “Users”, яка взагалі не була пов’язана з іншими таблицями , оскільки вона в подальшому використовувалася для інших цілей в роботі програми.

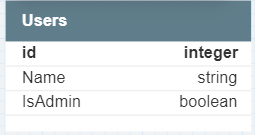


Рис. 1 Таблиця для логіну користувачів

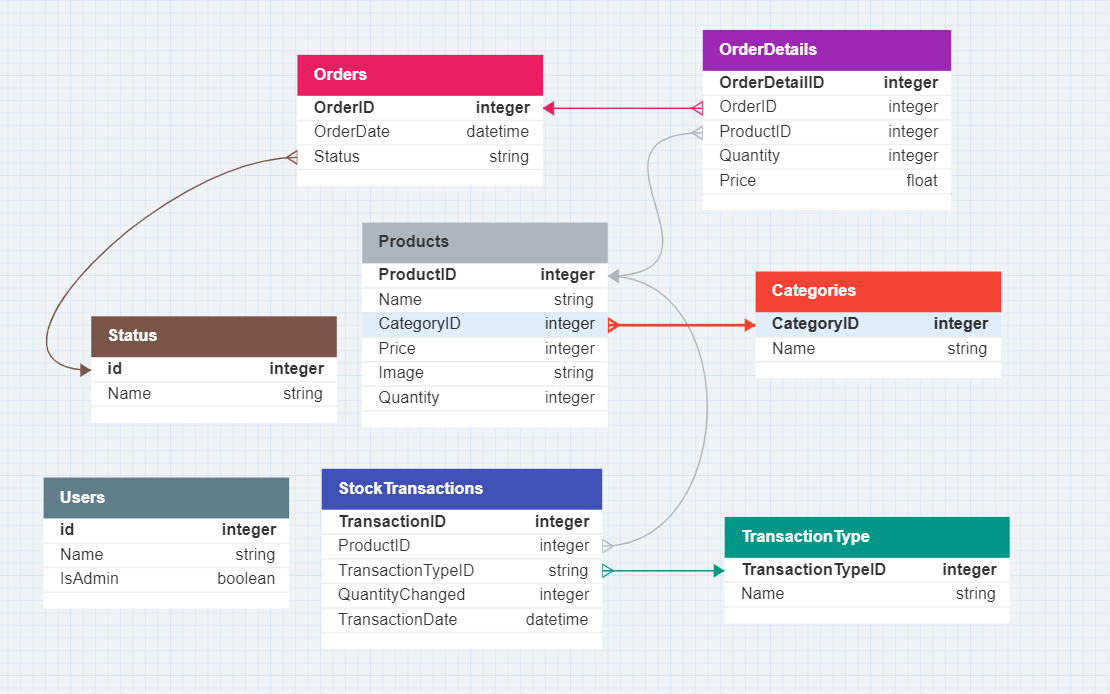


Рис. 2 Загальний вигляд всіх таблиць бази даних

# **АРХІТЕКТУРА**

Оскільки в завданні було чітко визначено структуру – я створив трьохрівневу архітектуру(див. Рис. *3*). Тобто у мене вийшло реалізував Рішення(Solution “Practise”), у якому було створено 3 проекта, а саме:

* Проект для (сутностей), який був створений як бібліотека класів (Storage.Core), що містить всі сутності, потрібні для програми та контекст бази даних.
* Проект для роботи із сутностями (додавання, видалення, редагування тощо) у вигляді бібліотеки класів (Storage.DataManager), що містить репозиторій, у який я імплементував інтрефейс.
* Проект для взаємодії з користувачем (інтерфейс користувача), створений як додаток WPF (Storage.UI).

