МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОДЕСЬКОГО НАЦІОНАЛЬНОГО ТЕХНОЛОГІЧНОГО УНІВЕРСИТЕТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітня програма: «Розробка програмного забезпечення»

Група: 4РП-05

Дипломний проект

здобувача освіти денної форми навчання
РП.05.15.000.ДП

НЄДЄВ
СТЕПАН
СЕРГІЙОВИЧ

м. Одеса
2022 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОДЕСЬКОГО НАЦІОНАЛЬНОГО ТЕХНОЛОГІЧНОГО УНІВЕРСИТЕТУ»

Спеціальність: **121 «Інженерія програмного забезпечення»**
Освітня програма: «**Розробка програмного забезпечення»**
Група: **4РП-05**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи) на тему:

Розробка запитів для обміну даними між сервером і базою даних за рахунок коштів Entity Framework Core

Проектний матеріал складається з пояснювальної записки на 48 сторінках та графічного (презентаційного) матеріалу на 10 аркушах (слайдах).

Дипломник _____ (Нєдєв С.С.)
Керівник _____ (Сологуб К.В.)

Консультанти:
з економічної частини _____ (Копайгородська Т.Г.)
з охорони праці_____ (Чорновол Н.І.)
з дотримання вимог ЄСКД _____ (Петрашова В.І.)
старший консультант _____ (Скорнякова О.В.)

До захисту допущений
Голова циклової комісії _____ (Скорнякова О.В.)
Завідувач відділення _____ (Суліма Ю.Ю.)

Захист «____»_____ 2022 р. Протокол ДКК №_____
Оцінка ДКК _____

Секретар ДКК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОДЕСЬКОГО НАЦІОНАЛЬНОГО ТЕХНОЛОГІЧНОГО УНІВЕРСИТЕТУ»

Відділення **комп'ютерних систем** Комісія **КТ та ІІІ**

Спеціальність **121 «Інженерія програмного забезпечення»**

Освітня програма **«Розробка програмного забезпечення»**

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР_____

_____

"_____"_____2022 р.

З А В Д А Н Н Я

на дипломний проект (роботу)

Здобувачеві (здобувачці) освіти Нєдєв Степан Сергійович

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розробка запитів для обміну даними між сервером і базою даних за рахунок коштів Entity Framework Core

затверджена наказом по коледжу від "30" грудня 2021 р. № 306-А2-ОД

2. Термін здачі закінченого проекту (роботи) _____

3. Вихідні данні до проекту (роботи) Microsoft Visual Studio, .Net, C#, MS SQL Server, HTML JavaScript, Docker, Swagger, API, HTTP

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

1. Доступ до оброблених даних між запитами. Транзакції запитання. 2. Економічний розрахунок. 3. Охорона праці.

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Презентація (10 слайдів)

6. Консультанти по проекту (роботі), із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний	Сологуб К.В.		
Економічний	Копайгородська Т.Г.		
Охорона праці	Чорновол Н.І.		
Нормоконтроль	Петрашова В.І.		
Старший консультант	Скорнякова О.В.		

7. Дата видачі завдання _____

Керівник _____

(підпис)

Завдання прийняв до виконання _____

(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту (роботи)	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1	Розділ 1. Доступ до оброблених даних між запитами		
2	Розділ 2. Економічний розрахунок		
3	Розділ 3. Охорона праці		
4	Розробка презентації до дипломної роботи		
5	Чистове оформлення пояснювальної записки		
6	Підготовка доповіді до захисту		
7	Отримання рецензії, відповіді на зауваження		
	рецензента		
8	Захист роботи		

Дипломник _____

(підпис)

Керівник _____

(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 05.15.000 ДП ПЗ	*Дипломний проект*		
А4			*РП 05.15.000 ДП ПЗ*	*Пояснювальна*	*1*	
				записка		
				Презентація		

Ізм.	Арк.	№ докум.	Підпис	Дата	*РП 05.22.000*		
Розробив		Нєдєв С.С.			Розробка запитів для обміну даними між сервером і базою даних за рахунок коштів Entity Framework Core	Літ.	Аркуш 3 / Аркушів
Перевірив		Сологуб К.В.					
Н. Контр.		Петрашова В.І				ВСП «ОТФК ОНТУ» гр.РП-05	
Затверд.		Скорнякова О.В.					

ЗМІСТ

ЗМІСТ 6

ВСТУП 7

МЕТА ТА ЗАВДАННЯ ПРОЕКТУ 9

1 ТЕХНОЛОГІЧНИЙ РОЗДІЛ. ДОСТУП ДО ОБРОБЛЕНИХ ДАНИХ МІЖ ЗАПИТАМИ. ТРАНЗАКЦІЇ ЗАПИТАННЯ 10

1.1 Розгортання та ініціалізація Entity Framework Core 10

1.2 Концептуальна модель даних 13

1.3 Реалізація "лейєрної архітектури". Структура обробки даних 15

1.4 Запити та настроювання середовища. Організація підтримки «Code first» архітектури даних 17

1.5 Транзакції даних. Побудова структури транзакції на CRUD – операції за даними 18

1.6 Побудова міграцій та їх застосування до бази даних. 23

1.7 Документація до проекту 27

1.7.1 Побудова класу DB Context 27

1.7.2 Автоматизація побудови баз даних під час запуску проекту 29

1.7.3 Структура CRUD – операцій 33

2 ЕКОНОМІЧНИЙ РОЗРАХУНОК 35

3 ОХОРОНА ПРАЦІ 41

ВИСНОВКИ 47

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 48

ВСТУП

Основою обліку, контролю та планування служать всілякі картотеки, реєстраційні журнали, списки тощо. Вони поступово накопичуються та оновлюються. При великому обсязі інформації пошук та узагальнення необхідних відомостей, що здійснюються вручну, є досить трудомістким процесом.

З появою ЕОМ та використанням їх для обробки інформації з'явилася можливість автоматизувати вирішення багатьох інформаційно-довідкових та розрахункових завдань.

Спочатку для накопичення та зберігання інформації на ЕОМ застосовувалися локальні масиви (або файли), при цьому для кожної з функціональних завдань, що вирішуються, створювалися власні файли вихідної і результатної інформації. Це призводило до значного дублювання даних, ускладнювало їх оновлення, ускладнювало вирішення взаємозалежних проблемних завдань.

Поступово з розвитком програмного забезпечення ЕОМ з'явилися ідеї створення керуючих систем, які дозволяли б накопичувати, зберігати і оновлювати взаємопов'язані дані з цілого комплексу завдань, наприклад при автоматизації бухгалтерського обліку на підприємстві. Ці ідеї знайшли своє втілення у системах управління базами даних (СУБД). СУБД взаємодіють немає з локальними, а взаємозалежними за інформацією масивами, званими базами даних. З появою персональних комп'ютерів СУБД стають найпопулярнішим засобом обробки табличної інформації. Вони є інструментальним засобом проектування банків даних під час обробки великих обсягів інформації.

Програмне забезпечення роботи з базами даних використовується на персональних комп'ютерах вже досить давно. На жаль, ці програми або були елементарними диспетчерами зберігання даних і не мали засобів розробки додатків, або були настільки складні і важкі, що навіть люди, що добре

знаються на комп'ютерах, уникали працювати з ними до тих пір, поки не отримували повних, орієнтованих на користувача додатків.

База даних - це впорядкований набір структурованої інформації або даних, які зазвичай зберігаються в електронному вигляді комп'ютерної системи. База даних зазвичай управляється системою управління базами даних (СУБД). Дані разом із СУБД, а також додатки, які з ними пов'язані, називаються системою баз даних, або, для стислості, просто базою даних.

