ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка
до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**Розробка програмного забезпечення для автоматизації процесів прийому та видачі замовлень швейного виробництва мовою програмування C#»**

Виконав: студент 4 курсу, групи ПД-43
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності/спеціалізації)

Кобиляцький І.А.
(прізвище та ініціали)

Керівник Гаманюк І.М.
(прізвище та ініціали)

Рецензент _____
(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти -«Бакалавр»
Спеціальність підготовки – 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____Негоденко О.В.
"____" _____2022 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТА

Кобиляцький Ілля Анатолійович
(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка програмного забезпечення для автоматизації процесів прийому та видачі замовлень швейного виробництва мовою програмування C#»
   Керівник роботи: Гаманюк І.М.старший викладач кафедри.
   (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
   затверджені наказом вищого навчального закладу від «16» лютого 2022 року № 22.
2. Строк подання студентом роботи «03» червня 2022 року.
3. Вхідні дані до роботи:
   Методи обробки та зберігання даних;
   Інструменти з розробки програмного забезпечення: Visual Studio, GIT, MS SQL Server management studio.
   Науково-технічна література з питань, пов'язаних з розробкою програмного рішення на основі прикладного програмного інтерфейсу;
   Технічна документація до об'єктно-реаляційного відображення Entiy Framework Core.
4. Зміст розрахування-пояснювальної записки (перелік питань, які потрібно розробити).
   4.1 Аналіз предметної області
   4.2 Вимоги та оцінка якості системи
   4.3 Проектування та реалізація веб-сервісу

4.4 Тестування системи

5. Перелік демонстраційного матеріалу (назва основних слайдів)

1. Перелік демонстраційного матеріалу (назва основних слайдів)

1. Мета, об'єкт та предмет дослідження
2. Актуальність роботи
3. Аналоги
4. Порівняння з аналогами
5. Технічне завдання
6. Програмні засоби реалізації
7. Діаграма пакетів
8. Діаграма класів
9. Приклад роботи пограмного забезпечення
10. Апробація результатів дослідження
11. Висновки

Дата видачі завдання «11» квітня 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз аналогів	11.04-13.04	Виконано
2	Створення вимог до системи	14.04-17.04	Виконано
3	Створення концепції та архітектури	18.04-25.04	Виконано
4	Розробка застосунку	25.04-19.05	Виконано
5	Моделювання та розробка бази даних	20.05-24.05	Виконано
6	Вступ, висновки, реферат	25.05-29.05	Виконано
7	Розробка обов‘язкових демонстраційних матеріалів	30.05-1.06	Виконано
8	Попередній захист роботи	02.06	Виконано
9	Здача роботи	03.06	

Студент _ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Об'єкт дослідження – процес прийому та видачі швейного виробництва

Предмет дослідження – програмне забезпечення для автоматизованого обліку швейного виробництва

Мета роботи – полегшити процес прийому та видачі швейного виробництва.

Методи дослідження - методи зберігання, передачі та обробки інформації, створення уніфікований процесу програмного забезпечення.

Було проведено аналіз існуючих аналогів таких як Excel, Access та Notepad++.

Недоліками цих продуктів перевантаженість інтерфейсу, відсутність безпеки даних та відсутність фільтру інформації.

Науковою новизною цієї роботи є інтуітивно зрозумілий інтерфейс без перевантаженості, присутня автентифікація та відкритість вихідного коду.

Програмне забезпечення має монолітну архітектуру і розроблено на фреймворку .NET Core та на мові програмування C# (.NET 5). В якості бази даних використано объектно-реляційну MSSQL Server. У додатку використані основні принципи SOLID, KISS, YAGNI з використання Dependency Injection.

Отже, було розроблено програмне забезпечення, та описано його. Воно допмогає спростити і покращити процес прийому та видачі швейного виробництва.

Галузь використання – шийне виробництво.

Зміст

ВСТУП

Обґрунтування вибору теми та її актуальність: в наш час шийне виробництво це дуже важлива індустрія. Кожен день виробляється та продається тисячі одиниць продукції виробленої на швейному підприємстві. Аболютно всі процеси виробництва автоматизуються, прийом та видача не є виключенням.

Для вирішення даної проблеми було спроектовано програмне забезпечення для автоматизації процесів прийому та видачі на шийному виробництві. І це в свою чергу дозволить полегшити та прискорити виробництво, та за рахунок цього збільшивши прибуток.

Ступінь вивчення проблеми: В відкритому доступі немає схожих продуктів для автоматизації процесів прийому та видачі швейного виробництва. Тому для вивчення було ознайомлено з схожими по функціоналу аналогами.

Об'єкт дослідження – процес прийому та видачі швейного виробництва

Предмет дослідження – програмне забезпечення для автоматизованого обліку швейного виробництва

Мета роботи – полегшити процес прийому та видачі швейного виробництва.

Завданням роботи є розробка програмного забезпечення з функціоналом: обліку.

Методика дослідження: Насамперед у аналізується та порівнюється існуючі програмні продукти. Тому спочатку я скористався пошуком в Інтернеті і в результаті побачив, що відкритих аналогів не має. Тому для порівняння було знайдено схожі за функціоналом аналоги. Вони мали занадто перевантажений інтерфейс та багато непотрібного функціоналу

Отже для вирішення даної проблеми потрібно створити власний продукт який буде виконувати потрібні функції схожих аналогів, та усувати недоліки та непотрібні функції

Наукова новизна роботи: полягає в вдосконаленні простого журналу в його електронну версію з покращеним функіоналом зручності та його доступності.

Практична значущість результатів: даний програмний продукт може бути використаний у будь якому шийному виробництві.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

В наш час автоматизація це невід’ємна частина будь-якої індустрії. На будь-якому виробництві присутня автоматизація всіх ланок виробництва від самих маленьких до великих. Це дозволяє пришвидшити роботу та обсяг виробництва, тим самим збільшивши дохід.

Шийне виробництво це одна з індустрій яку не обійшла автоматизація. Автоматизують усі процеси, від простої підготовки ткані до пошиття одягу.

Важливою частиною розробки є аналіз предметної області. Так як предметною областю проекту є створення програмного забезпечення для автоматизації процесів прийому та видачі замовлень швейного виробництва, потрібно проаналізувати програми схожі за функціоналом проекту та виділити їх переваги та недоліки.

1.1. Microsoft Excel

Microsoft Excel — це програма для роботи з електронними таблицями, що входить до пакету програм Microsoft Office. З Office 365 ви можете завантажити програму на свій жорсткий диск, а також матимете доступ до онлайн-версії. Онлайн-версія дає вам можливість обмінюватися файлами та співпрацювати з іншими над своїми файлами в режимі реального часу.

У електронних таблицях представлені таблиці значень, упорядковані в рядки та стовпці, якими можна маніпулювати математично, використовуючи як основні, так і складні арифметичні операції та функції. Програма може працювати на кількох платформах, таких як Windows, macOS, смартфони та планшети.

Рис. 1 – логотип Microsoft Excel

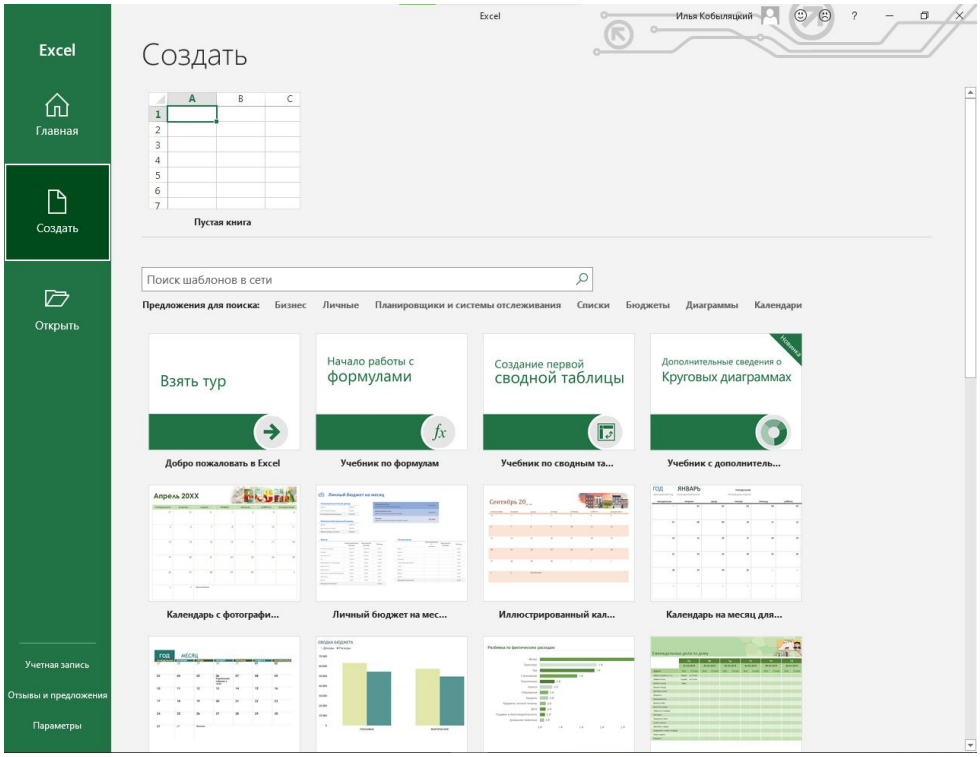


Рис. 2 – панель створення Excel файлу

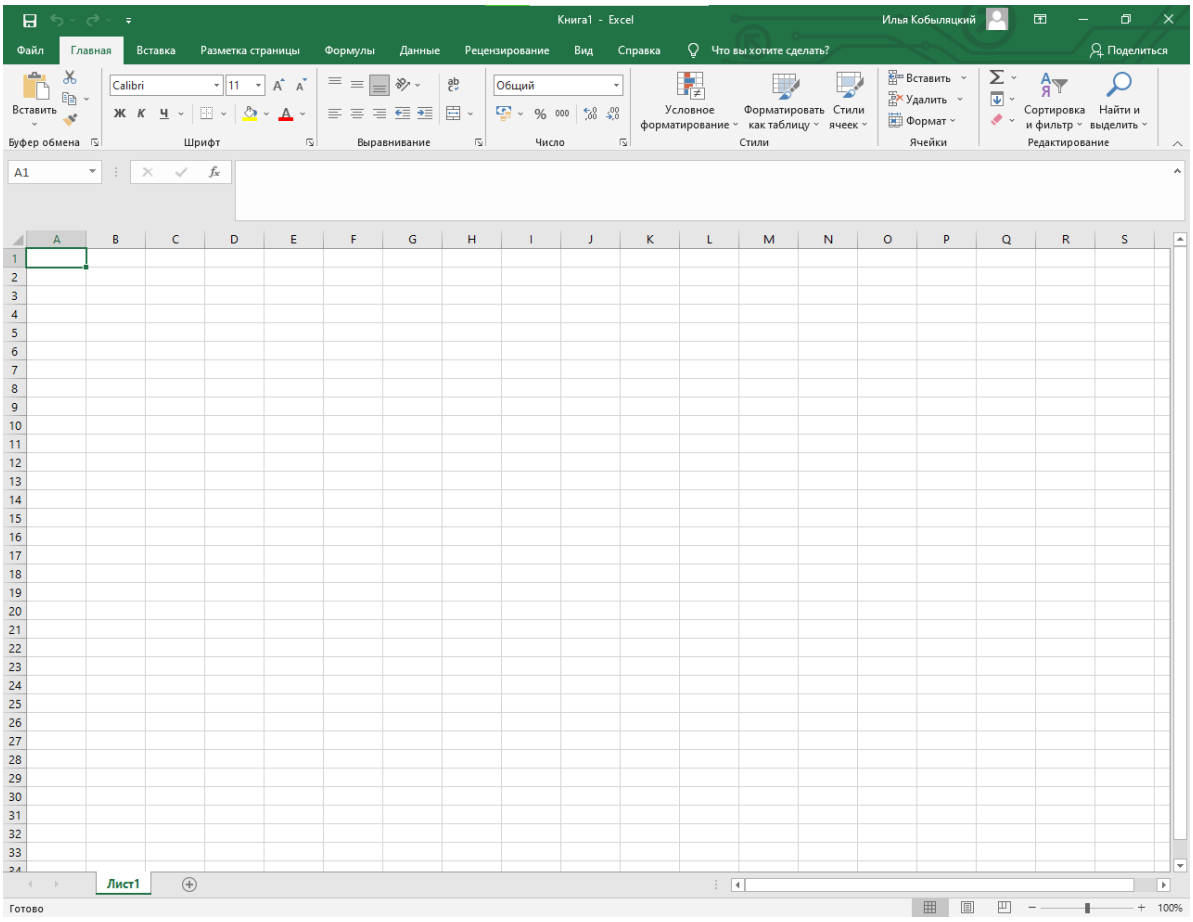


Рис. 3 – відкритий Excel файл

Для виконання потрібних операцій, не потрібен увесь функціонал Excel, тому було виділено основні необхідні переваги та недоліки.

Основні необхідні переваги Microsoft Excel:

- Зручність використання
- Пошук по фільтру

Основні необхідні функції які відсутні:

- Не використовується БД
- Відсутній безпечний доступ
- Не User friendly

1.2. Notepad++

Notepad++ — це безкоштовний (як у «вільній мові», а також у «безкоштовному пиві») редактор вихідного коду та заміна блокнота, який підтримує кілька мов. Використання в середовищі MS Windows регулюється Загальною публічною ліцензією GNU.

Заснований на потужному компоненті редагування Scintilla, Notepad++ написаний на C++ і використовує чистий Win32 API і STL, що забезпечує вищу швидкість виконання та менший розмір програми. Оптимізуючи якомога більше процедур, не втрачаючи зручності для користувача, Notepad++ намагається зменшити світові викиди вуглекислого газу. Використовуючи меншу потужність ЦП, комп'ютер може зменшити споживання енергії та зменшити споживання енергії, що призведе до більш екологічного середовища.

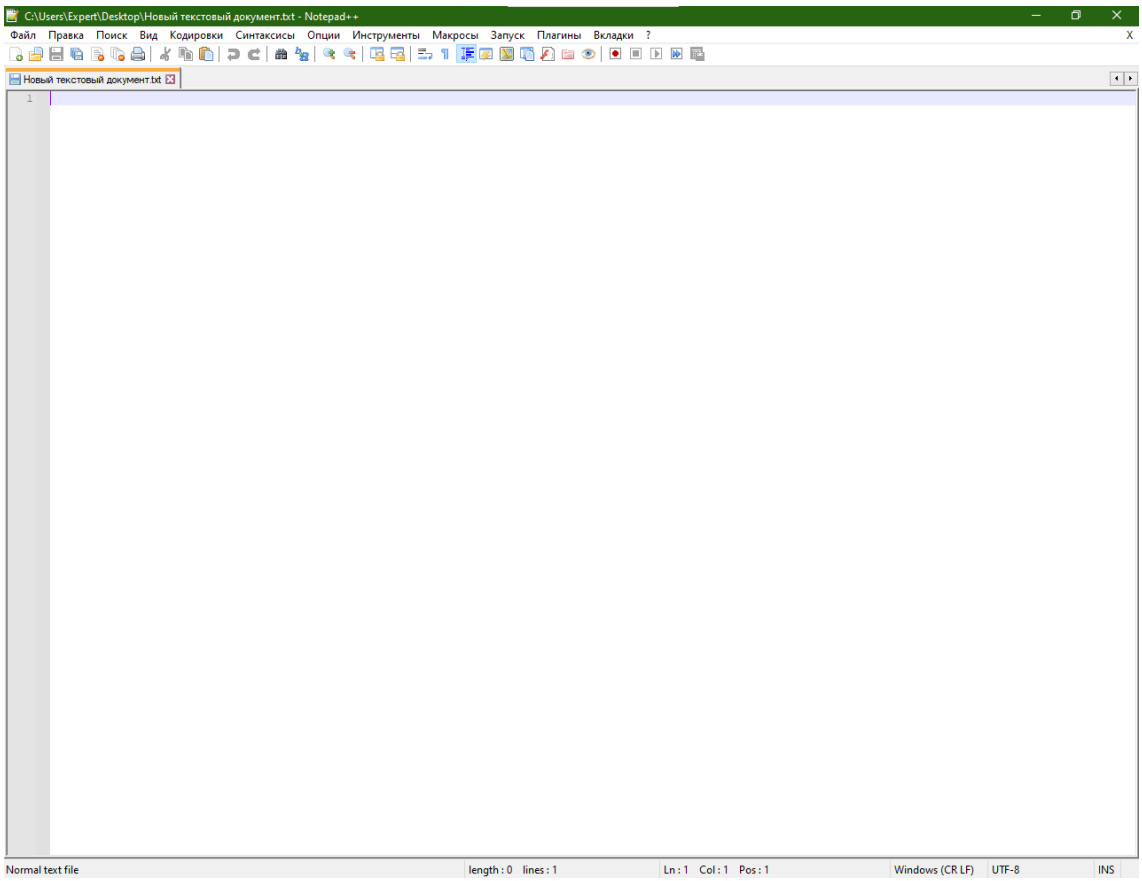


Рис. 4 – Notepad++

Для виконання потрібних операцій, не потрібен увесь функціонал Notepad++, тому було виділено основні необхідні переваги та недоліки.