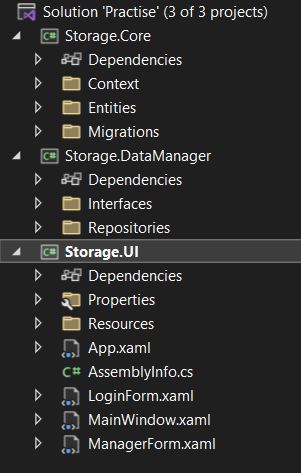


Рис. 3 Архітектура проекту

# **ОПИС РЕЗУЛЬТАТУ**

## **Користувацька частина**

Для користувача інтерфейс був розбитий на 3 частини:

* Форма для логіну (див. Рис. *4*).
* Форма для менеджера складу(див. Рис. *5* та Рис. *6*).
* Форма для адміністратора (див. Рис. *9*).

Під час запуску програми спершу користувачу показується форма для входу у свій акаунт.Перевіряється , чи дані були введені правильно. Далі в залежності від його ролі у системі запускається певна форма відповідно до рівня користувача, а поточна форма логіну закривається.

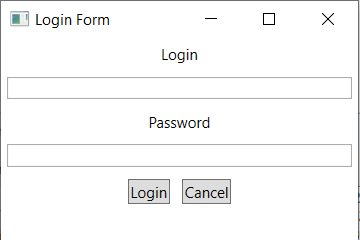


Рис. 4 Форма авторизації

У загальному форма для робітника не сильно відрізняється від форми адміністратора . Для робітника стоять обмеження , що він не може змінювати таблиці , для нього є можливість перегляду таблиць та оформлення замовлення .