Дані у найпоширеніших типах сучасних баз даних зазвичай зберігаються як рядків і стовпців формують таблицю. Цими даними можна легко керувати, змінювати, оновлювати, контролювати та впорядковувати. У більшості баз даних для запису та запитів даних використовується мова структурованих запитів (SQL).

МЕТА ТА ЗАВДАННЯ ПРОЕКТУ

Метою проекту є розробка функціоналу керування даними на веб-орієнтованій системі для керування даними на прикладі розробки блогу для організації спілкування користувачів з використанням функціоналу Entity Framework Core.

Для досягнення мети необхідно виконати такі завдання:

1. Розробити шаблон проекту за допомогою фреймворку ASP.NET MVC Core 6.

2. Реалізувати шаблон доступу до даних та маршрутизацію з використанням проміжного шару керування даними.

3. Реалізувати клас контексту задля забезпечення запиту до даних.

4. Використовуючи патерн "Репозиторій" реалізувати транзакції даних між Entity Framework та самою базою даних використовуючи мову інтегрованих запитів LINQ.

5. Грамотно скласти міграції, щоб відмалювати план проектування БД.

1 ТЕХНОЛОГІЧНИЙ РОЗДІЛ. ДОСТУП ДО ОБРОБЛЕНИХ ДАНИХ МІЖ ЗАПИТАМИ. ТРАНЗАКЦІЇ ЗАПИТАННЯ

1.1 Розгортання та ініціалізація Entity Framework Core

Entity Framework був вперше випущений у 2008 році, основним засобом взаємодії Microsoft між програмами .NET та реляційними базами даних. Entity Framework – це Object Relational Mapper (ORM), який є типом інструменту, який спрощує зіставлення між об'єктами у вашому програмному забезпеченні та таблицями та стовпцями реляційної бази даних.

Entity Framework (EF) – це середовище ORM з відкритим вихідним кодом для ADO.NET, яке є частиною .NET Framework.

ORM дбає про створення з'єднань з базою даних та виконання команд, а також про отримання результатів запиту та автоматичної матеріалізації цих результатів як об'єктів вашої програми.

ORM також допомагає відслідковувати зміни в цих об'єктах, і при отриманні інструкції він також зберігає ці зміни у базу даних.

Entity Framework (EF) – це середовище ORM з відкритим вихідним кодом для ADO.NET, яке є частиною .NET Framework.

ORM піклується про створення з'єднань з базою даних та виконання команд, а також отримання результатів запиту та автоматичної матеріалізації цих результатів як об'єктів програми.ORM также помогает отслеживать изменения в этих объектах, и при получении инструкции он также сохраняет эти изменения обратно в базу данных.

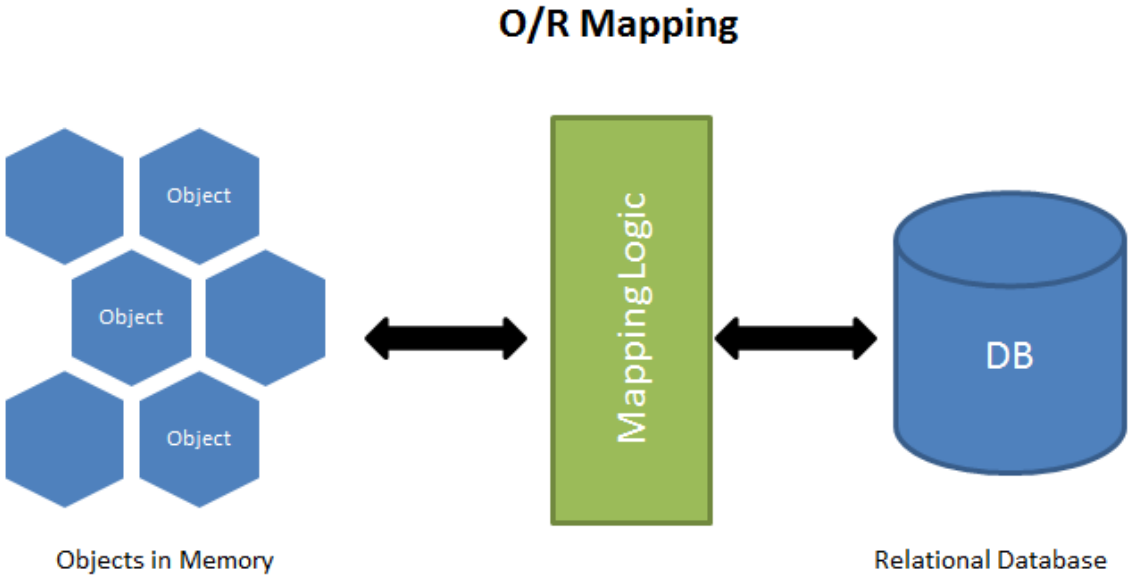

Рисунок 1.1 – Структура ORM – системи.

Entity Framework – це ORM, і ORM спрямовані підвищення продуктивність праці розробника з допомогою скорочення надлишкової завдання збереження даних, які у додатках.

Entity Framework може генерувати необхідні команди бази даних для читання або запису даних у базу даних та виконувати їх. Якщо потрібно запит, необхідно висловити свої запити до об'єктів зовнішнього домену, використовуючи LINQ для сутностей. Entity Framework виконає відповідний запит у базі даних, а потім матеріалізує результати до екземплярів зовнішніх доменних об'єктів для роботи в контексті програми.

Entity Framework може генерувати необхідні команди бази даних для читання або запису даних у базу даних та виконувати їх. Якщо йдеться про базу даних, необхідно висловити всі існуючі запити до об'єктів зовнішнього домену, використовуючи LINQ для сутностей. Entity Framework виконає відповідний запит у базі даних, а потім матеріалізує результати в екземплярах зовнішніх доменних об'єктів для роботи в контексті програми.

На ринку є й інші ORM, такі як **NHibernate** та **LLBLGen Pro**. Більшість ORM зазвичай відображають типи доменів безпосередньо у схему бази даних.

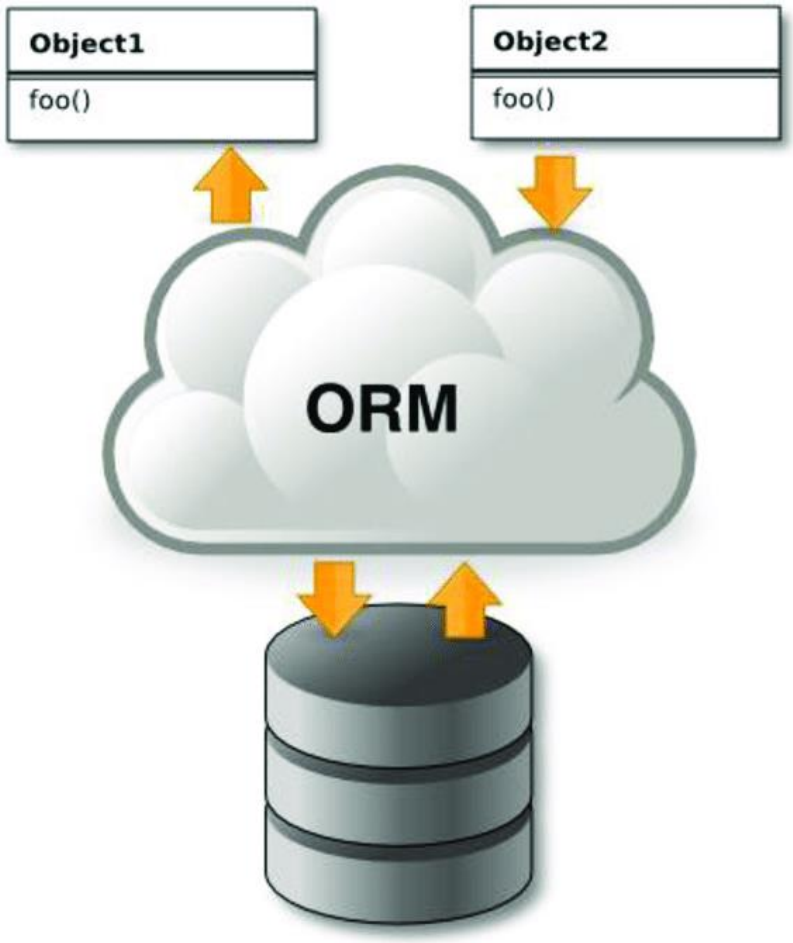

Рисунок 1.2 – Передача типів даних через ORM – систему

Entity Framework має більш детальний рівень відображення, тому на загальному рівні необхідно настроювати відображення, наприклад, зіставляючи один об'єкт із кількома таблицями бази даних або навіть кілька об'єктів із однією таблицею.