Основні необхідні переваги Notepad++:

- Зручність використання
- User friendly

Основні необхідні функції які відсутні:

- Не використовується БД
- Відсутній безпечний доступ
- Не User friendly

1.3 Microsotf Access

Microsoft Access — це система керування реляційною базою даних від Microsoft, яка поєднує в собі реляційний Microsoft Jet Database Engine з графічним інтерфейсом користувача. Він є членом пакету програм Microsoft Office, включеним до професійної та вищої версії або продається окремо.

Access може використовувати дані, що зберігаються в Access/Jet, Microsoft SQL Server, Oracle або будь-якому ODBC-сумісному контейнері даних. Кваліфіковані розробники програмного забезпечення та архітектори даних використовують його для розробки прикладного програмного забезпечення. Відносно некваліфіковані програмісти та «досвідчені користувачі», які не є програмістами, можуть використовувати його для створення простих програм. Він підтримує деякі об'єктно-орієнтовані методи, але не може бути повністю об'єктно-орієнтованим інструментом розробки.

Access також називалася комунікаційна програма від Microsoft, призначена для конкуренції з ProComm та іншими програмами. Цей доступ виявився

невдалим і був припинений. Через роки Microsoft повторно використала назву для свого програмного забезпечення для баз даних.

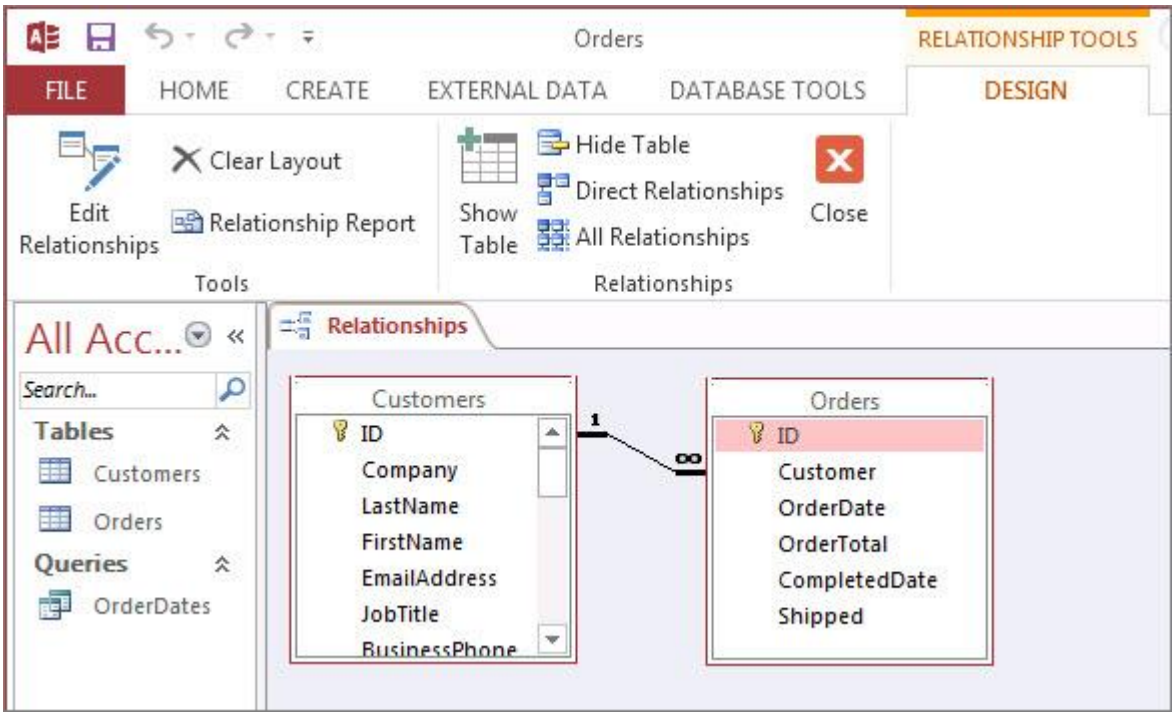


Рис. 5 – Microsoft Access

Основні необхідні переваги Microsoft Access:

- Зручність використання
- User friendly

Основні необхідні функції які відсутні:

- Не використовується БД
- Відсутній безпечний доступ
- Не User friendly

1.4 Висновок

Аналіз призводить до висновку, що у аналогів багато переваг, але є й загальні недоліки, і якщо скласти ці переваги та виключити недоліки, то можна отримати зручний продукт для учнів та викладачів. Складемо таблицю порівняння аналогів, де "+" - наявність, а "-" - відсутність.

Таблиця 1. – Порівняння аналогів з GPM

Особливості	**“GPM”**	**Access**	**Excel**	**Notepad++**
Використовується БД	+	+	-	-
Безпечний доступ	+	+	-	-
Зручність використання	+	-	+	+
User Friendly	+	-	-	+
Фільтр інформації	+	+	+	-

2. ВИМОГИ ТА ОЦІНКА ЯКОСТІ СИСТЕМИ

2.1. Функціональні вимоги

2.1.1. Авторизація

Для користування застосунком, користувач повинен мати змогу авторизуватись у застосунку. За створення аккаунтів користувачів повинен відповідати адмін який матиме змогу створювати, оновлювати та видаляти користувачів. Для створення користувача портібно буде указати логін та пароль.

2.1.2 Матеріали

В адміна повинна бути можливість створювати матеріали які далі будуть додаватися до замовлень. Матеріали матимуть три параметри:

- Назва матеріалу
- Міра обчислення
- Кількість

2.1.3. Замовлення

Користувач повинен мати можливість створювати, оновлювати, переглядати та видаляти замовлення. Користувач матиме змогу переглядати тільки свої замовлення язі прив’язані до нього по айді. Повинна бути можливість пошуку замовлень по фільтру параметрів. Адмін повинен мати можливість переглядати усі замовлення, і також створювати, оновлювати та видаляти їх. Замовлення мають мати наступні поля:

- Номер замовлення(автогенерований)
- Опис замовлення
- Список матеріалів

- Дата отримання замовлення
- Термін здачі замовлення

3. Проектування та реалізація

3.1 Інструменти для розробки

3.1.1 Visual studio

Visual studio це найпопулярніша IDE яка використовується для написання керованого коду, який підтримується Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight. Редактор коду Visual Studio .NET підтримує IntelliSense і рефакторинг коду, налагодження як на рівні джерела, так і на рівні машини.

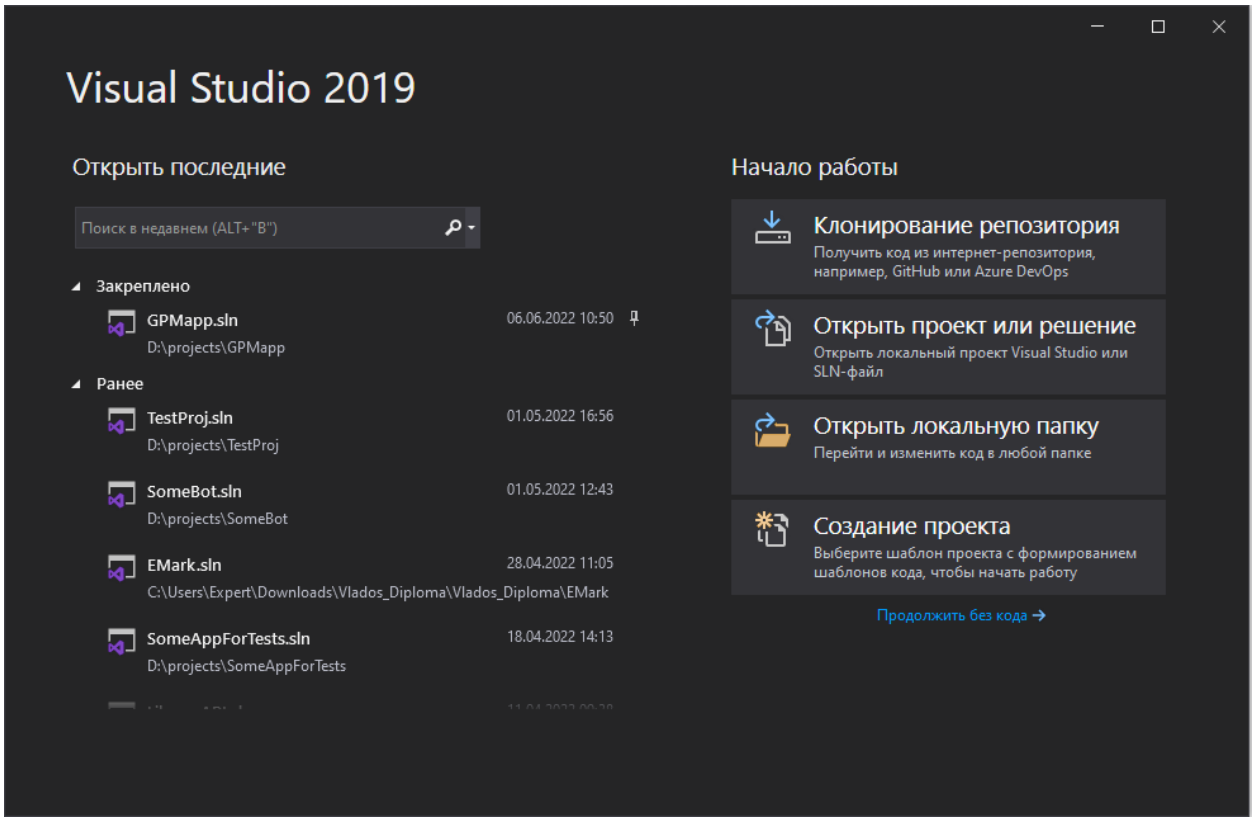


Рис. 1 – Домашній інтерфейс Visual studio

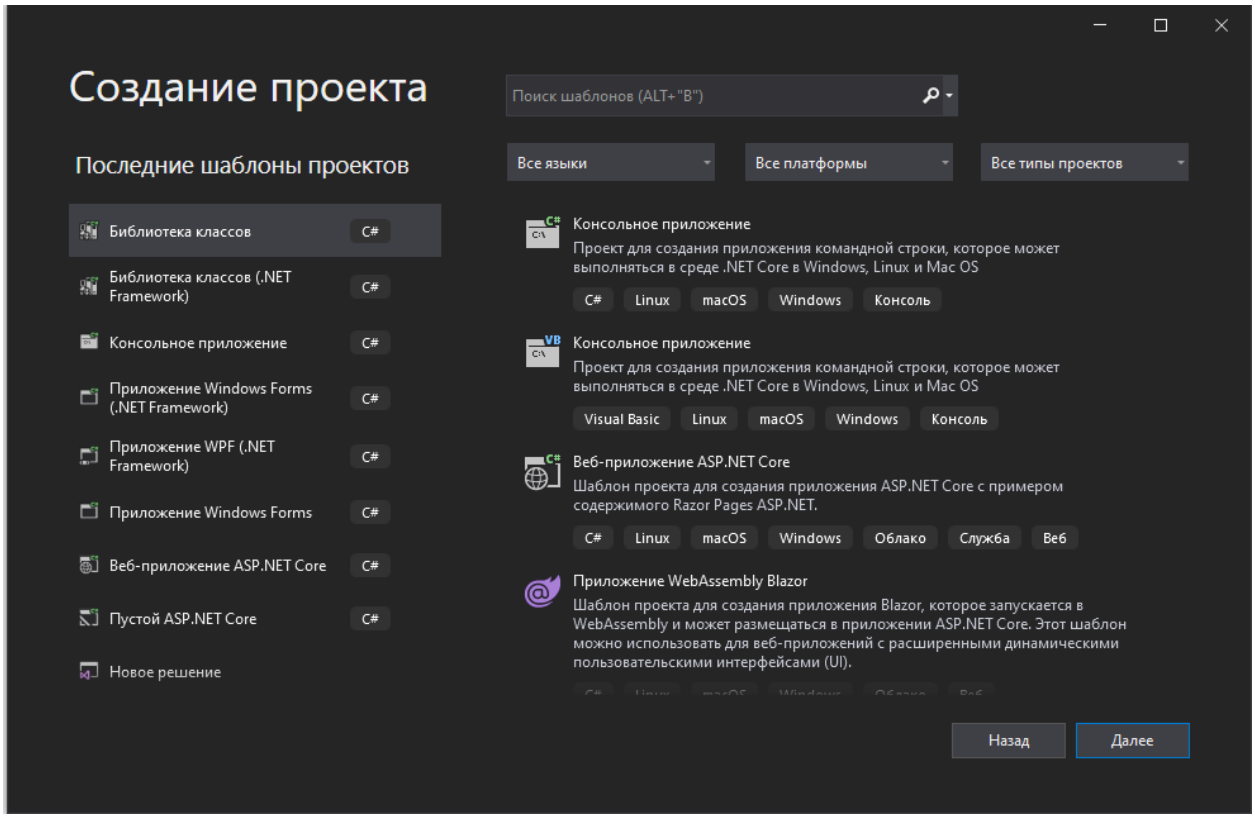


Рис. 2 – Інтерфейс створення проекту Visual studio

Варто зауважити що Visual studio має інтегровану систему контролю версій Git.