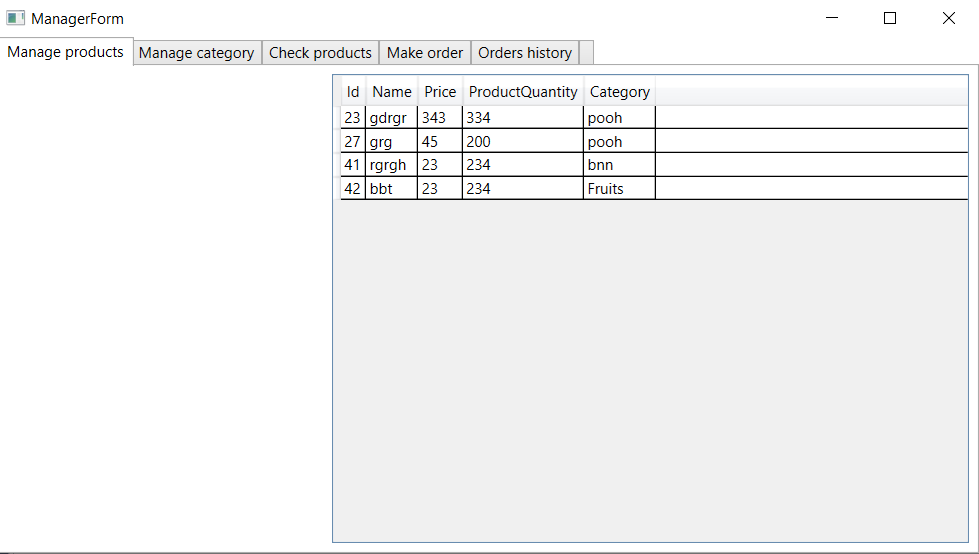


Рис. 5 Загальний вигляд форми робітника

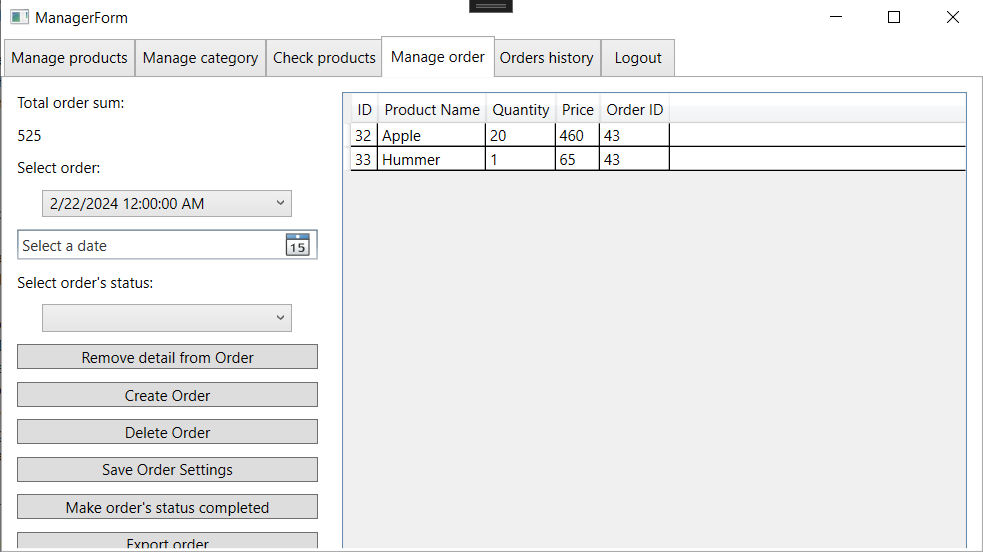


Рис. 6 частина форми для замовлення у робітника

В адміністратора натомість є всі можливості для додавання , віднімання та редагування всіх сутностей , а також все , що може робити робітник .

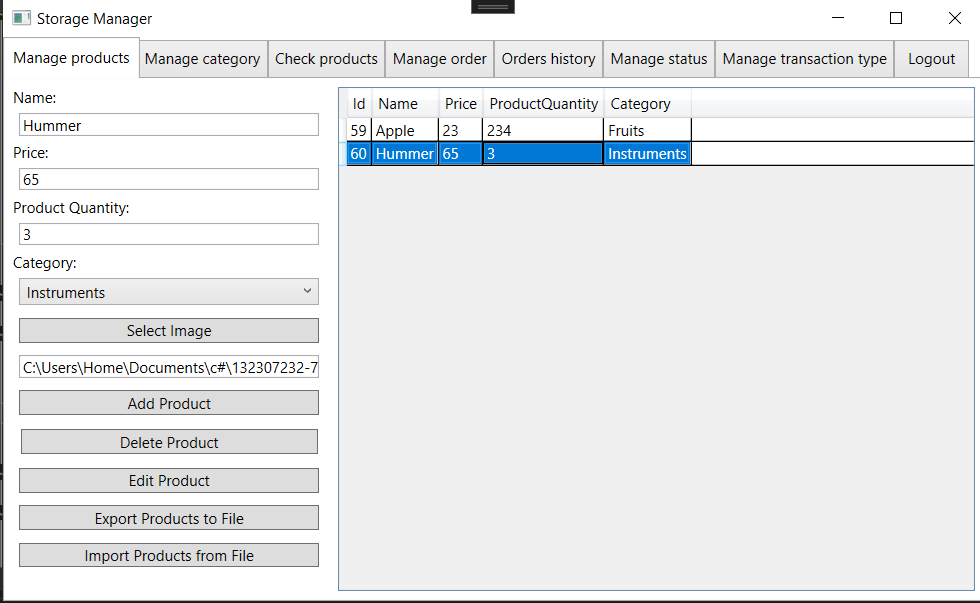


Рис. 7 частина форми для роботи адміністратора

## **Сервісна частина**

Для роботи форми із сутностями були імплементовані асинхронні методи у репозиторії, які виконують різні функції , такі як : додавання , редагування , видалення , збереження або LINQ запит. Адміністратор також може імпортувати або експортувати Товари із таблиці або у таблицю , використовуючи файл типу JSON або XLSX (див. Рис. *8*) окремо можливо імпортувати або експортувати Категорії Товарів із таблиці або у таблицю , використовуючи файл типу JSON або XLSX (див. Рис. *9*).