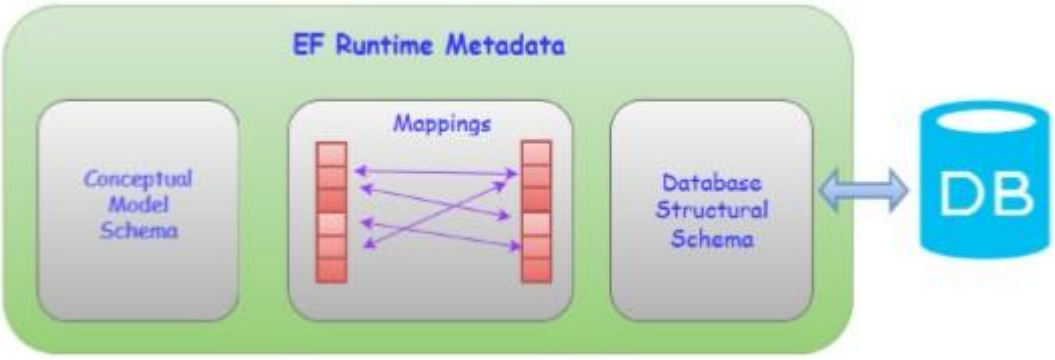

Рисунок 1.3 – Складання карти для відправки запиту в БД.

1.2 Концептуальна модель даних

У Entity Framework координаційний центр називається концептуальною моделлю. Це модель об'єктів у початковій програмі, а не модель бази даних, яка використовується за замовчуванням для збереження даних веб-програми.

Зовнішня концептуальна модель може збігтися зі схемою бази даних або може бути зовсім іншою.

Іноді можна використовувати Visual Designer для визначення зовнішньої концептуальної моделі, яка може генерувати класи, які в кінцевому підсумку буде використовуватися в зовнішньому додатку. Але найчастіше можна просто визначити свої класи та використовувати функцію Entity Framework, яка називається Code First. І тоді Entity Framework спіткає концептуальну модель.

У Entity Framework координаційний центр називається концептуальною моделлю. Це модель об'єктів у зовнішній програмі, а не модель бази даних, яку часто використовують для збереження даних веб-риложення.

Загальна концептуальна модель може збігтися із загальною чи концептуальною схемою бази даних або може бути зовсім іншою. Іноді можна використовувати Visual Designer для визначення загальної концептуальної моделі, яка може генерувати класи, які в кінцевому підсумку можна використовувати в своєму додатку. Можна просто визначити свої класи та використовувати функцію Entity Framework, яка називається Code First. І тоді Entity Framework спіткає концептуальну модель.

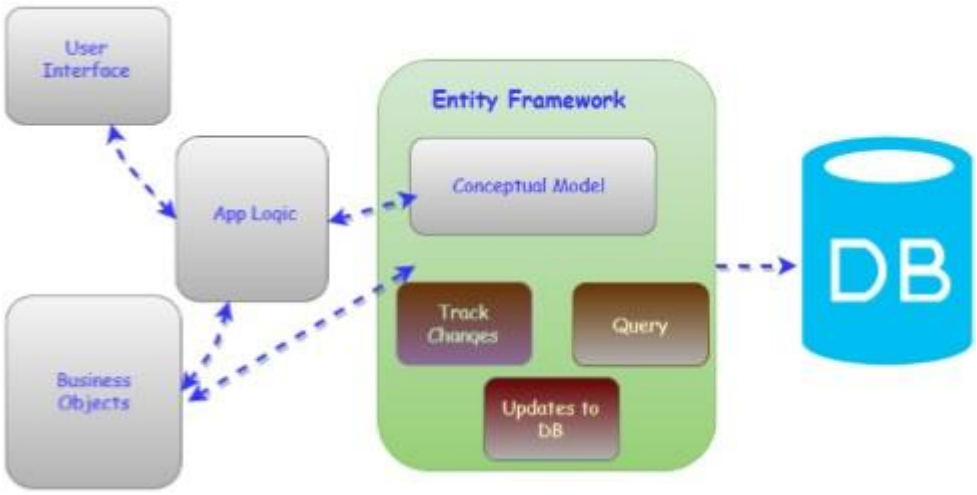

Рисунок 1.4 – Побудова логіки запиту між моделями та БД.

В даному випадку Entity Framework вирішує, як перейти від вихідної концептуальної моделі до існуючої бази даних. Таким чином, виходячи з контексту завдання, можна виконувати запити до об'єктів концептуальної моделі та працювати безпосередньо з ними.

1.3 Реалізація "лейєрної архітектури". Структура обробки даних

Архітектура Entity Framework складається з наступного:

1. Постачальники даних

Залежать від джерела, які абстрагують інтерфейси ADO.NET для підключення до бази даних під час програмування на основі концептуальної схеми. Він перекладає поширені мови SQL, такі як LINQ, через дерево команд у власний вираз SQL та виконує його для конкретної системи СУБД.

2. Entity Client

Цей рівень виставляє рівень сутності верхній рівень. Сутнісний клієнт надає можливість працювати з сутностями у формі рядків і стовпців, використовуючи запити сутнісних SQL без необхідності створювати класи для представлення концептуальної схеми. Entity Client показує рівні структури сутності, які є основними функціями. Ці верстви називаються Entity Data Model.