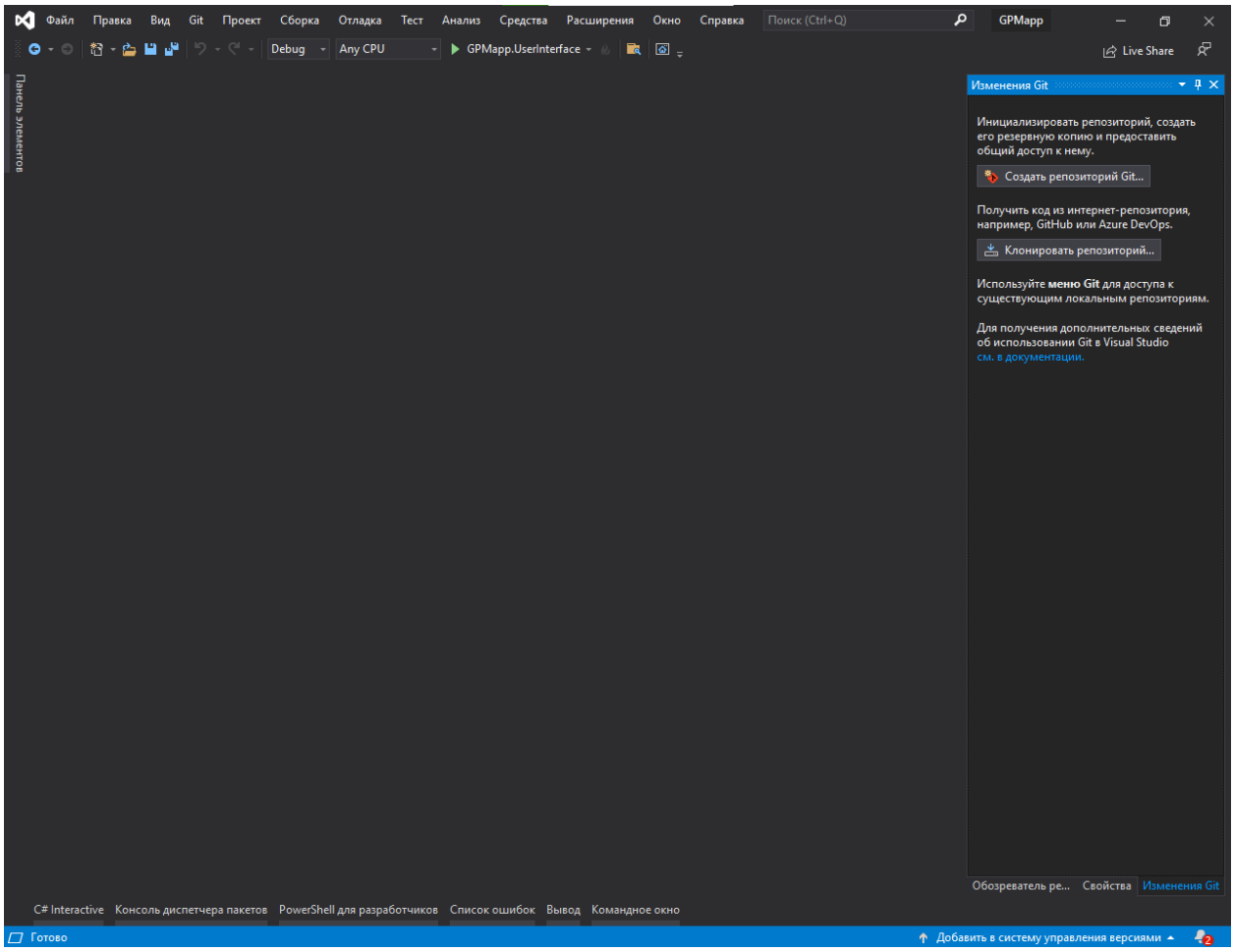


Рис. 3 – Інтерфейс проекту Visual studio

Також Visual studio має інтегровану систему управління пакетами NuGet.

3.1.2 Git

Git це найпопулярніша й найзручніша система управління версіями. Контроль версій є найбільш необхідною функцією, яка потрібна під час роботи в середовищі. GIT є найпопулярнішою системою керування версіями, яка є відкритим вихідним кодом, безкоштовна у використанні, працює на основі сценаріїв оболонки на основі Linux і легко інтегрується з будь-якими типами IDE.

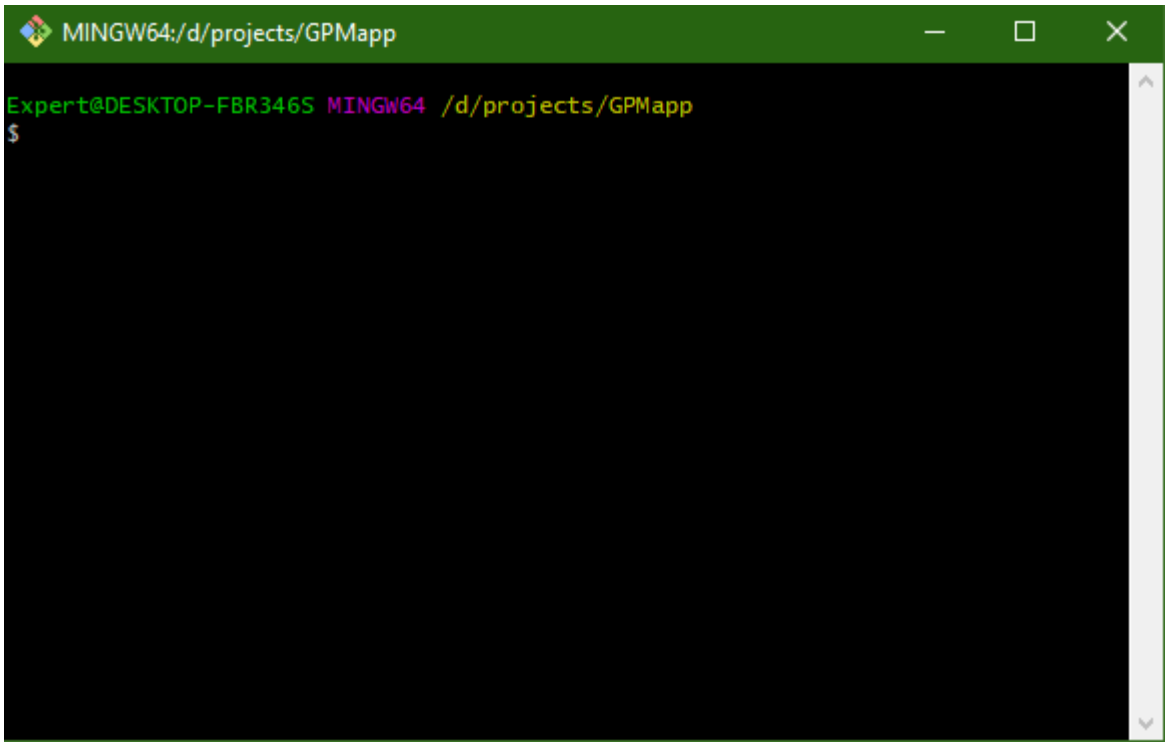


Рис. 4 – Консоль Git

Також Git зберігає проект у веб-сервісі GitHub.

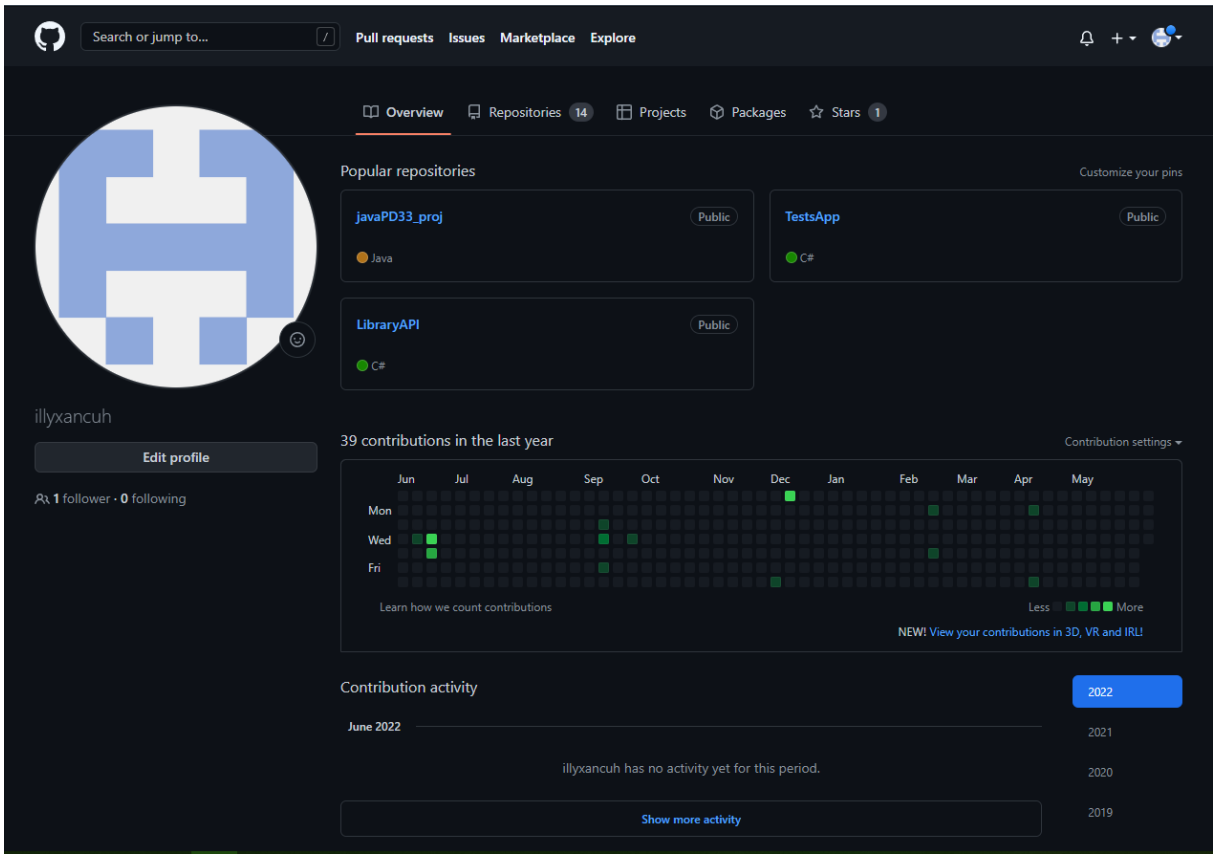


Рис. 5 – Веб-сервіс Github

3.1.3 MS SQL Server

MS SQL Server система управління реляційною базою даних, розроблена Microsoft. Microsoft. SQL Server — це система управління та аналізу баз даних для рішень для електронної комерції, бізнесу та сховищ даних.

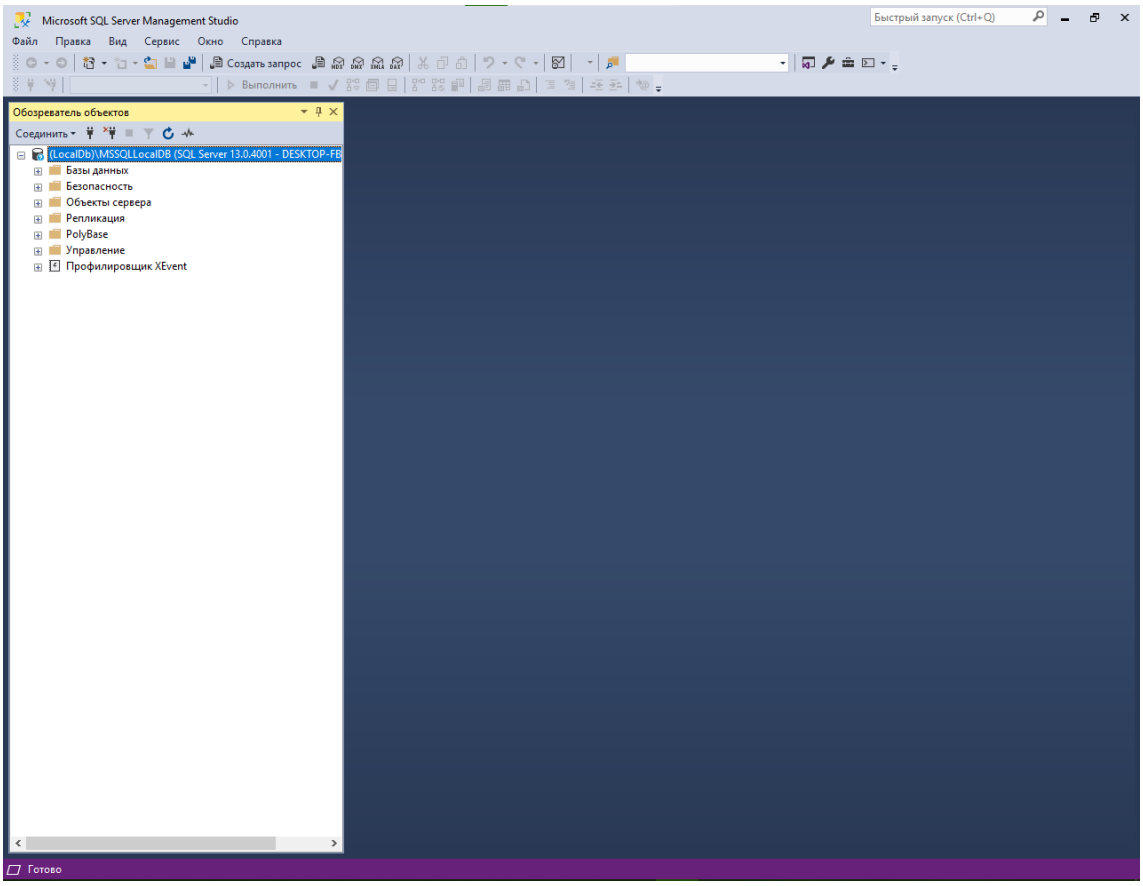


Рис. 7 – Інтерфейс Microsoft SQL Server Management Studio

Для користування MS SQL Server використовується Microsoft SQL Server Management Studio.

3.2 Архітектура системи

При проектуванні було використано монолітну архітектуру, завдяки їй:

- Легше розвивати.

  Робота з одним виконуваним файлом проста. Отже, для простих застосувань або початку проекту розробки монолітна архітектура легше. Однак у міру розвитку та виникнення складнощів монолітні середовища можуть стати недоліком.

- Легше перевірити.

  Через характер програми ви можете просто запустити програму та протестувати інтерфейс користувача за допомогою певного інструменту. Використання одного виконуваного файлу означає, що вам потрібно налаштувати лише одну програму для реєстрації, моніторингу та тестування.

- Легше розгорнути.

  При роботі з одним виконуваним файлом складності набагато менше. Для розгортання в інших системах вам потрібно скопіювати запакований додаток на інший сервер і запустити його.

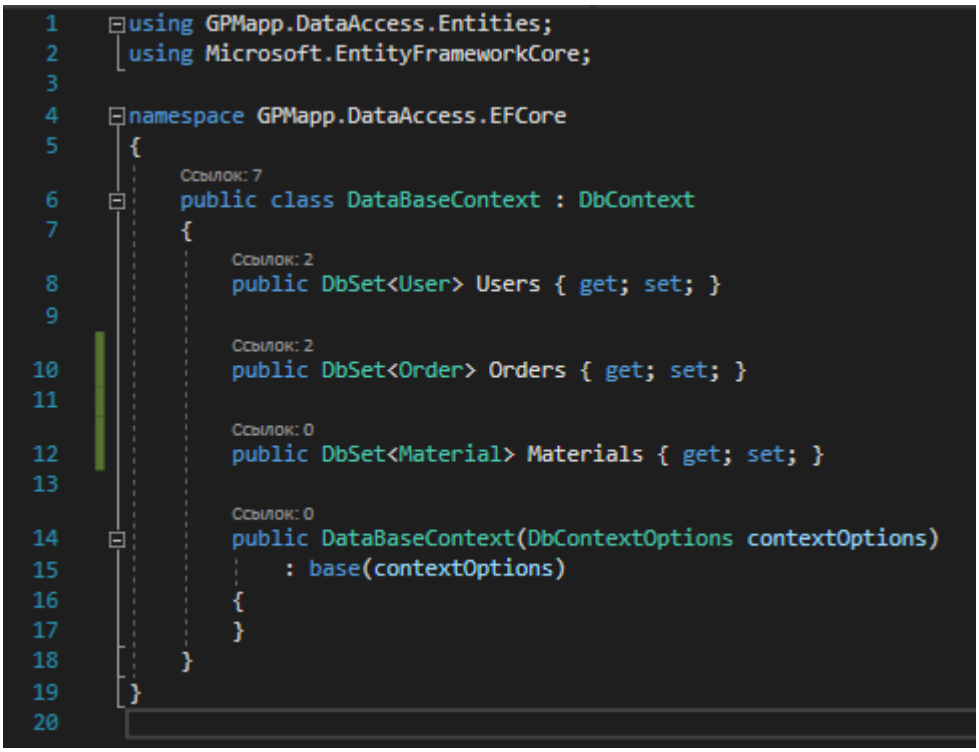
- Менш складні та менші накладні витрати.

  Мікросервіси можуть швидко ускладнюватися. Однак у монолітній архітектурі ви можете уникнути додаткових витрат, пов’язаних із міжсервісним зв’язком, виявленням та реєстрацією служб, балансуванням навантаження, розподіленим журналюванням, розподіленим моніторингом і керуванням продуктивністю, а також керуванням даними.