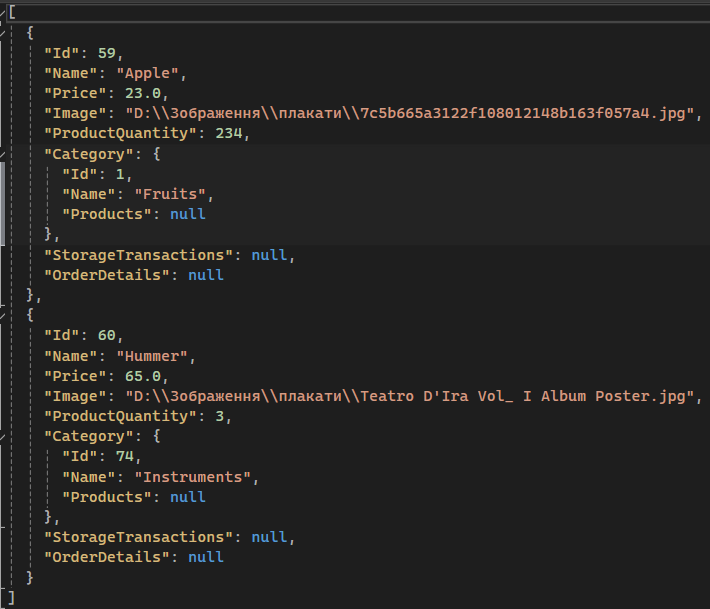


Рис. 8 Експортована таблиця Товарів у файл JSON

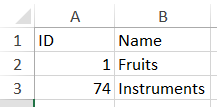


Рис. 9 Експортована таблиця Категорій Товарів у файл XLSX

## **Репозиторій**

Для ефективної взаємодії із сутностями було створено інтерфейс “IRepository” , якого в подальшому потрібно було імплементувати у репозиторій. Були створені асинхронні прототипи методів для “CRUD”, а також інші методи длі інших задач.

**Лістинг 1. Інтерфейс IRepository**

|  |
| --- |
| public interface IRepository<TEntity>  {  Task<List<TEntity>> GetAllAsync();  Task<TEntity?> GetByIdAsync(int id);  IQueryable<TEntity> GetQueryable();  void Insert(TEntity entity);  void Update(TEntity entity);  void Delete(TEntity entity);  Task SaveChangesAsync();  } |

Інтерфейс був імплементований у репозиторії . Крім методів для “CRUD” , були також реалізовані методи для витягнення всіх об’єктів з певної таблиці , збереження змін у базі даних , витягнення певного об’єкта за його Id з певної таблиці та метод для створення певного запиту , використовуючи LINQ.

**Лістинг 2. Репозиторій Repository**

|  |
| --- |
| public class Repository<TEntity> : IRepository<TEntity>  where TEntity : class  {  private readonly StorageContext \_context;  public Repository(StorageContext context)  {  this.\_context = context;  }  public async Task<List<TEntity>> GetAllAsync()  {  return await \_context.Set<TEntity>().ToListAsync();  }  public async Task<TEntity?> GetByIdAsync(int id)  {  return await \_context.Set<TEntity>().FindAsync(id);  }  public IQueryable<TEntity> GetQueryable()  {  return \_context.Set<TEntity>();  }  public void Insert(TEntity entity)  {  \_context.Set<TEntity>().Add(entity);  }  public void Update(TEntity entity)  {  \_context.Set<TEntity>().Update(entity);  }  public void Delete(TEntity entity)  {  \_context.Set<TEntity>().Remove(entity);  }  public Task SaveChangesAsync()  {  return \_context.SaveChangesAsync();  }  } |

Використовуючи цей репозиторій , потрібно для кожного класу створити унікальний контекст , який буде підв’язаний до певної таблиці. Всі контексти ініціалізувалися в методі , який буде спрацьовувати призапуску програми , оскільки цей метод був використаний у конструкторі форми.