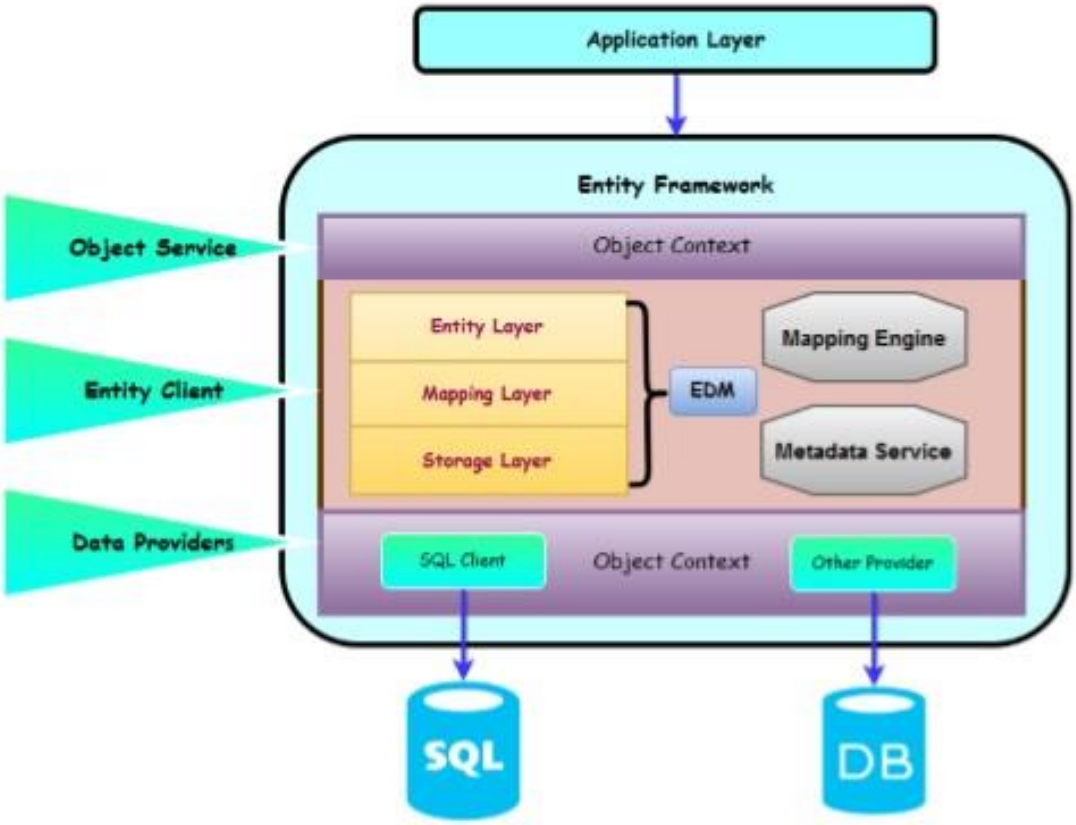

Рисунок 1.5 – Структура «леєрного» запиту в Entity Framework Core.

- Рівень зберігання містить всю схему бази даних у форматі XML.
- Шар сутності, який також є файлом XML, визначає сутності та відносини.
- Рівень відображення – це XML-файл, який відображає сутності та відносини, визначені на концептуальному рівні, з фактичними відносинами та таблицями, визначеними на логічному рівні.
- Служби метаданих, які також представлені в Entity Client, надають централізований API для доступу до шарів Entity, Mapping та Storage, які зберігаються у метаданих.

Об'єкт Сервіс

Object Services – це Object Context, який представляє сеанс взаємодії між програмами та джерелом даних.

Основне використання контексту об'єкта полягає у виконанні різних операцій, таких як додавання, видалення екземплярів об'єктів та збереження зміненого стану назад у базу даних за допомогою запитів.

Це рівень ORM Entity Framework, який є результатом даних для екземплярів об'єктів. За рахунок таких ресурсів послуги дозволяють розробнику використовувати деякі багаті функції ORM, такі як зіставлення первинних ключів, відстеження змін тощо. буд., шляхом написання запитів з використанням LINQ та Entity SQL.

1.4 Запити та настроювання середовища. Організація підтримки «Code first» архітектури даних

Підхід Database First надає альтернативу підходам Code First та Model First моделі Entity Data. Він створює коди моделей (класи, властивості, DbContext і т. д.) з бази даних у проекті, і ці класи стають сполучною ланкою між базою даних та контролером.

Перший підхід до бази даних створює структуру сутності існуючої бази даних. У проекті використовуються всі інші функціональні можливості, такі як синхронізація моделі/бази даних та генерація коду, так само, при використанні їх у підході Model First.

Два з них, Database First та Model First, залежали від конструктора Entity Framework у поєднанні з генерацією коду.

Третій Code First дозволяє пропустити візуальний дизайнер і просто написати свій власний код.

Класи доменів, і один або кілька класів Entity Framework DbContext дозволяють отримувати та зберігати дані, що стосуються цих класів. API-інтерфейс DbContext у ваших додатках використовується як мост між вашими

класами і вашою базою даних. DbContext є одним з найважливіших класів в Entity Framework. Це дозволяє висловлювати та виконувати запити. Він бере результати запитів з бази даних і перетворює їх на екземпляри класів концептуальної моделі.

Він може відслідковувати зміни в сутності, включаючи додавання та видалення, а потім ініціює створення операторів вставки, оновлення та видалення, які надсилаються до бази даних на вимогу.

1.5 Транзакції даних. Побудова структури транзакції на CRUD – операції за даними

У всіх версіях Entity Framework щоразу, коли при виконанні SaveChanges() для вставки, оновлення або видалення бази даних, ця оболонка буде укладати цю операцію транзакцію. При викликі SaveChanges контекст автоматично запускає транзакцію і фіксує або відкочує її в залежності від того, чи успішно збережено.

Entity Framework 6 забезпечує наступне:

Database.BeginTransaction()

Це простий і легший метод у існуючому DbContext для запуску та завершення транзакцій для користувачів. Дозволяє об'єднати кілька операцій в одній транзакції і, отже, всі вони зафіксовані, або всі відкатані як одна. Це також дозволяє користувачеві легко вказати рівень ізоляції для транзакції.

```
#region Transactions

2 references | 0 changes | 0 authors, 0 changes
public IDbContextTransaction BeginTransaction()
{
    return this.context.BeginTransaction();
}

3 references | 0 changes | 0 authors, 0 changes
public async Task<IDbContextTransaction> BeginTransactionAsync(
    CancellationToken cancellationToken = default(CancellationToken))
{
    return await this.context.BeginTransactionAsync(cancellationToken);
}

2 references | 0 changes | 0 authors, 0 changes
public async Task<IDbContextTransaction> BeginTransactionAsync(
    System.Data.IsolationLevel isolationLevel,
    CancellationToken cancellationToken = default(CancellationToken))
{
    return await this.context.BeginTransactionAsync(isolationLevel, cancellationToken);
}

#endregion
```

Рисунок 1.6 – Структура транзакций на уровне «репозитория»

Після цього поточні транзакції пов'язуються з транзакціями рівня контексту, який у свою чергу перевизначає метод SaveChangesAsync():

```
2 references | 0 changes | 0 authors, 0 changes
public virtual IDbContextTransaction BeginTransaction()
{
    return this.Database.BeginTransaction();
}

3 references | 0 changes | 0 authors, 0 changes
public virtual async Task<IDbContextTransaction> BeginTransactionAsync(
    CancellationToken cancellationToken = default(CancellationToken))
{
    return await this.Database.BeginTransactionAsync(cancellationToken);
}

2 references | 0 changes | 0 authors, 0 changes
public virtual async Task<IDbContextTransaction> BeginTransactionAsync(
    IsolationLevel isolationLevel,
    CancellationToken cancellationToken = default(CancellationToken))
{
    return await this.Database.BeginTransactionAsync(isolationLevel, cancellationToken);
}

12 references | 0 changes | 0 authors, 0 changes
public override Task<int> SaveChangesAsync(CancellationToken cancellationToken = default(CancellationToken))
{
    this.AddTimestamps();

    return base.SaveChangesAsync(cancellationToken);
}

0 references | 0 changes | 0 authors, 0 changes
public override int SaveChanges()
{
    this.AddTimestamps();

    return base.SaveChanges();
}
```

Рисунок 1.7 – Структура контекста сохранения данных

Більшість операцій з даними так чи інакше є CRUD операції (Create, Read, Update, Delete), тобто створення, отримання, оновлення та видалення. Entity Framework Core дозволяє легко виконувати всі ці дії.

Використовуємо сутність, яка показана на рисунку 1.11.