3.2.1 Entity Framework

Entity Framework - це платформа ORM (Object-Relational Mapper) для доступу до даних у .Net. Він був випущений разом з .NET Core і є розширюваною, легкою, з відкритим вихідним кодом і кросплатформною версією технології доступу до даних Entity Framework. Він працює на кількох операційних системах, таких як Windows, Mac і Linus.

Термін ORM означає Object-Relational Mapper і автоматично створює класи на основі таблиць бази даних, і навпаки. Тобто він також може генерувати необхідний SQL для створення бази даних на основі класів.



```
using GPMapp.DataAccess.Entities;
using Microsoft.EntityFrameworkCore;

namespace GPMapp.DataAccess.EFCore
{
    Ссылок: 7
    public class DataBaseContext : DbContext
    {
        Ссылок: 2
        public DbSet<User> Users { get; set; }

        Ссылок: 2
        public DbSet<Order> Orders { get; set; }

        Ссылок: 0
        public DbSet<Material> Materials { get; set; }

        Ссылок: 0
        public DataBaseContext(DbContextOptions contextOptions)
            : base(contextOptions)
        {
        }
    }
}
```

Рис. 6 – DataBaseContext

Для зв’язку проекту з Entity framework створюється DataBaseContext.

3.2.2 LINQ

Повною формою LINQ є Language-Integrated Query(Інтегрований мовний запит) і представлений у .NET Framework 3.5 для запиту даних з різних джерел даних, таких як колекції, загальні засоби, документи XML, набори даних ADO.NET, SQL, веб-сервіси тощо в C# і VB.NET. LINQ надає багатий стандартизований синтаксис запитів на мові програмування .NET, такій як C# і VB.NET, що дозволяє розробникам взаємодіяти з будь-якими джерелами даних.

У C# або VB.NET функціональність LINQ може бути досягнута, імпортуючи простір імен System.Linq в нашу програму. Як правило, LINQ містить набір методів розширення, які дозволяють нам запитувати джерело об’єкта даних безпосередньо в нашому коді на основі вимоги.

LINQ є простішим, упорядкованим і високорівневим, ніж SQL. Коли ми хочемо використовувати базу даних запитів, у більшості випадків LINQ є більш продуктивною мовою запитів, ніж SQL. Крім того, ми маємо переваги IntelliSense, оскільки запит LINQ написаний за кодом. LINQ має повну перевірку типів під час компіляції, щоб ми могли вловити будь-яку помилку під час компіляції. У C# або VB.Net писати запит у LINQ веселіше.

```
_databaseContext.Group.AsNoTracking().Include(group => group.GroupUsers)
.Where(group => group.TeacherId == id || group.GroupUsers.Any(groupUser => groupUser.UserId == id)).ToArrayAsync();
```

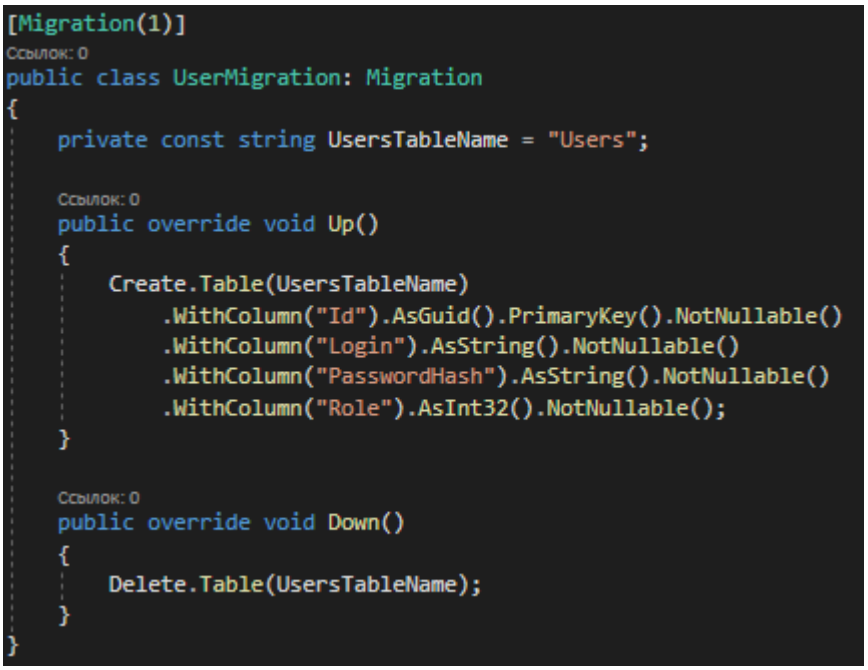
Рис. 6 – Приклад LINQ запиту

3.2.3 Міграції

Міграція бази даних відноситься до переміщення даних з однієї бази даних в іншу. Існує багато причин, чому підприємство може вирішити перейти з існуючої бази даних, наприклад, для досягнення масштабованості, економії грошей, підвищення надійності або досягнення іншої важливої мети. Хоча міграція баз даних може надати зростаючому бізнесу незліченну кількість переваг, процес може варіюватися від простого до складного.

Деякі форми міграції бази даних просто передбачають переміщення даних з одного екземпляра бази даних до іншого такого ж типу. Наприклад, компанія може передавати дані з бази даних MySQL на одному сервері в іншу базу даних MySQL на іншому сервері. Більш складні форми можуть включати кілька типів баз даних.

Для міграцій було використано Fluent migrator який спрощує процес міграцій.



```
[Migration(1)]
Ссылок: 0
public class UserMigration: Migration
{
    private const string UsersTableName = "Users";

    Ссылок: 0
    public override void Up()
    {
        Create.Table(UsersTableName)
            .WithColumn("Id").AsGuid().PrimaryKey().NotNullable()
            .WithColumn("Login").AsString().NotNullable()
            .WithColumn("PasswordHash").AsString().NotNullable()
            .WithColumn("Role").AsInt32().NotNullable();
    }

    Ссылок: 0
    public override void Down()
    {
        Delete.Table(UsersTableName);
    }
}
```

Рис. 7 – Приклад міграції

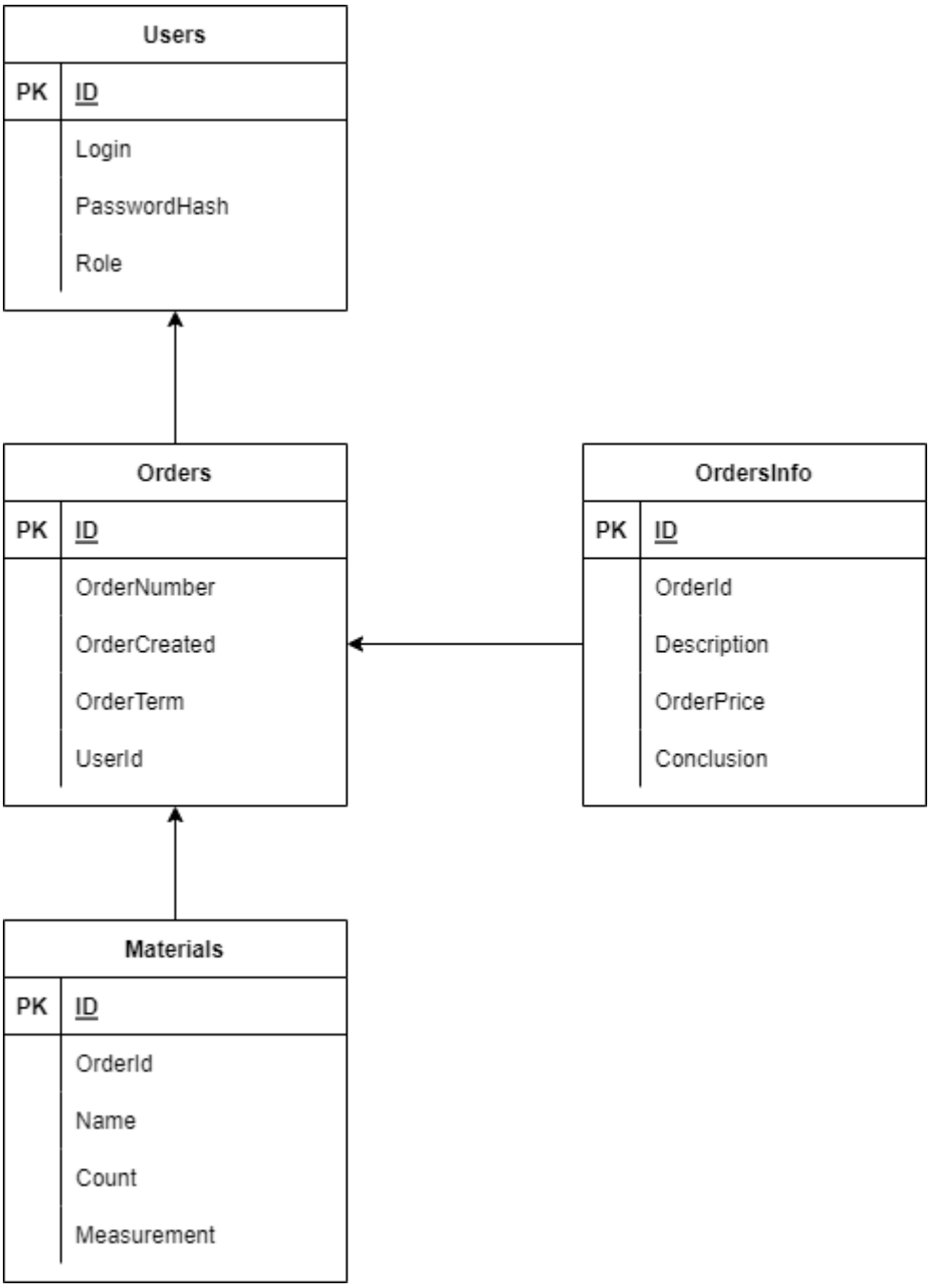


Рис. 8 – ER діаграма для БД

3.2.4 ООП

Об’єктно-орієнтоване програмування - це використання класу мов програмування та методів, заснованих на концепції «об’єкта», який є структурою даних, інкапсульованою з набором підпрограм, які називаються «методами», які оперувати даними. Операції з даними можна виконувати лише за допомогою цих методів, які є загальними для всіх об’єктів, які є екземплярами певного «класу».

Таким чином, інтерфейс до об’єктів добре визначений і дозволяє змінювати код, що реалізує методи, доки інтерфейс залишається незмінним.

3.2.5 SOLID

SOLID - це абревіатура перших п'яти принципів об'єктно-орієнтованого проектування.

- SRP(Single Responsibility Principle): Принцип єдиної відповідальності. Модуль повинен відповідати лише одному актору.

Клас має бути відповідальним лише за щось одне. Якщо клас відповідає за вирішення кількох завдань, його підсистеми, що реалізують розв'язання цих завдань, виявляються пов'язаними один з одним. Зміни в одній такій підсистемі ведуть до змін в іншій.

- OCP(Open-Closed Principle): Відкритий закритий принцип. Артефакт програмного забезпечення має бути відкритим для розширення, але закритим для модифікації.

Наслідування принципу OCP полягає в тому, що програмне забезпечення змінюється не через зміну існуючого коду, а через додавання нового коду. Тобто створений спочатку код залишається «недоторканим» і стабільним, а нова функціональність впроваджується через спадкування реалізації, або через використання абстрактних інтерфейсів і поліморфізм.

- LSP(Liskov Substitution Principle): Принцип заміщення Ліскова

Ціль цього принципу полягає в тому, щоб класи-спадкоємці могли б використовуватися замість батьківських класів, від яких вони утворені, не порушуючи роботу програми. Якщо виявляється, що в коді перевіряється тип класу, то принцип підстановки порушується.

- ISP(Interface Segregation Principle): Принцип сегрегації інтерфейсу.

Створюйте чіткі інтерфейси, які залежать від клієнта.

- DIP(Dependency Inversion Principle): Принцип інверсії залежностей.

Залежіть від абстракцій, а не від конкреций.

3.2.6 Принцип Kiss

Принцип «нехай це буде просто дурним» (KISS) — це правило проектування, яке стверджує, що системи працюють найкраще, коли вони мають прості проекти, а не складні. KISS не означає дурість. Навпаки, зазвичай це асоціюється з інтелектуальними системами, які через їх спрощену конструкцію можуть бути неправильно витлумачені як дурні. Принцип KISS перешкоджає та запобігає повзучому характеру, відмову системи та іншим проблемам.

3.2.7 YAGNI

YAGNI - You Ain't Gonna Need It (вам це не знадобиться), це екстремальний принцип програмування, який стверджує, що ви не повинні впроваджувати функції, які ви вважаєте важливими в майбутньому. Цей принцип допомагає зосередитися на необхідних функціях і не додавати непотрібні функції.

3.2.8 DRY

Дублювання коду може ускладнити обслуговування коду. Будь-яка зміна логіки може зробити код схильним до помилок або ускладнити зміну коду. Це можна виправити за допомогою повторного використання коду (принцип DRY).

Принцип DRY сформульовано так: «Кожна частина знання повинна мати єдине, однозначне, авторитетне уявлення в системі».

Спосіб досягнення DRY полягає в створенні функцій і класів, щоб переконатися, що будь-яка логіка повинна бути записана лише в одному місці.

3.2.9 Unit Of Work

Unit of Work — це шаблон проектування, який зменшує повторюваність коду під час реалізації управління транзакціями та накладні витрати на кодування, пов'язані з дотриманням групування DML за рахунок широкого використання карт і списків. Це не обов'язкова вимога для впровадження рівня сервісу, але це може допомогти.

```
using GPMapp.DataAccess.Contracts;
using GPMapp.DataAccess.EFCore;
using GPMapp.DataAccess.Repositories;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GPMapp.DataAccess
{
    ссылка: 1
    public class UnitOfWork : IUnitOfWork
    {
        public readonly DataBaseContext _dataBaseContext;
        private IUserRepository _users;
        private IOrderRepository _orders;

        Ссылок: 0
        public UnitOfWork(DataBaseContext dataBaseContext)
        {
            _dataBaseContext = dataBaseContext;
        }

        ссылка: 1
        public IUserRepository Users
        {
            get
            {
                if (_users == null)
                {
                    _users = new UserRepository(_dataBaseContext);
                }

                return _users;
            }
        }

        ссылка: 1
        public IOrderRepository Orders
        {
            get
            {
                if (_orders == null)
                {
                    _orders = new OrderRepository(_dataBaseContext);
                }

                return _orders;
            }
        }
    }
}
```

Рис. 9 – Приклад використання патерну UnitOfWork

Патерн repository — це шаблон проектування, який передає дані з шарів домену та доступу до даних (наприклад, Entity Framework Core / Dapper). Репозиторії - це класи, які приховують логіку, необхідну для зберігання або отримання даних. Таким чином, наша програма не буде дбати про те, який тип ORM ми використовуємо, оскільки все, що стосується ORM, обробляється в межах рівня сховища. Це дозволяє чіткіше розділити проблеми. Шаблон репозиторію є одним із широко використовуваних шаблонів проектування для створення більш чистих рішень.

```
using GPMapp.DataAccess.Entities;
using GPMapp.DataAccess.Contracts;
using GPMapp.DataAccess.EFCore;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using GPMapp.Shared.Exceptions;

namespace GPMapp.DataAccess.Repositories
{
    Ссылок: 2
    public class UserRepository : RepositoryBase<User>, IUserRepository
    {
        ссылка: 1
        public UserRepository(DataBaseContext dataBaseContext)
            :base(dataBaseContext)
        {
        }

        ссылка: 1
        public User GetUserById(int id)
        {
            return _dataBaseContext.Users
                .AsNoTracking()
                .Include(user => user.Role)
                .SingleOrDefault(user => user.Id == id)
                ?? throw new NotFoundException($"User with id {id} doesn't exist.");
        }

        ссылка: 1
        public User GetUserByLogin(string login)
        {
            return _dataBaseContext.Users
                .AsNoTracking()
                .Include(user => user.Role)
                .SingleOrDefault(user => user.Login == login)
                ?? throw new NotFoundException($"User with login {login} doesn't exist.");
        }
    }
}
```

Рис. 10 – Приклад використання патерну Repository

3.2.10 Iterator

Ітератор — це поведінковий шаблон проектування, який дозволяє вам обходити елементи колекції, не відкриваючи її базове представлення (список, стек, дерево тощо).