**Лістинг 3. Контексти для різних таблиць бази даних**

|  |
| --- |
| private StorageContext dbContext;  private Repository<Category> \_categoryRepository;  private Repository<Product> \_productRepository;  private Repository<OrderDetails> \_orderDetailsRepository;  private Repository<Order> \_orderRepository;  private Repository<TransactionType> \_transactionTypeRepository;  private Repository<Status> \_statusRepository;    private void InitializeRepositories()  {  dbContext = new StorageContext();  \_categoryRepository = new Repository<Category>(dbContext);  \_productRepository = new Repository<Product>(dbContext);  \_orderDetailsRepository = new Repository<OrderDetails>(dbContext);  \_orderRepository = new Repository<Order>(dbContext);  \_transactionTypeRepository = new Repository<TransactionType>(dbContext);  \_statusRepository = new Repository<Status>(dbContext);  } |

# **ЗОВНІШНІ ПАКЕТИ**

1. Microsoft.EntityFrameworkCore (використана версія 7.0.15) - це бібліотека для мови програмування C#, яка надає доступ до бази даних з використанням Entity Framework Core (EF Core). EF Core - це технологія об'єктно-реляційного відображення (ORM), яка дозволяє взаємодіяти з реляційною базою даних, використовуючи об'єктно-орієнтований підхід.
2. Microsoft.EntityFrameworkCore.SqlServer (використана версія 7.0.15) - це частина пакету Entity Framework Core, яка надає можливість взаємодії з базами даних SQL Server. Цей пакет дозволяє розробникам здійснювати зв'язок з серверами SQL Server та працювати з ними за допомогою Entity Framework Core.
3. Microsoft.EntityFrameworkCore.Tools (використана версія 7.0.15) - це набір інструментів для Entity Framework Core, які допомагають в розробці та управлінні базами даних. Цей пакет містить різноманітні інструменти командного рядка, які дозволяють виконувати такі дії, як створення міграцій, застосування міграцій до бази даних, генерація коду моделі з існуючої бази даних, зворотне проектування бази даних та інші операції.
4. EPPlus (використана версія 7.0.8) – це бібліотека для мови програмування C#, яка дозволяє зчитувати, записувати та редагувати файли Excel безпосередньо з програмного коду. Вона широко використовується для автоматизації обробки даних у форматі Excel, створення звітів, таблиць та інших електронних документів в додатках інформаційних систем.
5. ExcelDataReader (використана версія 3.6.0) - це бібліотека для мови програмування C#, яка призначена для читання даних з файлів Excel. Вона дозволяє отримувати доступ до даних, що зберігаються у різних аркушах та стовпцях у форматі Excel, незалежно від версії Excel або платформи.
6. Newtonsoft.Json (використана версія 13.0.3) - це бібліотека для роботи з форматом JSON в платформі .NET. Вона надає зручні інструменти для серіалізації та десеріалізації об'єктів .NET у формат JSON і навпаки. Newtonsoft.Json дозволяє легко конвертувати об'єкти між форматом JSON і об'єктами .NET, що дозволяє зручно обмінюватися даними між різними додатками та сервісами. Вона широко використовується в розробці програмного забезпечення для обробки даних, взаємодії з веб-сервісами, зберігання конфігураційних даних та багато іншого.

# **ВИСНОВОК**

Під час виконання даної практичної роботи я отримав важливий досвід, який сприяв не лише поглибленню моїх знань у програмуванні, але й оволодінню новими технологіями та навичками.

У першу чергу, робота з WPF додатком дозволила мені здобути практичний досвід у створенні інтерфейсу користувача, що є ключовим елементом багатьох програмних рішень. Я оволодів основними принципами роботи з різноманітними елементами керування, такими як кнопки, текстові поля, списки та інші, та здатний ефективно їх використовувати для створення зручного та інтуїтивно зрозумілого інтерфейсу.