Далі, виходячи з процесу побудови транзакції, можемо побудувати стандартні CRUD – опрерації по роботі з даними:

```
2 references | 0 changes | 0 authors, 0 changes
public async Task<Blog> AddBlogAsync(Blog blog)
{
    using (IDbContextTransaction dbContextTransaction = await this.BeginTransactionAsync())
    {
        try
        {
            blog.BlogId = blog.BlogId == null ? Guid.NewGuid().ToString() : blog.BlogId;

            this.context.Blogs.Add(blog);

            await this.context.SaveChangesAsync();

            dbContextTransaction.Commit();

            return blog;
        }
        catch (Exception ex)
        {
            dbContextTransaction.Rollback();

            throw ex;
        }
    }
}
```

Рисунок 1.8 – Додавання блога

```
4 references | 0 changes | 0 authors, 0 changes
public async Task<Blog> UpdateBlogAsync(string blogId, string title, string content, string userId)
{
    blogId = blogId ?? throw new ArgumentNullException(nameof(blogId));

    try
    {
        Blog blog = await this.context.Blogs.FirstOrDefaultAsync(bl => bl.IsActive.Value && bl.BlogId.Equals(blogId));

        if (!Equals(blog, null))
        {
            if (!blog.UserId.Equals(userId))
            {
                throw new Exception("You can update only own blogs!");
            }

            blog.Title = title;
            blog.Content = content;
        }
        else
        {
            throw new Exception($"Blog with id {blogId} not found.");
        }

        await this.context.SaveChangesAsync();

        return blog;
    }
    catch (Exception ex)
    {
        throw ex;
    }
}
```

Римсунок 1.9 – Оновлення блога

```
2 references | 0 changes | 0 authors, 0 changes
public async Task<bool> DeleteBlogAsync(string blogId, string userId)
{
    try
    {
        Blog deleteBlog = await this.context.Blogs
            .FirstOrDefaultAsync(a => a.IsActive.Value && a.BlogId.Equals(blogId));

        if (!Equals(deleteBlog, null))
        {
            if (!deleteBlog.UserId.Equals(userId))
            {
                throw new Exception("You can remove only own blogs!");
            }

            this.context.Blogs.Remove(deleteBlog);
        }
        else
        {
            throw new Exception($"Blog with id {blogId} not found.");
        }

        return await this.context.SaveChangesAsync() > 0;
    }
    catch (Exception ex)
    {
        throw ex;
    }
}
```

Рисунок 1.10 – Видалення блога

1.6 Побудова міграцій та їх застосування до бази даних.

Entity Framework включає функцію Code First Migrations, яка дозволяє поступово розвивати схему бази даних у міру зміни моделі з часом. Для розбудови проекту це є більшим поліпшенням порівняно з опціями ініціалізатора бази даних із випусків 4.1 та 4.2, які вимагають ручного оновлення бази даних або її видалення та повторного створення при зміні моделі.

До Entity Framework 4.3, якщо присуджують дані (крім початкових даних) або наявні процедури, що тривають, тригери і т. д. У існуючій базі даних, ці стратегії використовувалися для видалення всієї бази даних і її відтворення, щоб уникнути втрати даних.

При міграції він (Entity Framework) автоматично оновить схему бази даних, коли поточна модель зміниться без втрати існуючих даних або інших об'єктів бази даних.

Він використовує новий ініціалізатор бази даних з ім'ям **MigrateDatabaseToLatestVersion**.

Існує два види міграції:

1. Автоматизована міграція

Автоматизована міграція була вперше представлена Entity Framework 4.3. При автоматичній міграції не потрібно обробляти міграцію бази даних вручну у файлі коду. Наприклад, для кожної зміни також потрібно буде змінити класи поточного домену. Але для автоматичної міграції потрібно просто виконати команду в консолі диспетчера пакетів, щоб зробити це.

```
6   namespace Application.DataAccess.Entities
7   {
8       using System.Collections.Generic;
9       using System.ComponentModel.DataAnnotations;
10      using System.ComponentModel.DataAnnotations.Schema;
11
12      public class Blog : ApplicationEntity
13      {
14          [Key]
15          [StringLength(50)]
16          public string BlogId { get; set; }
17
18          [StringLength(255)]
19          public string Title { get; set; }
20
                6 references | 0 changes | 0 authors, 0 changes
21          public string Content { get; set; }
22
23          [Required]
24          [StringLength(50)]
                0 references | 0 changes | 0 authors, 0 changes
25          public string Author { get; set; }
26
27          [ForeignKey("ApplicationUser")]
28          [StringLength(450)]
                8 references | 0 changes | 0 authors, 0 changes
29          public string UserId { get; set; }
30
                0 references | 0 changes | 0 authors, 0 changes
31          public virtual ApplicationUser ApplicationUser { get; set; }
32
                0 references | 0 changes | 0 authors, 0 changes
33          public ICollection<FileStorage> FileStorages { get; set; }
34      }
35  }
```

Рисунок 1.11 - Конкретна «сутність» об'єкту Blog

Щоб запустити процес автоматичної міграції, необхідно виконати команду:

```
>enable-migrations -EnableAutomaticMigrations:$true
```

Рисунок 1.12 - Підключення використання міграцій у проекті

Після успішного виконання команди вона створює внутрішній запечатаний клас Configuration у папці Migration проекту, як показано на рисунку 1.13.

```
114         modelBuilder.Entity("Application.DataAccess.Entities.Blog", b =>
115             {
116                 b.Property<string>("BlogId")
117                     .HasColumnType("nvarchar(50)")
118                     .HasMaxLength(50);
119 
120                 b.Property<string>("Author")
121                     .IsRequired()
122                     .HasColumnType("nvarchar(50)")
123                     .HasMaxLength(50);
124 
125                 b.Property<string>("Content")
126                     .HasColumnType("nvarchar(max)");
127 
128                 b.Property<string>("CreatedBy")
129                     .IsRequired()
130                     .HasColumnType("nvarchar(450)")
131                     .HasMaxLength(450);
132 
133                 b.Property<DateTime>("CreatedDate")
134                     .ValueGeneratedOnAddOrUpdate()
135                     .HasColumnType("DateTime2")
136                     .HasDefaultValueSql("GETDATE()");
137 
138                 b.Property<bool?>("IsActive")
139                     .IsRequired()
140                     .ValueGeneratedOnAdd()
141                     .HasColumnType("bit")
142                     .HasDefaultValueSql("1");
143 
144                 b.Property<byte[]>("RowVersion")
145                     .IsConcurrencyToken()
146                     .ValueGeneratedOnAddOrUpdate()
147                     .HasColumnType("rowversion");
148 
149                 b.Property<string>("Title")
150                     .HasColumnType("nvarchar(255)")
151                     .HasMaxLength(255);
152 
153                 b.Property<string>("UpdatedBy")
154                     .HasColumnType("nvarchar(450)")
155                     .HasMaxLength(450);
156 
157                 b.Property<DateTime?>("UpdatedDate")
158                     .ValueGeneratedOnAddOrUpdate()
159                     .HasColumnType("DateTime2")
160                     .HasComputedColumnSql("GETDATE()");
161 
162                 b.Property<string>("UserId")
163                     .HasColumnType("nvarchar(450)")
164                     .HasMaxLength(450);
165 
166                 b.HasKey("BlogId");
167 
168                 b.HasIndex("UserId");
169 
170                 b.ToTable("Blogs");
171             });
```

Рисунок 1.13 - Знімок бази даних

```
migrationBuilder.CreateTable(
    name: "Blogs",
    columns: table => new
    {
        BlogId = table.Column<string>(maxLength: 50, nullable: false),
        CreatedDate = table.Column<DateTime>(type: "DateTime2", nullable: false, defaultValueSql: "GETDATE()"),
        UpdatedDate = table.Column<DateTime>(type: "DateTime2", nullable: true, computedColumnSql: "GETDATE()"),
        IsActive = table.Column<bool>(nullable: false, defaultValueSql: "1"),
        RowVersion = table.Column<byte[]>(rowVersion: true, nullable: true),
        CreatedBy = table.Column<string>(maxLength: 450, nullable: false),
        UpdatedBy = table.Column<string>(maxLength: 450, nullable: true),
        Title = table.Column<string>(maxLength: 255, nullable: true),
        Content = table.Column<string>(maxLength: 2550, nullable: true),
        UserId = table.Column<string>(maxLength: 450, nullable: true)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Blogs", x => x.BlogId);
        table.ForeignKey(
            name: "FK_Blogs_AspNetUsers_UserId",
            column: x => x.UserId,
            principalTable: "AspNetUsers",
            principalColumn: "Id",
            onDelete: ReferentialAction.Restrict);
    });
```