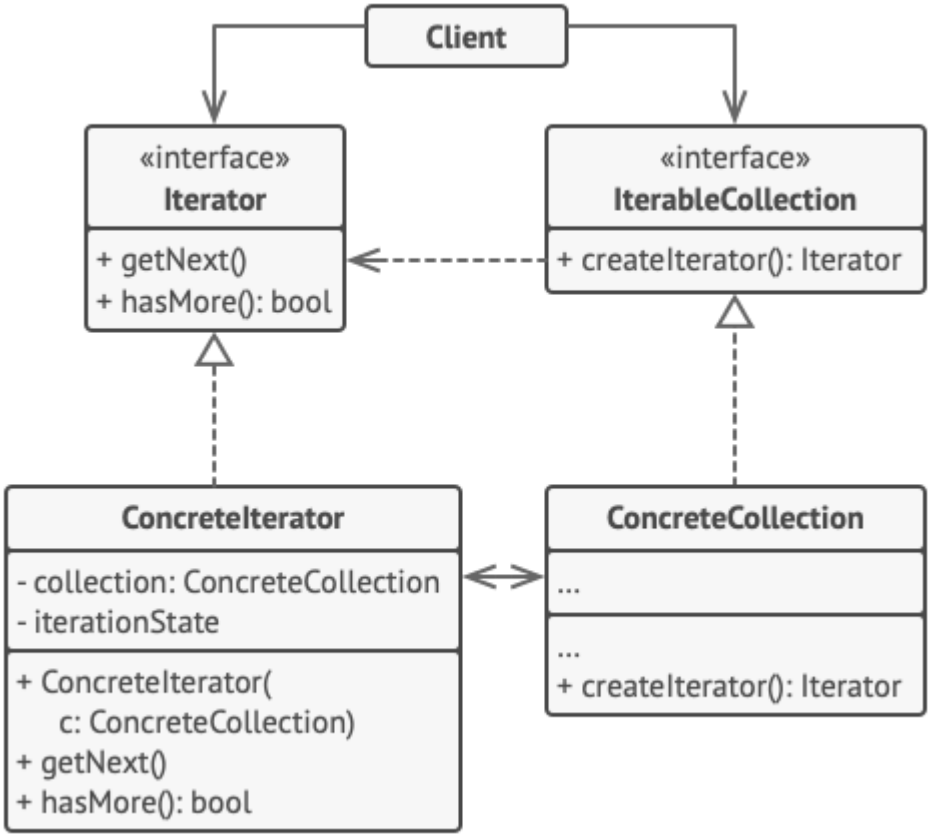


Рис. 11 – Структура патерну Iterator

Ітератор інкапсулює деталі роботи зі складною структурою даних, надаючи клієнту кілька простих методів доступу до елементів колекції. Хоча цей підхід дуже зручний для клієнта, він також захищає колекцію від необережних або зловмисних дій, які клієнт міг би виконати, працюючи безпосередньо з колекцією.

Шаблон надає кілька загальних інтерфейсів як для колекцій, так і для ітераторів. Враховуючи, що код тепер використовує ці інтерфейси, він все одно

працюватиме, якщо передати йому різні види колекцій та ітераторів, які реалізують ці інтерфейси.

3.2.11 Command

Command — це поведінковий шаблон проектування, який перетворює запит на окремий об’єкт, який містить всю інформацію про запит. Це перетворення дозволяє передавати запити як аргументи методу, відкладати чи ставити в чергу виконання запиту, а також підтримувати операції, які неможливо виконати.

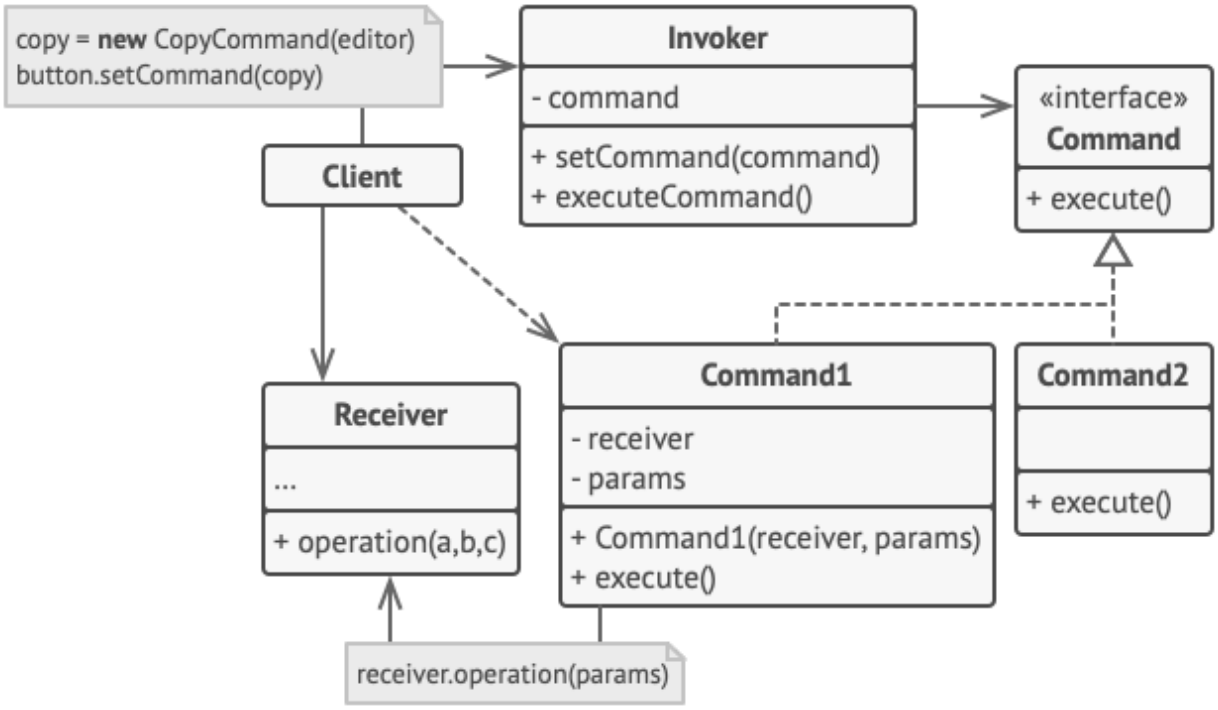


Рис. 12 – Структура патерну Command

Шаблон Command може перетворити певний виклик методу в окремий об’єкт. Ця зміна відкриває багато цікавих застосувань: ви можете передавати команди як аргументи методу, зберігати їх всередині інших об’єктів, перемикати пов’язані команди під час виконання тощо.

Як і будь-який інший об’єкт, команду можна серіалізувати, що означає перетворення її в рядок, який можна легко записати у файл або базу даних. Пізніше рядок можна відновити як початковий командний об’єкт. Таким чином, ви можете відкласти та запланувати виконання команди. Але є ще більше! Таким же чином ви можете тримати в черзі, реєструвати або надсилати команди через мережу.

3.2.12 Strategy

Стратегія – це поведінковий шаблон проектування, який дозволяє визначити сімейство алгоритмів, помістити кожен з них в окремий клас і зробити їх об’єкти взаємозамінними.

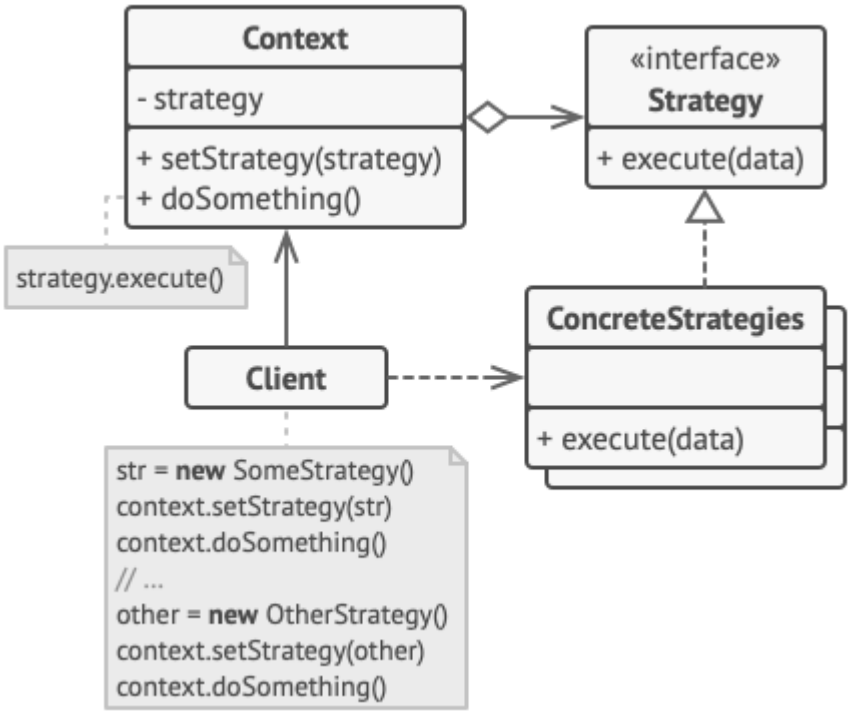


Рис. 13 – Структура патерну Strategy

3.2.13 Windows forms

Windows Forms — це структура інтерфейсу користувача для створення настільних програм Windows. Він забезпечує один з найпродуктивніших способів створення настільних програм на основі візуального дизайнера, наданого у Visual Studio. Такі функціональні можливості, як розміщення візуальних елементів керування перетягуванням, дозволяють легко створювати настільні програми.

За допомогою Windows Forms ведеться розробка графічних програм, які легко розгортати, оновлювати та працювати в автономному режимі або підключені до Інтернету. Програми Windows Forms можуть отримати доступ до локального обладнання та файлової системи комп'ютера, на якому запущена програма.

Windows Forms — це технологія інтерфейсу користувача для .NET, набору керованих бібліотек, які спрощують звичайні завдання програми, такі як читання та запис у файлову систему. Коли вкиористовується середовище розробки Visual Studio, ви можете створювати інтелектуальні клієнтські програми Windows Forms, які відображають інформацію, запитують введення від користувачів і спілкуються з віддаленими комп'ютерами через мережу.

У Windows Forms форма — це візуальна поверхня, на якій відображається інформація для користувача. Зазвичай ви створюєте програми Windows Forms, додаючи елементи керування до форм і розробляючи відповіді на дії користувача, наприклад клацання мишею або натискання клавіш. Елемент керування — це дискретний елемент інтерфейсу користувача, який відображає дані або приймає введення даних.

Коли користувач щось робить з формою або одним із елементів керування, ця дія створює подію. Додаток реагує на ці події за допомогою коду та обробляє події, коли вони відбуваються.

Windows Forms містить різноманітні елементи керування, які можна додати до форм: елементи керування, які відображають текстові поля, кнопки, спадні

вікна, перемикач і навіть веб-сторінки. Якщо наявний елемент керування не відповідає вашим потребам, Windows Forms також підтримує створення власних спеціальних елементів керування за допомогою класу UserControl.