Далі, інтеграція додатка з базою даних дозволила мені оволодіти навичками роботи з великим обсягом даних та їх ефективного управління. Я здобув навички створення підключень до баз даних, виконання SQL-запитів для отримання, оновлення та видалення даних, а також використання ORM-фреймворків, наприклад Entity Framework, для спрощення роботи з базами даних.

Додатково, нові NuGet пакети, які я використовував у процесі роботи, виявилися незамінними помічниками. Вони забезпечили швидке та ефективне розв’язання певних завдань, допомагаючи оптимізувати процес розробки та полегшуючи взаємодію з базою даних.

У цілому, виконання цієї практичної роботи виявилося надзвичайно корисним і пізнавальним досвідом. Працюючи з WPF додатком та інтегруючи його з базою даних, я зміг значно поглибити свої знання в області програмування та розширити свої навички в розробці інтерфейсів користувача та управлінні даними. Усе це разом створило надійну основу для мого подальшого розвитку як програміста та відкрило нові можливості для вдосконалення моїх професійних навичок у майбутньому.

# **СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. ListView Overview. 2023 URL: <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/controls/listview-overview?view=netframeworkdesktop-4.8> (10.02.2024).
2. Create a simple data application with WPF and Entity Framework 6. 2022 URL: <https://learn.microsoft.com/en-us/visualstudio/data-tools/create-a-simple-data-application-with-wpf-and-entity-framework-6?view=vs-2022> (дата звернення: 08.02.2024).
3. Microsoft Learn Migrations Overview – EF Core. 2023 URL: <https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=vs> (дата звернення 06.02.2024).
4. [Milan Jovanović](https://www.youtube.com/@MilanJovanovicTech) How I Use The Generic Repository Pattern In Clean Architecture. 2023 URL: <https://www.youtube.com/watch?v=kwehxBDX_o8&ab_channel=MilanJovanovi%C4%87> (дата звернення 09.02.2024).
5. Language Integrated Query (LINQ). 2023 URL: <https://learn.microsoft.com/en-us/dotnet/csharp/linq/> (дата звернення 11.02.2024).

**ГРАФІК ПРОХОДЖЕННЯ**

**Навчальної практика з дисципліни "Програмування на C#"**

**здобувача 2-го курсу першого (бакалаврського) рівня вищої освіти**

**освітньої програми «Комп’ютерні науки»**

Маріана Жарчинського

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Дата | Назви робіт | Відмітки про виконання |
|
| 1 | 05.02.2024 | Розробка схеми таблиць для бази даних. | виконано |
| 2 | 06.02.2024 | Створення бази даних , міграцій. | виконано |
| 3 | 07.02.2024 | Створення репозиторію для роботи із сутностями. | виконано |
| 4 | 08.02.2024 | Ознайомлення із технологією WPF. | виконано |
| 5 | 09.02.2024 | Написання коду для UI на WPF. | виконано |
| 6 | 10.02.2024 | Написання коду для роботи із сутностями , використовуючи репозиторій. | виконано |
| 7 | 11.02.2024 | Зміна деяких полів сутностей , створення нових міграцій бази даних. | виконано |
| 8 | 12.02.2024 | Створення форми для логіну користувачів. | виконано |
| 9 | 13.02.2024 | Написання коду для логіки роботи форми для логіна. | виконано |
| 10 | 14.02.2024 | Редагування деяких методів для роботи із сутностями на самій формі. | виконано |
| 11 | 14.02.2024 | Зміна деяких полів сутностей , створення нових міграцій бази даних. | виконано |
| 12 | 15.02.2024 | Виправлення методів для імпорту та експорту 2-х сутностей. | виконано |
| 13 | 15.02.2024 | Виправлення методів для роботи з таблицею “Замовлення”. | виконано |
| 14 | 16.02.2024 | Написання звіту . | виконано |

Керівник

від бази практики                                      \_\_\_\_\_\_      Наталія ЖУКОВСЬКА

М.П.        (підпис)      (власне ім’я та прізвище)

**практики**