Рисунок 1.14 - Міграція побудови «сутності» Blog

```
migrationBuilder.CreateTable(
    name: "FileStorages",
    columns: table => new
    {
        FileId = table.Column<string>(maxLength: 125, nullable: false),
        CreatedDate = table.Column<DateTime>(type: "DateTime2", nullable: false, defaultValueSql: "GETDATE()"),
        UpdatedDate = table.Column<DateTime>(type: "DateTime2", nullable: true, computedColumnSql: "GETDATE()"),
        IsActive = table.Column<bool>(nullable: false, defaultValueSql: "1"),
        RowVersion = table.Column<byte[]>(rowVersion: true, nullable: true),
        CreatedBy = table.Column<string>(maxLength: 450, nullable: false),
        UpdatedBy = table.Column<string>(maxLength: 450, nullable: true),
        Url = table.Column<string>(maxLength: 255, nullable: true),
        Title = table.Column<string>(maxLength: 75, nullable: true),
        Description = table.Column<string>(maxLength: 255, nullable: true),
        UserId = table.Column<string>(maxLength: 450, nullable: true),
        BlogId = table.Column<string>(nullable: true)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_FileStorages", x => x.FileId);
        table.ForeignKey(
            name: "FK_FileStorages_Blogs_BlogId",
            column: x => x.BlogId,
            principalTable: "Blogs",
            principalColumn: "BlogId",
            onDelete: ReferentialAction.Restrict);
        table.ForeignKey(
            name: "FK_FileStorages_AspNetUsers_UserId",
            column: x => x.UserId,
            principalTable: "AspNetUsers",
            principalColumn: "Id",
            onDelete: ReferentialAction.Restrict);
    });
```

Рисунок 1.15 - Міграція побудови «сутності» FileStorage

2. Міграція на основі коду

Модель даних часто змінюється, і щоразу, коли модель змінюється, вона синхронізується з базою даних. Після налаштування Entity Framework для автоматичного видалення та повторного створення бази даних при кожній зміні моделі даних. Міграція на основі коду корисна, коли проект потребує більшого контролю над міграцією.

При додаванні, видаленні або зміні класу сутностей або зміні базового класу DbContext, при наступному запуску програми воно автоматично видаляє існуючу базу даних, створює нову, відповідну моделі та заповнює її тестовими даними.

Функція Code First Migrations вирішує цю проблему, дозволяючи Code First оновлювати схему бази даних замість її видалення та повторного створення. Щоб розгорнути програму, потрібно увімкнути міграцію.

Ось основне правило для перенесення змін до бази даних:

• Увімкнути міграцію

• Додати міграцію

• Оновлення бази даних

Використовуючи команду з рисунку 1.12 ми включаємо процес міграції, тепер використовуємо функцію додавання міграції, виконавши таку команду:

```
> dotnet ef migrations add AddBlogMigration
```

Рисунок 1.16 – Додавання міграції до проекту

1.7 Документація до проекту

1.7.1 Побудова класу DB Context

Для побудови класу контексту необхідно для початку створити шаблон класу контексту:

```
public partial class DataContext : IdentityDbContext<ApplicationUser, ApplicationRole, string>
{
    private readonly ClaimsPrincipal principal;

    private string userName;

    1 reference | 0 changes | 0 authors, 0 changes
    public DataContext(DbContextOptions<DataContext> options)
        : base(options)
    {
    }

    0 references | 0 changes | 0 authors, 0 changes
    public DataContext(DbContextOptions<DataContext> options, IPrincipal principal)
        : base(options)
    {
        this.principal = principal as ClaimsPrincipal;
    }
```

Рисунок 1.17 – Шаблон класу контексту

В даному шаблоні класу контексту ми реалізовуємо спадкування від базового класу контексту, а так само підключаємо рольову систему для автентифікації та реалізовуємо конструктори для визначення класів доступності за замовчуванням.

```
1 reference | 0 changes | 0 authors, 0 changes
internal void ApplicationEntityDefaultValueSqlForAllObjects(ModelBuilder builder)
{
    // Execute ApplicationEntityDefaultValueSql for every child of the ApplicationEntity
    var applicationEntityTypes = typeof(ApplicationEntity).GetTypeInfo().Assembly.GetTypes().Where(
        t => t.GetTypeInfo().IsClass
        && typeof(ApplicationEntity).IsAssignableFrom(t)
        && !t.GetTypeInfo().IsAbstract
        && t.GetTypeInfo().IsSubclassOf(typeof(ApplicationEntity)));

    foreach (Type type in applicationEntityTypes)
    {
        var dbSetType = this.GetType()
            .GetRuntimeProperties()
            .Where(o =>
            o.PropertyType.GetTypeInfo().IsGenericType &&
            o.PropertyType.GetGenericTypeDefinition() == typeof(DbSet<>) &&
            o.PropertyType.GenericTypeArguments.Contains(type))
            .FirstOrDefault();

        if (dbSetType != null)
        {
            MethodInfo method = typeof(DataContext)
                .GetMethod("ApplicationEntityDefaultValueSql", BindingFlags.Instance | BindingFlags.NonPublic);
            if (method == null)
            {
                throw new NotImplementedException("The 'ApplicationEntityDefaultValueSql' method is not implemented");
            }

            MethodInfo generic = method.MakeGenericMethod(type);
            generic?.Invoke(this, new object[] { builder });
        }
    }
}
```

Рисунок 1.18 – Перевизначення значень для SQL – об'єктів за замовчуванням

Цей функціонал будемо використовувати для перевизначення значень полів за промовчанням для класів контектів. Наприклад для абстрактного класу, який за замовчуванням визначає поля для сутностей:

```
public abstract class ApplicationEntity
{
    [Column(TypeName = "DateTime2")]
    [DatabaseGenerated(DatabaseGeneratedOption.Computed)] // this will prevent the column to be updated
    10 references | 0 changes | 0 authors, 0 changes
    public DateTime CreatedDate { get; internal set; }

    [Column(TypeName = "DateTime2")]
    [DatabaseGenerated(DatabaseGeneratedOption.Computed)]
    5 references | 0 changes | 0 authors, 0 changes
    public DateTime? UpdatedDate { get; internal set; }

    [Required]
    40 references | 0 changes | 0 authors, 0 changes
    public bool? IsActive { get; set; }

    [NotMapped]
    6 references | 0 changes | 0 authors, 0 changes
    public virtual string Timestamp
    {
        get { return this.RowVersion == null ? null : Convert.ToBase64String(this.RowVersion); }
        set { this.RowVersion = string.IsNullOrEmpty(value) ? null : Convert.FromBase64String(value); }
    }
```

Рисунок 1.19 – Клас контексту, для розширення сутностей

1.7.2 Автоматизація побудови баз даних під час запуску проекту

Для початку необхідно налаштувати метод, який підключаючись до інших методів буде за рахунок міграція будувати базу даних:

```
using (var serviceScope = app.ApplicationServices.GetRequiredService<IServiceScopeFactory>().CreateScope())
{
    using (var context = serviceScope.ServiceProvider.GetService<DataContext>())
    {
        try
        {
            context.Database.Migrate();

            int result = context.Initialize().Result;
        }
        catch
        {
            throw;
        }
    }
}
```

Рисунок 1.20 – Конфігурація створення БД під час запуску проекту

При побудові бази даних нам необхідно орієнтуватися на 2 типи побудови та збереження даних:

1. Побудова інформаційної таблиці

При побудові інформаційної таблиці необхідно побудувати та ініціалізувати спеціальний клас, який займатиметься наповненням даних існуючої таблиці:

А. Побудови метод який займатиметься додаванням даних по id міграції:

```
1 reference | 0 changes | 0 authors, 0 changes
public async Task<int> Initialize()
{
    try
    {
        int updatedRowsCount = 0;
        List<CompleteMigration> completeMigrations = await this.CompleteMigrations.ToListAsync();

        using (var dbContextTransaction = await this.BeginTransactionAsync())
        {
            try
            {
                updatedRowsCount += await this.InitializeDbContext(completeMigrations);
                dbContextTransaction.Commit();
                return updatedRowsCount;
            }
            catch (Exception ex)
            {
                dbContextTransaction.Rollback();
                throw ex;
            }
        }
    }
    catch (Exception ex)
    {
        throw ex;
    }
}
```

Рисунок 1.21 – Ініціалізація даних для заповнення таблиці

Б. Наприклад напишемо метод який ініціалізуватиме сам контекст даних:

```
1 reference | 0 changes | 0 authors, 0 changes
private async Task<int> InitializeDbContext(IList<CompleteMigration> completeMigrations)
{
    bool toSave = false;
    int numberUpdatedRecords = 0;

    string initMigrationId = "D31E30D8-321D-4A9B-9757-3B6W3E4A4215";

    if (!completeMigrations.Any(cm => cm.CompleteMigrationId == initMigrationId))
    {
        // example for put a function
        // toSave |= this.Initialize();

        // save only if there is any updates
        if (toSave)
        {
            numberUpdatedRecords = await this.SaveChangesAsync();
        }
    }

    return numberUpdatedRecords;
}
```

Рисунок 1.22 – Ініціалізація контексту БД

Також підключимо сам метод який виводитиме дані в БД при ініціалізації даних

```
int result = context.Initialize().Result;
```

Рисунок 1.23 – Відпрацювання даних.

Наповнення існуючих таблиць даними, що вводяться користувачем

MVC розшифровується як Model View Controller. Це шаблон проектування, який використовується для поділу бізнес-логіки, логіки уявлення та даних. По суті він надає шаблон для стилізації веб-додатку. Відповідно до MVC ви можете розділити додаток на 3 шари таким чином:

Рівень моделі: компонент Model відповідає всій чи будь-якій логіці, пов'язаній з даними, з якою працює користувач. Це представлятиме або інформацію, яка передається між компонентами View і Controller, або інші дані, пов'язані з бізнес-логікою. Наприклад, об'єкт Customer буде отримувати інформацію про клієнта з бази даних, маніпулювати нею та оновлювати свої дані назад у базу даних або використовувати її для відображення даних.

Рівень представлення: компонент View використовується для всієї логіки інтерфейсу користувача. Наприклад, уявлення Customer буде включати

всі компоненти інтерфейсу користувача, такі як текстові поля, що розкриваються списки і т. д., з якими взаємодіє кінцевий користувач.

Контролер: Контролери діють як інтерфейс між Моделью та розглядають компоненти для обробки всієї бізнес-логіки та вхідних запитів, маніпулювання даними за допомогою компонента Моделі та взаємодії з Уявленнями для відображення кінцевого результату. Наприклад, контролер клієнта оброблятиме всі взаємодії та вхідні дані з представлення клієнта та оновлюватиме базу даних за допомогою клієнтської моделі.

ЗНАЧЕННЯ CRUD: CRUD є комбінацією з чотирьох функцій, які створюють, читають, оновлюють і видаляють, які необхідні для обслуговування будь-якої програми зберігання, в основному він показує основні чотири основні функції, які можуть бути виконані в будь-якій базі даних, пов'язаної з нашим додатком .

Програми CRUD - це ті, що містять основні функції CRUD, які присутні у різних системах РСУБД, таких як Microsoft SQL Server, MySQL, база даних Oracle та інші. Чотири функції, використовувані користувачами для виконання різних типів операцій із вибраними даними у базі даних. Це можна було зробити лише за допомогою коду або через графічний інтерфейс користувача (GUI).

1.7.3 Структура CRUD – операцій

Для виконання CRUD – операцій над об'єктом для початку в класі контексту необхідно оголосити низку установок середовища для розпізнавання контексту роботи над об'єктом:

```
0 references | 0 changes | 0 authors, 0 changes
protected override void OnModelCreating(ModelBuilder builder)
{
    base.OnModelCreating(builder);

    this.ApplicationEntityDefaultValueSqlForAllObjects(builder);

    builder.Entity<UserAcceptedTerm>()
        .HasKey(t => new { t.UserId, t.TermsOfServiceId });
}
```

Рисунок 1.24 – Ініціалізація моделі додавання даних

```
1 reference | 0 changes | 0 authors, 0 changes
public virtual DbSet<CompleteMigration> CompleteMigrations { get; set; }

public virtual DbSet<UserAcceptedTerm> UserAcceptedTerms { get; set; }

public virtual DbSet<TermsOfService> TermsOfServices { get; set; }

8 references | 0 changes | 0 authors, 0 changes
public virtual DbSet<Blog> Blogs { get; set; }

public virtual DbSet<FileStorage> FileStorages { get; set; }
```

Рисунок 1.25 – Инициализация сущностей для отработки Entity Framework Core

Потім за допомогою класу репозиторію можна ініціалізувати процес додавання самих операцій до БД.

```
2 references | 0 changes | 0 authors, 0 changes
public async Task<bool> SaveBlogFileAsync(string userId, string fileBlobName, string title)
{
    try
    {
        FileStorage blogFile = new FileStorage
        {
            FileId = fileBlobName,
            UserId = userId,
            Title = title ?? string.Empty,
        };

        this.context.FileStorages.Add(blogFile);

        return await this.context.SaveChangesAsync() > 0;
    }
    catch
    {
        throw;
    }
}
```

Рисунок 1.26 – Додавання файлу до БД.

```
2 references | 0 changes | 0 authors, 0 changes
public async Task<FileStorage> GetBlogFileAsync(string fileBlobName)
{
    return await this.context.FileStorages.FirstOrDefaultAsync(f => f.FileId.Equals(fileBlobName));
}

1 reference | 0 changes | 0 authors, 0 changes
public async Task<List<FileStorage>> GetAllFilesAsync()
{
    return await this.context.FileStorages.ToListAsync();
}
```

Рисунок 1.27– Отримання списку файлів із БД

2 ЕКОНОМІЧНИЙ РОЗРАХУНОК

2.1 Резюме

Темою даного дипломного проекту є «Розробка запитів для обміну даними між сервером і базою даних за рахунок коштів Entity Framework Core». Було розроблено інформаційне та програмне забезпечення комплексу завдань, які дозволяють скоротити трудомісткість виконуваних робіт та вартісні витрати на їх виконання.

Ефективність кожного програмного продукту визначається його якістю та ефективністю процесу розробки. Якість ПП визначається наступними складовими: з точки зору користувача; з позиції використання ресурсів; виконання вимог до програмного забезпечення.

Оцінка якості програмного продукту з точки зору користувача надала такі переваги порівняно із звичайним документообігом:

1. Можливість оперативного контролю за достовірністю інформації;
2. Зменшення кількості можливих помилок при генеруванні похідних даних;
3. Можливість швидкого доступу до будь-яких даних;
4. Економія пам'яті в процесі складання запиту для зв'язування та управління даними звіту.

Оцінка якості програмного продукту включає визначення трудомісткості і вартості його створення.

2.2. Визначення трудомісткості розробки програмного забезпечення.

Тривалість розробки програмного продукту залежить від його обсягу, трудомісткості розробки, кваліфікації виконавців, а також планових термінів, визначених умовами ринку. Методом структурної аналогії по відповідних каталогах аналогів програмного забезпечення визначаємо обсяг програмних засобів, у тисячах умовних машинних команд програми аналога

У таблиці 4.1 представлені аналоги програмного забезпечення, функції яких, у більшому або меншому ступені, виконує розроблений програмний продукт.