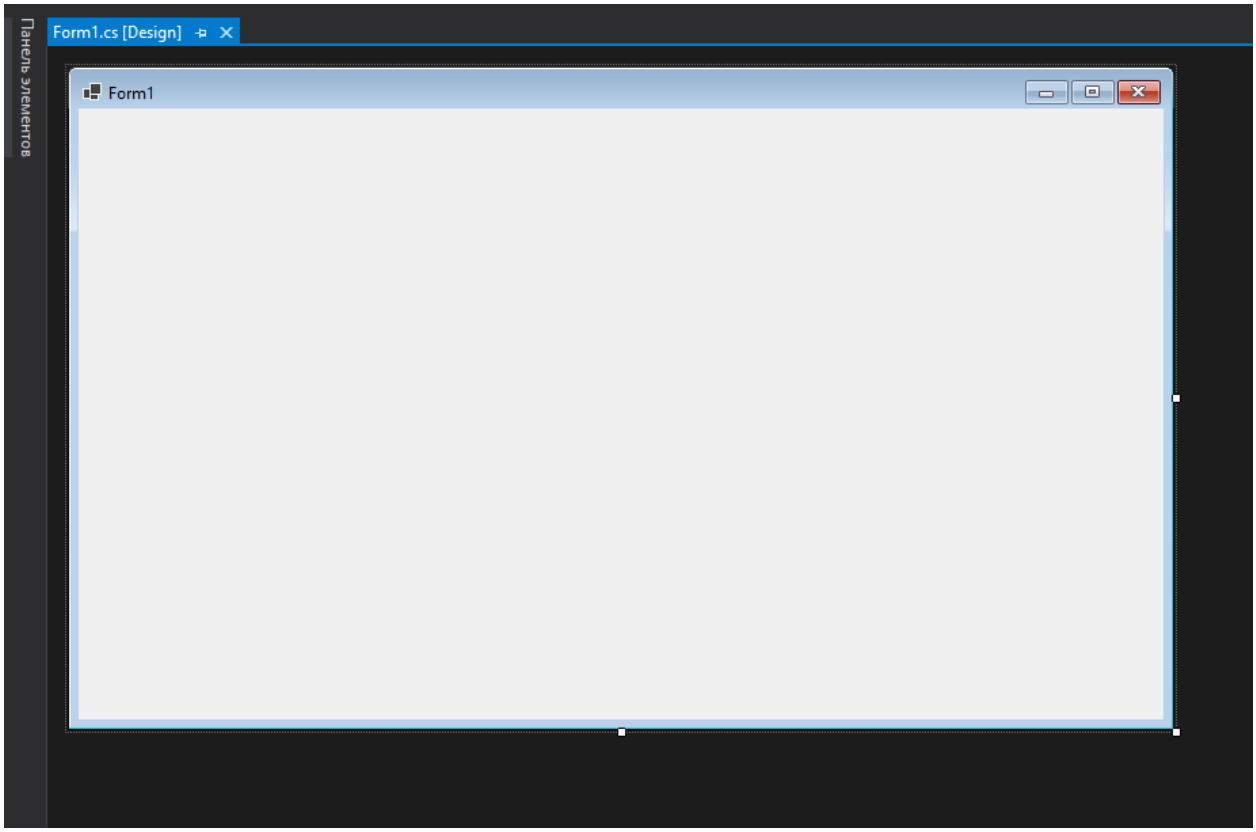


Рис. 14 – Вікно редактора форм Windows forms

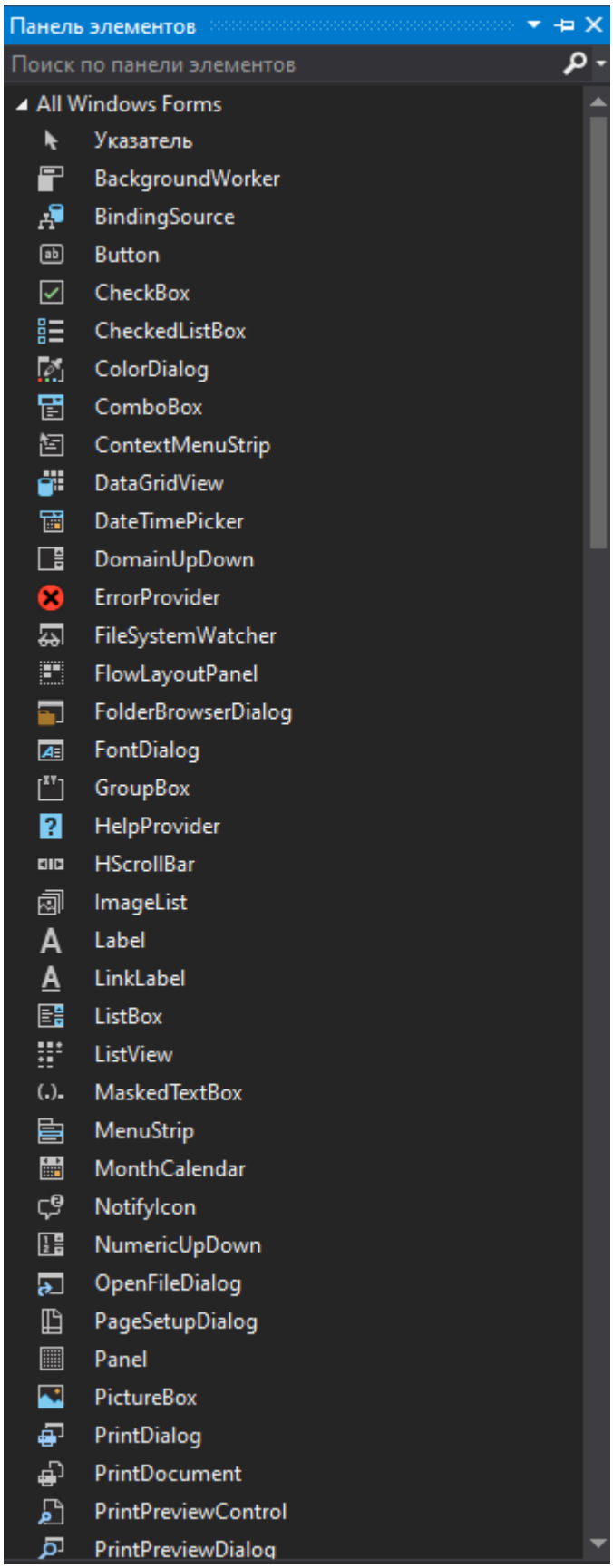


Рис. 15 – Панель елементів для форм Windows forms

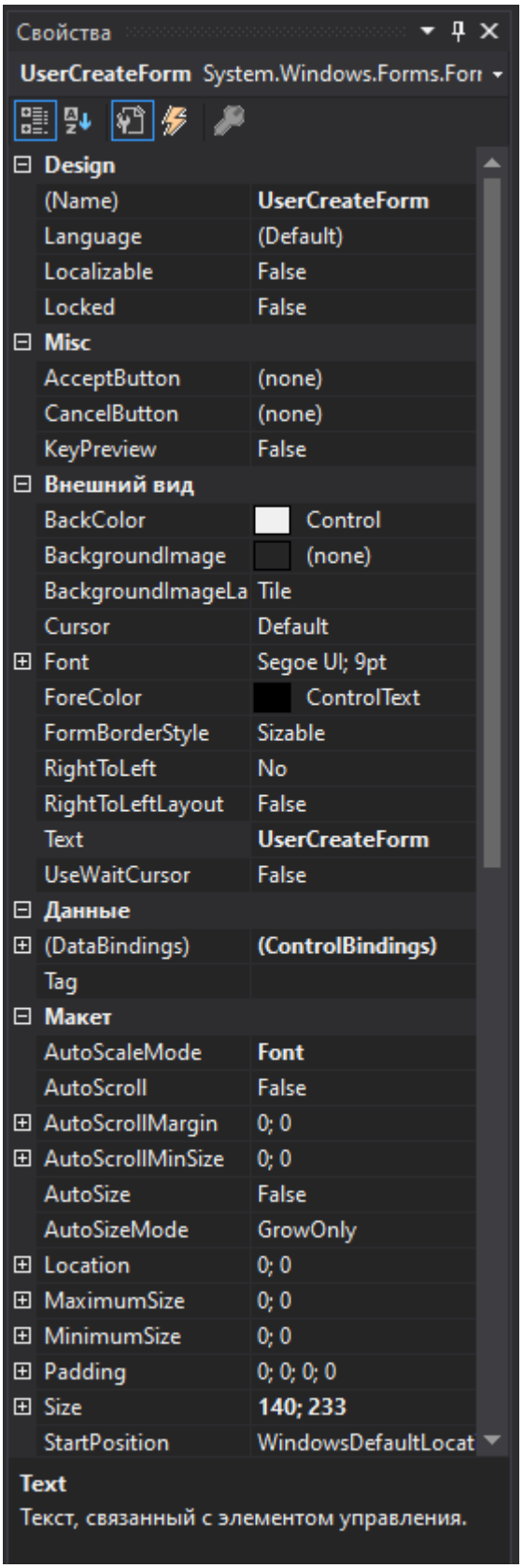


Рис. 16 – Властивості та налаштування в Windows forms

3.2.14 Dependency injection

Dependency injection (Ін’єкція залежності) - це шаблон проектування, який дозволяє об’єкту А оголошувати свої залежності від зовнішнього об’єкта, який надає ці залежності. Залежності, оголошені, зазвичай є інтерфейсами класів, а залежності, які надає, є конкретними реалізаціями для цих інтерфейсів.

Це допускає слабке з’єднання коду, оскільки об’єкт більше не потребує ініціалізації власних залежностей. Об’єкт вирішує, які реалізації надати об’єкту А на основі конфігурації або бажаної поведінки.

```
static IContainer ConfigureDependencyInjection()
{
    var builder = new ContainerBuilder();
    builder.RegisterType<WorkerRepository>().As<IWorkerRepository>();
    builder.RegisterType<WorkerService>().As<IWorkerService>();
    builder.RegisterType<WorkerDataGridController>();
    builder.RegisterAutoMapper(typeof(WorkerProfile).Assembly);
    builder.RegisterType<Form1>();
    return builder.Build();
}
```

Рис. 17 – Приклад використання паттерну Dependency injection

3.3 Налаштування проекту

3.3.1 Layers

Перш за все проект було поділено на шари кожен з яких відповідає за своє. Це потрібно для зменшення залежності та збільшення безпеки проекту.

Проект було поділено на наступні шари: GPMapp.BussinessLogic – шар який відповідає за логіку проекту. Він

містить увесь функціонал проекту, а також з’єднує шари GPMapp.DataAccess та GPMapp.UserInterface

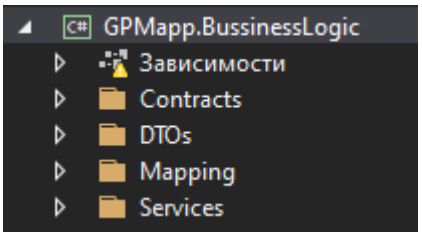


Рис. 18 – Шар BussinessLogic

GPMapp.DataAccess – шар який відповідає за дії з базою данних. Він містить у собі контекст для зв’язку з базою, репозиторії, контракти до репозиторіїв, ентіті та клас патерну UnitOfWork.

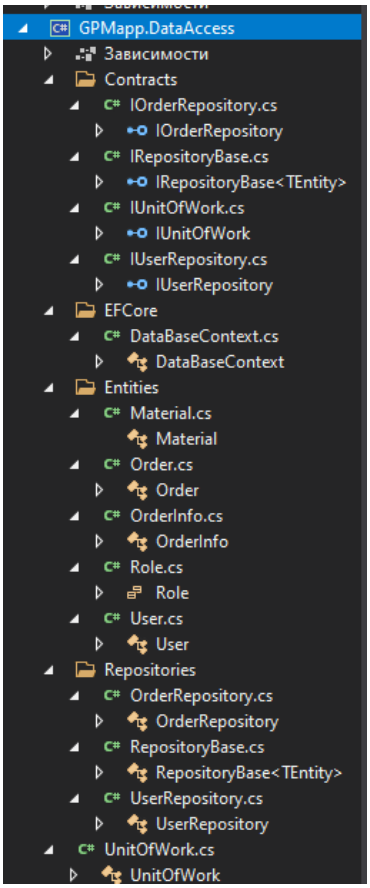


Рис. 19 – Шар DataAccess

GPMapp.DataAccess.Migrations – шар який відповідає за міграції до бази данних.

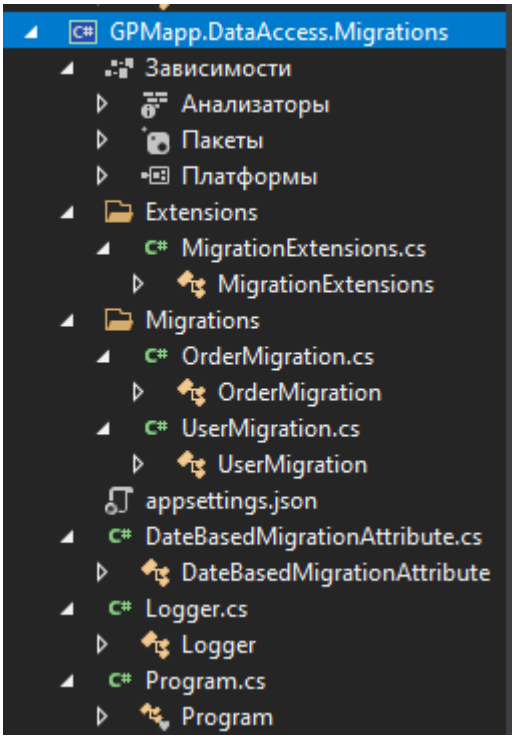


Рис. 20 – Шар DataAccess.Migrations

GPMapp.UserInterface – шар який відповідає за UI проекту.

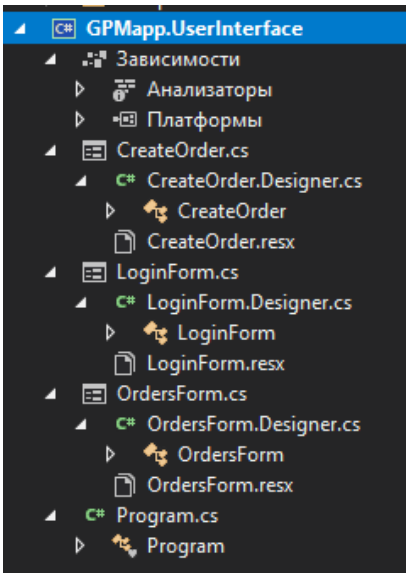


Рис. 21 – Шар UserInterface

3.3.2 Шари DataAccess та DataAccess.Migrations

Для початку було спроектовано діаграмму классів на якій буде видно класи та їх відносини між ними.

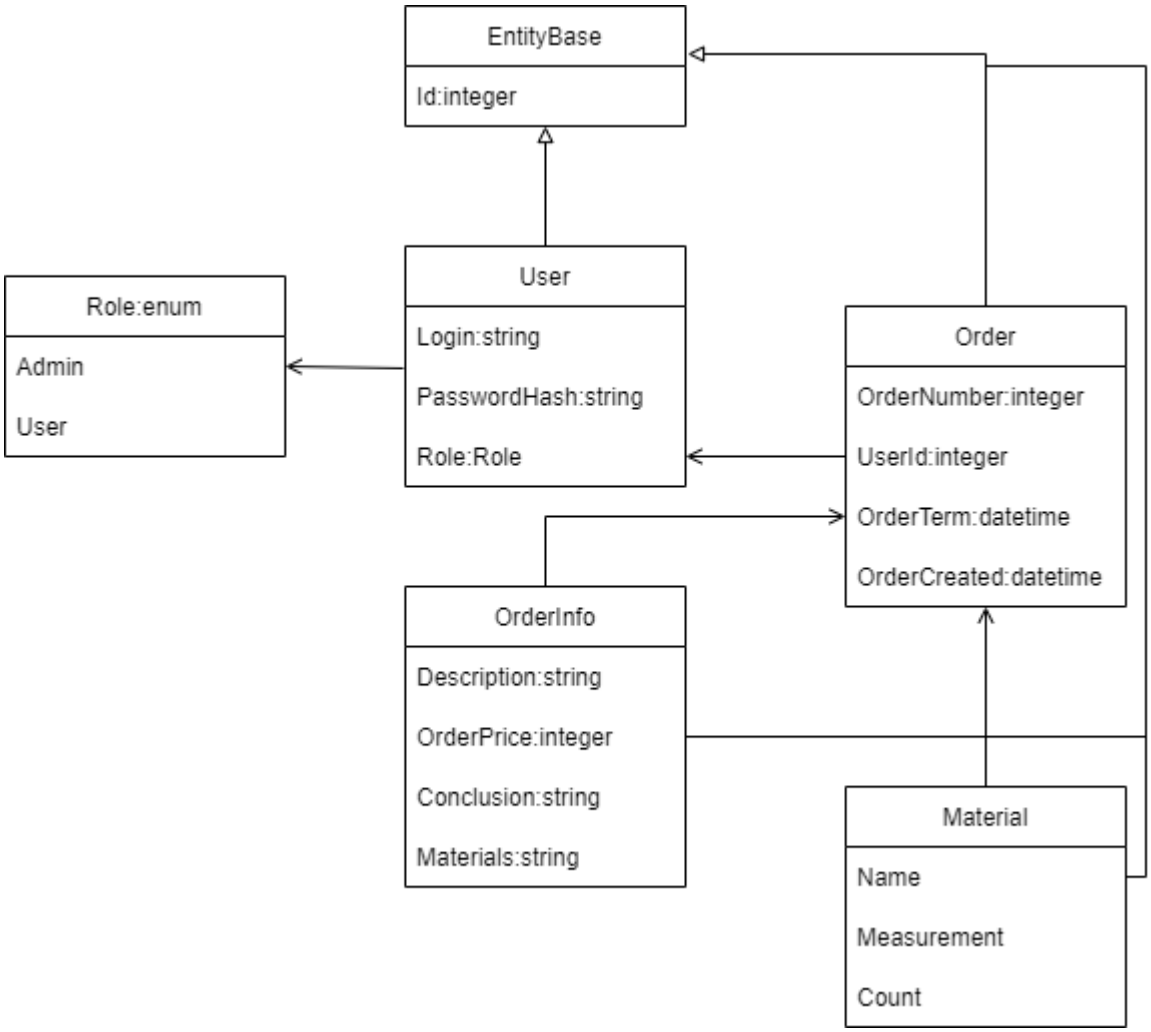


Рис. 22 – Діаграма класів

Після цього було створено ентіті у шарі DataAccess.

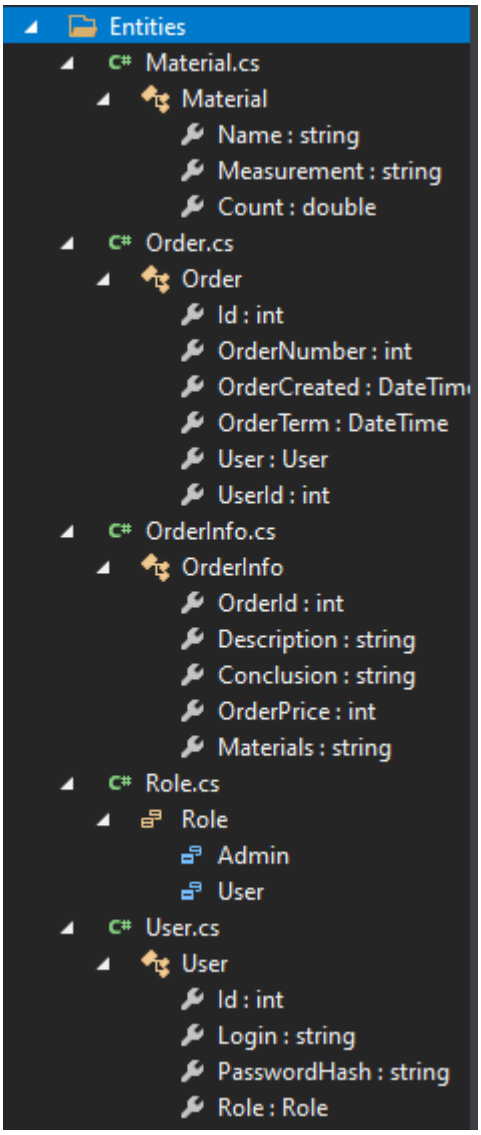


Рис. 23 – Ентіті

Після ентіті було створенно міграції та налаштовано міграції.

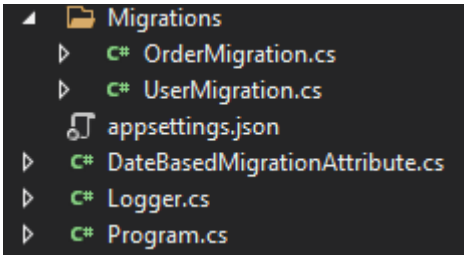


Рис. 24 – Міграції та їх налаштування

```
using System;

namespace TWMapp.DataAccess.Migrations
{
    Ссылок: 10
    public static class Logger
    {
        Ссылок: 5
        public static void Information(string text)
        {
            Console.ForegroundColor = ConsoleColor.White;
            Console.WriteLine(text);
        }

        Ссылок: 2
        public static void Error(string text, Exception exception = null)
        {
            Console.ForegroundColor = ConsoleColor.Red;

            if (exception == null)
            {
                Console.WriteLine(text);
            }
            else
            {
                Console.WriteLine(text + "\nException: " + exception.Message);
            }
        }

        Ссылок: 3
        public static void Warning(string text)
        {
            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.WriteLine(text);
        }

        Ссылок: 0
        public static void Success(string text)
        {
            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine(text);
        }

        Ссылок: 0
        public static void Other(string text, ConsoleColor color)
        {
            Console.ForegroundColor = color;
            Console.WriteLine(text);
        }
    }
}
```

Рис. 25 – Логер міграцій

```
private static int Main(string[] args)
{
    try
    {
        Logger.Information("Starting db migrator");

        IServiceProvider serviceProvider = CreateServices();

        // Put the database update into a scope to ensure
        // that all resources will be disposed.
        using IServiceScope scope = serviceProvider.CreateScope();

        MigrationDirection migrationDirection = args.Length > 0
            ? Enum.Parse<MigrationDirection>(args[0])
            : MigrationDirection.Up;

        RetryPolicy retryPolicy = Policy
                  .Handle<NpgsqlException>()
                  .WaitAndRetry(5,
                                i =>
                                {
                                    TimeSpan sleepDuration = (i * 3).Seconds();
                                    Logger.Warning($"Wait for {sleepDuration.Humanize()} and retry");
                                    return sleepDuration;
                                });

        IServiceProvider scopeServiceProvider = scope.ServiceProvider;

        retryPolicy.Execute(() =>
        {
            try
            {
                UpdateDatabase(scopeServiceProvider, migrationDirection);
            }
            catch (NpgsqlException exception)
            {
                Logger.Error($"Failed to update database.", exception);
                if (!exception.IsTransient && exception.InnerException != null)
                {
                    throw exception.InnerException;
                }
                throw;
            }
        });
        return 0;
    }
    catch (Exception exception)
    {
        Logger.Error("Db migrator terminated unexpectedly", exception);
        return 1;
    }
```

Рис. 26 – Program.cs

```
private static IServiceProvider CreateServices()
{
    return new ServiceCollection()
          .AddScoped(provider => Configuration)
          .AddFluentMigratorCore()
          .ConfigureRunner(runnerBuilder =>
                           runnerBuilder
                              .AddPostgres()
                              .WithGlobalCommandTimeout(TimeSpan.FromMinutes(1))
                              .WithGlobalConnectionString(provider =>
                              {
                                  IConfiguration configuration =
                                      provider.GetRequiredService<IConfiguration>();

                                  string connectionString = configuration.GetConnectionString("migration-db");
BUG
                                  Logger.Information($"ConnectionString: {connectionString}");

                                  return connectionString;
                              })
                              // Define the assembly containing the migrations
                              .ScanIn(typeof(Program).Assembly)
                              .For.Migrations())
          // Enable logging to console
          .AddLogging(loggingBuilder => loggingBuilder.AddFluentMigratorConsole())
          .BuildServiceProvider(false);
}

ссылка: 1
private static IConfiguration Configuration
{
    get
    {
        if (_configuration != null)
        {
            return _configuration;
        }

        IConfigurationRoot configurationRoot = new ConfigurationBuilder()
                                               .AddJsonFile("appsettings.json")
                                               .AddEnvironmentVariables()
                                               .Build();

        _configuration = configurationRoot;

        return configurationRoot;
    }
}
```

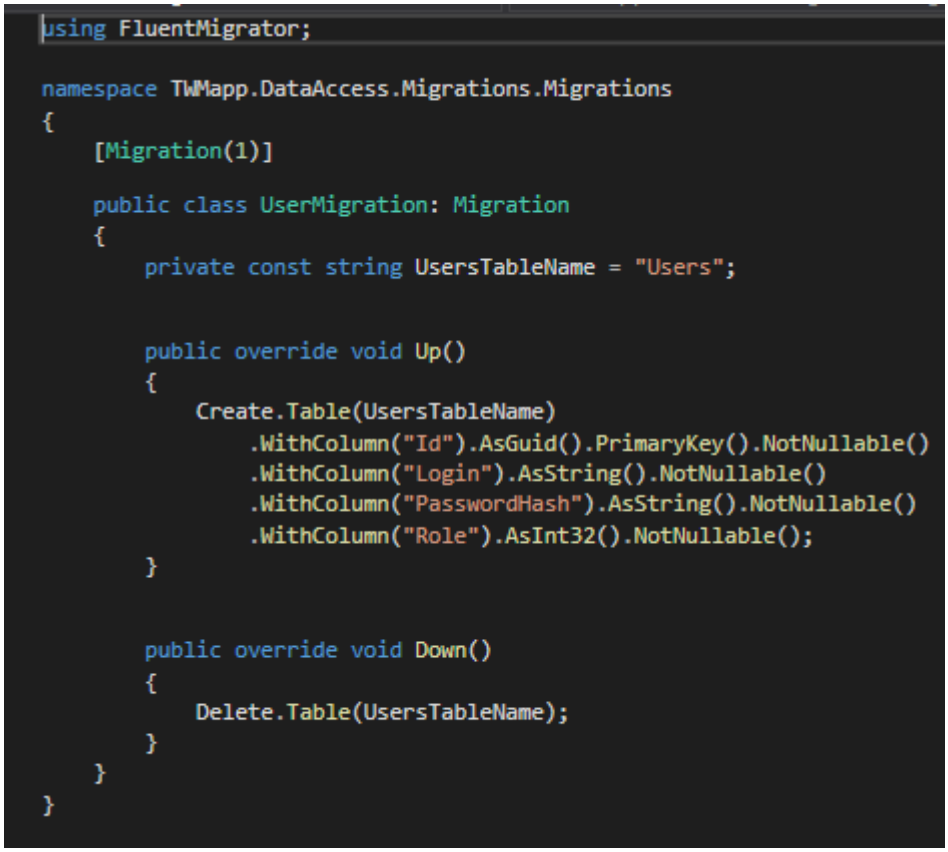
Рис. 27 – Program.cs

```
private static void UpdateDatabase(IServiceProvider serviceProvider, MigrationDirection migrationDirection)
{
    // Instantiate the runner
    IMigrationRunner runner = serviceProvider.GetRequiredService<IMigrationRunner>();

    Logger.Information("List all migrations");
    runner.ListMigrations();

    // Execute the migrations
    if (migrationDirection == MigrationDirection.Up)
    {
        if (!runner.HasMigrationsToApplyUp())
        {
            Logger.Warning("No applicable migrations to apply up");
            return;
        }

        Logger.Information("Execute all found (and not applied) migrations");

        runner.ValidateVersionOrder();
        runner.MigrateUp();
    }
    else
    {
        if (!runner.HasMigrationsToApplyRollback())
        {
            Logger.Warning("No applicable migrations to apply rollback");
            return;
        }

        Logger.Information("Rollback one step");
        runner.Rollback(1);
    }
}
```

Рис. 28 – Program.cs

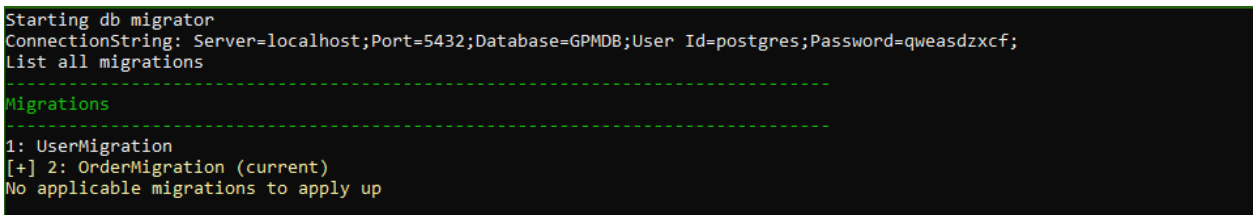
```
using FluentMigrator;
using System.Data;

namespace TWMapp.DataAccess.Migrations.Migrations
{
    [Migration(2)]
    Ссылок: 0
    public class OrderMigration: Migration
    {
        private const string UsersTableName = "Orders";

        Ссылок: 0
        public override void Up()
        {
            Create.Table(UsersTableName)
                .WithColumn("Id").AsGuid().PrimaryKey().NotNullable()
                .WithColumn("OrderNumber").AsInt32().NotNullable()
                .WithColumn("Description").AsString().NotNullable()
                .WithColumn("UserId").AsGuid().NotNullable().ForeignKey("Users", "Id").OnDelete(Rule.Cascade);
        }

        Ссылок: 0
        public override void Down()
        {
            Delete.Table(UsersTableName);
        }
    }
}
```

Рис. 29 – OrderMigration.cs



```
using FluentMigrator;

namespace TWMapp.DataAccess.Migrations.Migrations
{
    [Migration(1)]

    public class UserMigration: Migration
    {
        private const string UsersTableName = "Users";


        public override void Up()
        {
            Create.Table(UsersTableName)
                .WithColumn("Id").AsGuid().PrimaryKey().NotNullable()
                .WithColumn("Login").AsString().NotNullable()
                .WithColumn("PasswordHash").AsString().NotNullable()
                .WithColumn("Role").AsInt32().NotNullable();
        }


        public override void Down()
        {
            Delete.Table(UsersTableName);
        }
    }
}
```

Рис. 30 – UserMigration.cs



```
Starting db migrator
ConnectionString: Server=localhost;Port=5432;Database=GPMDB;User Id=postgres;Password=qweasdzxcf;
List all migrations
----------------------------------------------------------------------------
Migrations
----------------------------------------------------------------------------
1: UserMigration
[+] 2: OrderMigration (current)
No applicable migrations to apply up
```

Рис. 31 – Робота мігратора

Поля Entity та Migration майже ідентичні, оскільки деякі поля Entity необхідні для зручного вилучення пов'язаних об'єктів бази даних під час виконання програми.

Для зручності міграції встановлено пакет FluentMigrator, який спрощує їхній зовнішній вигляд.

Як видно на екрані UserMigration.cs, міграція складається з атрибуту та двох методів. Константа необов'язкова, оскільки це рядок імен таблиць. Атрибут [Migrations(1)] потрібний для послідовного створення таблиць. Метод Up() викликається під час створення таблиці, а метод Down() викликається при видаленні.

3.3.3 UserInterface

GPMapp.UserInterface – це шар який відповідає за вигляд проекту для користувача. Він містить у собі форми проекту з якими взаємодіє користувач.

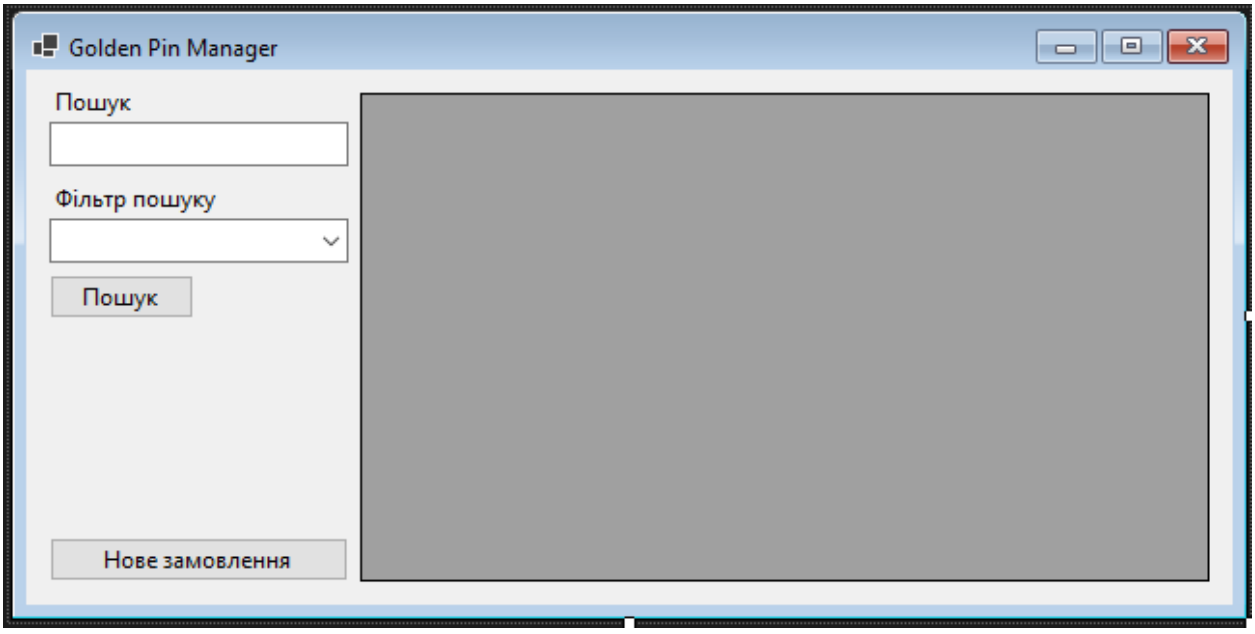


Рис. 32 – Домашній інтерфейс форми управління замовленнями

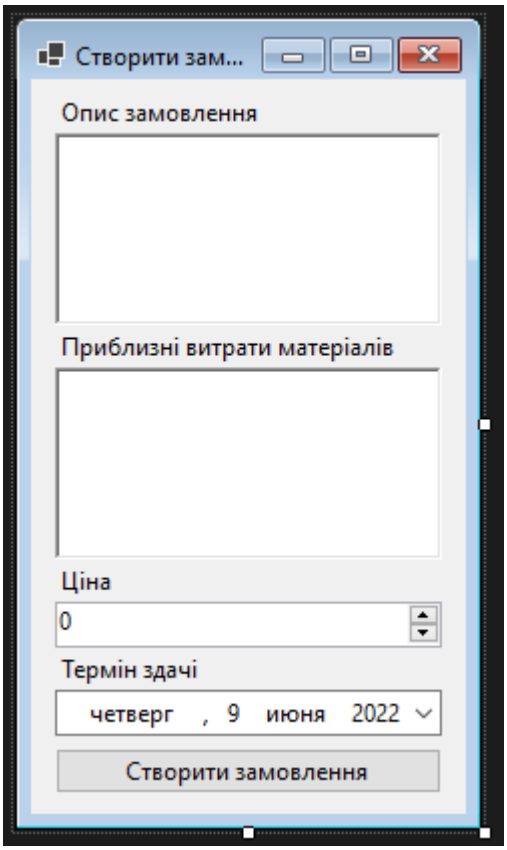


Рис. 33 – Форма створення замовлення

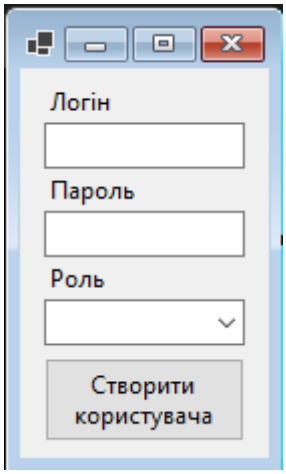


Рис. 34 – Форма створення користувача

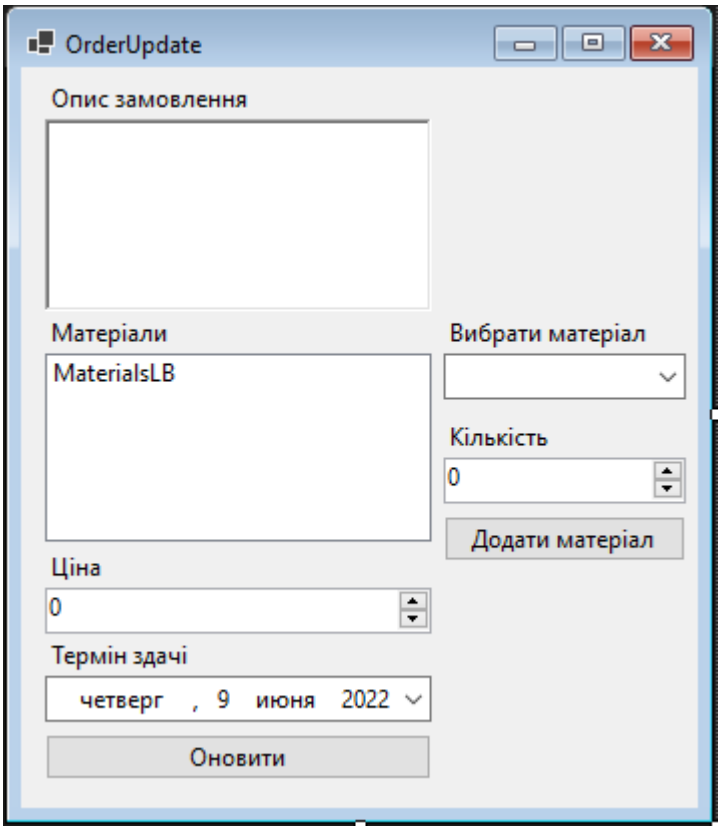


Рис. 35 – Форма оновлення замовлення

4. Тестування системи

Тестування це важлива частина розробки. Ручне тестування — це процес тестування програмного забезпечення, в якому ми виконуємо тестові випадки вручну без використання будь-яких автоматизованих інструментів тестування. Відповідно до точки зору кінцевого користувача, тестувальники виконуватимуть тестові випадки вручну. Він перевіряє, чи працює програма, як показано в документі з вимогами, чи ні. Це один із важливих процесів тестування, оскільки він може виявити як приховані, так і видимі помилки програмного забезпечення. Різниця між фактичним і очікуваним результатом називається помилкою.

Програмне забезпечення для ручного тестування є обов'язковим для кожного нещодавно створеного програмного забезпечення перед автоматизованим тестуванням. Ручне тестування вимагає багато часу та зусиль, але воно дає гарантію відсутності помилок у програмному забезпеченні.

4.1 Тест створення замовлення

Входимо на головну форму управління замовленнями та нажимаємо на кнопку «Нове замовлення»

Рис. 1 – Форма контролю замовлень

Після цього заповнюємо поля форми й натискаємо створити замовлення.

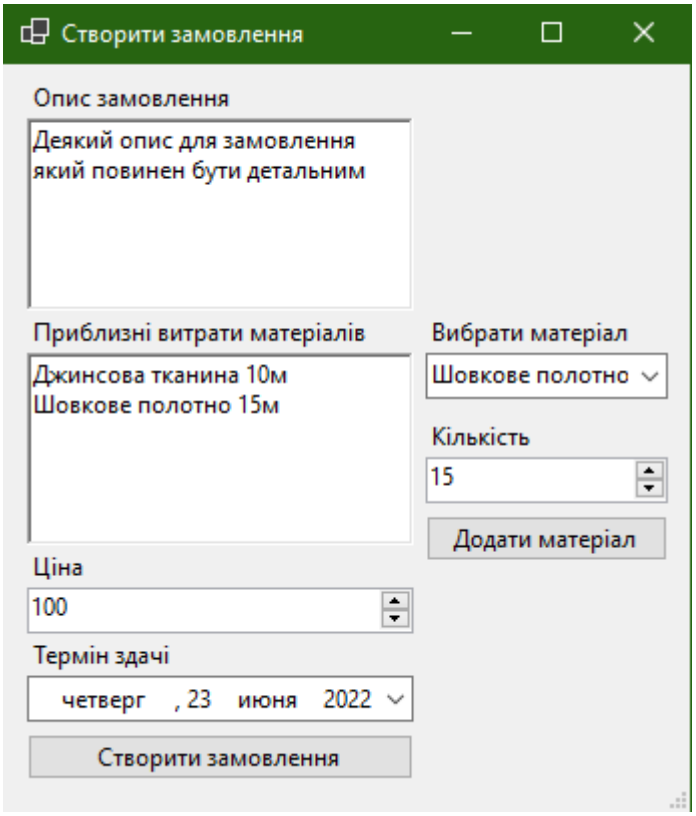


Рис. 2 – Створення замовлення

Після цього перевіряємо результат

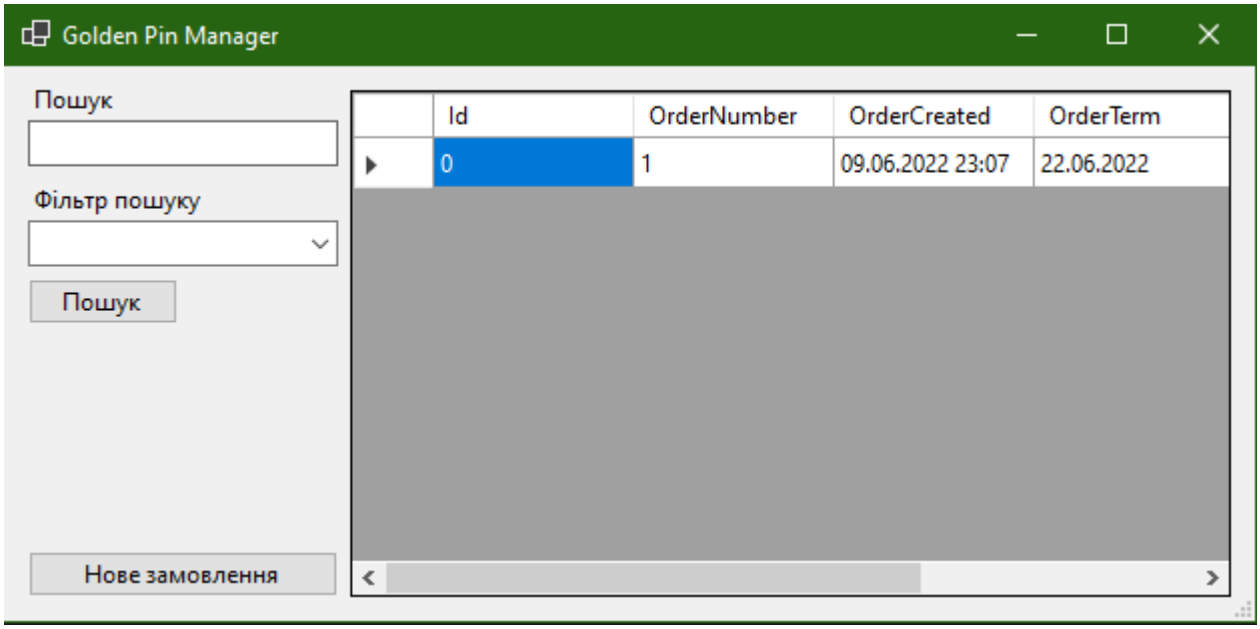


Рис. 3 – Створення замовлення

Висновки

Дана робота була спрямована на проектування програмного забезпечення для автоматизації прийому та видачі швейного виробництва.

1. Проведено аналіз актуальності розробленої системи завдяки дослідженням аналогів.
2. Вибрана монолітна архітектура з логічним поділом. В якості мови програмування було обрано C # 9.0, а також фреймворк ASP.NET Core для об’єктно-реляційної бази даних MSSQL Server, систему керування Git, інтегроване середовище розробки Visual studio та приклади переваг інструментів.
3. Було описано монолітну архітектуру та алгоритм роботи веб-сервісу.
4. Було розроблено програмне забезпечення для автоматизації прийому та видачі швейного виробництва мовою C#.
5. Розроблено програмні модулі для роботи з базою даних в основах системи ORM Entity Framework Core.
6. Було досліджено автентифікація на основі JWT токену.

   Результат дослідження бакалаврської роботи апробовані на всеукраїнських науково-технічний конференції: «використання MS SQL SERVER в якості системи управління базою данних» з темою «Використання ORM Entity Framework».

Перелік посилань

1. Git Docs [Електронний ресурс]: [Веб-сайт]. – електронні дані. – електронні дані. Режим доступу до ресурсу: https://git-scm.com/docs/git/ Дата звертання: 25.05.2022
2. Docker Manuals [Електронний ресурс]: [Веб-сайт]. – електронні дані. – електронні дані. Режим доступу до ресурсу: https://docs.docker.com/desktop/ Дата звертання: 25.05.2022
3. What is ProgreSQL [Електронний ресурс]: [Веб-сайт]. – електронні дані. – електронні дані. Режим доступу до ресурсу: https://www.postgresql.org/about/ Дата звертання: 25.05.2022
4. FluentValidation [Електронний ресурс]: [Веб-сайт]. – електронні дані. – електронні дані. Режим доступу до ресурсу: https://docs.fluentvalidation.net/en/latest/ Дата звертання: 25.05.2022
5. UML Diagram Types Guide: Learn About All Types of UML Diagrams with Examples [Електронний ресурс]: [Веб-сайт]. – електронні дані. – електронні дані. Режим доступу до ресурсу: https://creately.com/blog/diagrams/uml-diagram-types-examples/ Дата звертання: 25.05.2022
6. Entity Framework documentation [Електронний ресурс]: [Веб-сайт]. – електронні дані. – електронні дані. Режим доступу до ресурсу: https://docs.microsoft.com/en-us/ef/ Дата звертання: 25.05.2022
7. C# Documentation [Електронний ресурс]: [Веб-сайт]. – електронні дані. – електронні дані. Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/csharp/ Дата звертання: 25.05.2022
8. What is REST [Електронний ресурс]: [Веб-сайт]. – електронні дані. – електронні дані. Режим доступу до ресурсу: https://www.codecademy.com/article/what-is-rest/ Дата звертання: 25.05.2022
9. ASP.NET Core Performance Best Practices [Електронний ресурс]: [Веб-сайт]. – електронні дані. – електронні дані. Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/performance/performance-best-practices?view=aspnetcore-6.0/ Дата звертання: 25.05.2022

10. Code First vs. Database First vs. Model First Approach [Електронний ресурс]: [Веб-сайт]. – електронні дані. – електронні дані. Режим доступу до ресурсу: https://www.c-sharpcorner.com/blogs/code-first-vs-database-first-vs-model-first-approach1/ Дата звертання: 25.05.2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмного забезпечення для автоматизації процесів прийому та видачі замовлень швейного виробництва мовою програмування C#

Виконав студент 4 курсу
Групи ПД-43
КобиляцькийІлля Анатолійович
Керівник роботи
ст. викл. кафедри ІПЗ ГаманюкІгор Михайлович

Київ – 2022

АНАЛОГИ

Mircosoft Access

Microsoft Excel

Notepad++

АНАЛІЗ АНАЛОГІВ

Особливості	"GPM"	Access	Excel	Notepad++
Використовується БД	+	+	-	-
Безпечний доступ	+	+	-	-
Зручність використання	+	-	+	+
User Friendly	+	-	-	+
Фільтр інформації	+	+	+	-

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи - полегшити процес прийому та видачі швейного виробництва

Об'єкт дослідження – процес прийому та видачі швейного виробництва

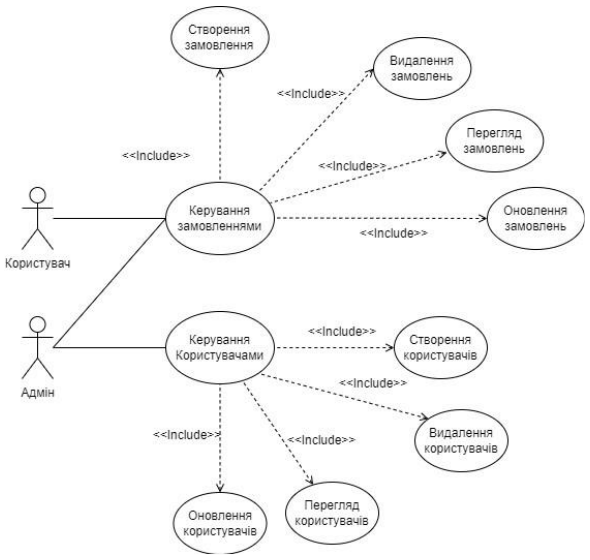
Предмет дослідження – програмне забезпечення для автоматизованого обліку швейного виробництва

ТЕХНІЧНІ ЗАВДАННЯ

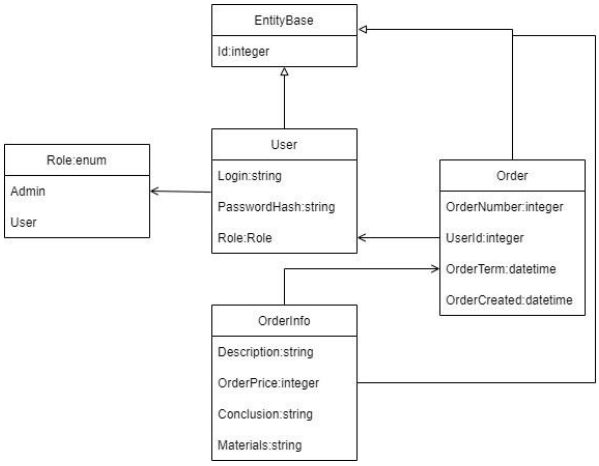
1. Під’єднати базу даних для зберігання замовлень
2. Реалізувати взаємодію з базами через шаблон Unit of Work
3. Розробити зручниі та інтуїтивно зрозумілі форми
4. Розробити функціонал управління користувачами
5. Розробити функціонал управління замовленнями

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

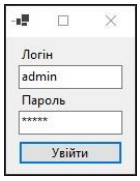
ДІАГРАМА ПРЕЦЕДЕНТІВ



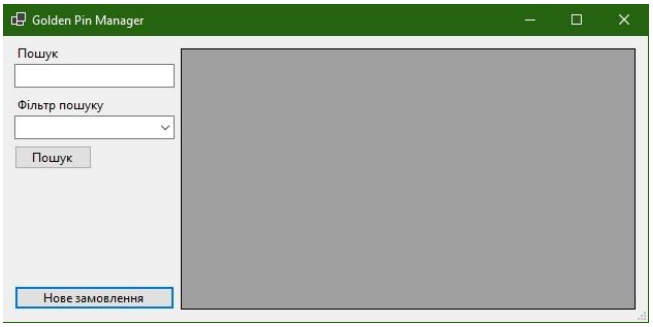
ДІАГРАМА КЛАСІВ



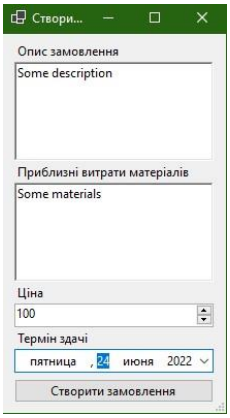
ПРИКЛАД РОБОТИ



Приклад входу



Форма керування замовленнями



Приклад створення замовлень

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

І.А. Кобиляцький використання MS SQL SERVER в якості системи управління базою данних / І.М. Гаманюк, І.А. Кобиляцький // Науково-технічна конференція «Застосування програмного забезпечення в інфокомунікаційних технологіях». Збірник тез. – К.: ДУТ, 2022. – С. 10-11

ВИСНОВКИ

1. Було розроблено програмні модулі для підключення до бази даних
2. Було розроблено зручний та інтуїтивно зрозумілий інтерфейс
3. Було розроблено функціонал управління замовленнями
4. Було розроблено програмний продукт для автоматизації процесів прийому та видачі шийного виробництва

ДЯКУЮ ЗА УВАГУ!