Каталог аналогів

Таблиця 2.1

Найменування ПП	Обсяг функції ПП – V_o, усл. машинних командах.
1. ПП СУБД	2500 – 9800
2. Комплексні системи ведення БД	950 – 7430
3. ПП організації обчислювального процесу	13000 – 10200

Для нашого варіанта виділено сірим кольором.

Вибравши аналог ПП, що містить Vo в умовних машинних командах, трудомісткості визначати на основі табл.4.2

Таблиця 2.2

Обсяг ПП, тис.умов.машинних команд	Норма часу, люд/год
1.00	229
2.00	244
3.00	262

На підставі отриманого значення, по довіднику, визначається укрупнена норма часу на розробку аналога програмного забезпечення (коректується поправочним коефіцієнтом враховуючої умови розробки ПП, тобто в умовах комп'ютера, Кк=0,7÷0,8): $T^a = 229 \times 0,8 = 183,2$ (люд/годин).

Трудомісткість програмного продукту визначається по кожному етапу розробки окремо на підставі трудомісткості аналога з урахуванням складності розробки, ступеня новизни і ступеня використання в розробці стандартних модулів на підставі формул:

$$T_{ТЗ} = T^a p \times L_1 \times K_H \qquad (4.1)$$

$$T_{ТП} = T^a p \times L_2 \times K_H \qquad (4.2)$$

$$T_{РП} = T^a p \times L_3 \times K_H \times K_T \qquad (4.3)$$

Для розрахунку необхідні наступні коефіцієнти:

Li – питома вага і-го етапу розробки (див. табл. 4.2.);

Кн – поправочний коефіцієнт, що враховує ступінь новизни (див. табл. 4.3.);Кт – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм (див. табл. 4.4.).

Таблиця 4.2.Значення питомих коефіцієнтів трудомісткості стадії в загальній трудомісткості розробки ПП.

Код стадії	Ступінь новизни		
	А	Б	В
ТЗ (L_1)	0,15	0,12	0,12
ТП (L_2)	0,16	0,15	0,11
РП (L_3)	0,55	0,58	0,61

Для нашого варіанта виділено сірим кольором.

Таблиця 4.3. Значення поправочного коефіцієнта, що враховує ступінь новизни

Кодступеня новизни	Ступінь новизни	Значення $К_н$
А	Принципово нові ПО	1,75 – 1,2
Б	ПО – розвиток визначеного параметричного ряду	1,0 – 0,8
В	ПО маючий аналог	0,7

Для нашого варіанта виділено сірим кольором.

Таблиця 4.4. Значення коефіцієнта ступеня використання в розробці типових програм

Ступінь охоплення реалізованих функцій розроблювального ПО типовими програмами, %	Значення $К_т$
60 і вище	0,6
40-60	0,7
20-40	0,8
До 20	0,9

Для нашого варіанта виділено сірим кольором.

Тепер розраховуємо трудомісткість по кожному етапу окремо:

Трудомісткість технічного завдання

$$Т_{тз}=Т^а *L_1*К_н= 183,2 *0,12*0,7= 15,39 \text{ (люд/годин)} \quad (4.1)$$

Трудомісткість розробки технічного проекту

$$Т_{тп}=Т^а *L_2*К_н =183.2 *0,11*0,7 = 17,42 \text{ (люд/годин)} \quad (4.2)$$

Трудомісткість розробки робочого проекту

$$Т_{рп}=Т^а *L_3*К_н*К_т =183,2 *0,61*0,7*0,7=54,76 \text{ (люд/годин)} \quad (4.3)$$

Для подальших розрахунків визначили кількість папера, витраченого на кожен етап: технічне завдання $N_{тз}$= 2 (стор), розробка ТП $N_{тп}$=18 (стор),

розробка робочого проекту $N_{рп}$=25(стор), пояснювальна записка відповідно $N_{пз}$= 10 (стор) Розрахунок зведений у таблицю 4.5

Таблиця 4.5. Розрахунок трудомісткості ПП

Найменування етапів	Розрахунок, годин.		
1	2	3	4
1.ТЗ	$T_{Pтз}$=15,39	$T_{кк}=0,7*N_{тз}$= 0,7*2=1,4	$T_{нк}=0,15*N_{тз}$=0,15*2=0,30
2.Розробка ТП	$T_{Pтп}$=14,12	$T_{кк}=0,7*N_{тп}$=0,7*18=12,6	$T_{нк}=0,15*N_{тп}$=0,15*18=2,7
3.Розробка РП	$T_{Pрп}$=54,76	$T_{кк}=0,7*N_{рп}$=0,7*25=17,5	$T_{нк}=0,15*N_{рп}$=0,15*25=3.8
4.Розробка ПЗ	$T_{пз}=1,5*$ $*N_{пз}=1,5*10$ =15	$T_{кк}=0,7*N_{тз}$=0,7*10=7	$T_{нк}=0,15*N_{пз}$=0,15*10 =1,5
Усього, в т.ч.:	ΣT=146,1		
- на розробку	$ΣT_р$=99,3		
- контроль керівника		$ΣT_{кк}$=38,5	
- нормоконтроль			$ΣT_{нк}$=8,3

2.3 Розрахунок ціни програмного продукту.

У цьому розділі для визначення ціни розраховуємо основну заробітну плату виконавців, матеріальні витрати, вартість машино – години і витрати на розробку ПО. Розрахунок основної заробітної плати виконавців приведений у таблиці 4.6. Відповідно до статті 8 «Закону про Державний бюджет України на 2022» встановлено мінімальну заробітну плату у місячному розмірі з 1 січня 2022 року - 6500 гривень; мінімальну погодинну тарифну ставку – 39.26 грн.

Таблиця 4.6 Розрахунок основної заробітної плати виконавців.

Найменування робіт	Трудомісткість робіт, години	Погодинна тарифна ставка, грн.	Розрахунок, грн.
1.Розробка ПП	99,3	39.26	3898,52
2.Контроль керівника	38,5	60.00	2310
3.Нормоконт-роль	8,3	60.00	498

Усього	-	-	ΣЗо= 6706,52

Зробимо розрахунок матеріальних витрат на розробку ПП. Розрахунок зведемо в таблицю 4.7

Таблиця 4.7 Розрахунок матеріальних витрат на розробку ПО

Найменування матеріальних витрат	Тип, модель	Кількість	Ціна одиниці, грн.	Вартість, грн.
Папір	Лист А4	50	2,5	125
Разом	-	-	-	$В_{м\,і}$ =
Транспортно– заготівельні витрати (10%)				$Втр\_з = 0{,}1 \times Вм1$= 12,5
Усього				$В_м = В_{мі} + В_{тр\_з}$= 137,5

На підставі отриманих даних по окремих статтях витрат складена калькуляція планової собівартості в цілому ПП за формою, приведеною в таблиці 4.8.

Таблиця 4.8. Розрахунок статей витрат планової собівартості

Стаття витрат	Значення, грн.	Формула розрахунку
1. Матеріали	137,5	$В_м$ (див. табл. 4.7)
2. Основна заробітна плата	6706,52	$З_о$ (див. табл. 4.6)
3.Додаткова заробітна плата	1005,98	$Зд = 0{,}15 \times Зо$ =0,15*6706,52
4.Відрахування до єдиного фонду соціального внеску	1696,75	$Вє.с.в. = 0{,}22 \times (Зо + Зд)$ =0,22* (6706,52+1005,98)
5. Накладні витрати	2011,96	$Внак. = 0{,}3 \times Зо$ =0,3* 6706,52
6. Повна собівартість	11558,71	$С_{пов}$= $В_м$ +$З_о$ + $З_д$ +Вє.с.в. + Внак.=

Розмір прибутку, що включається в ціну, визначаємо по наступній формулі:

$П=(С_п*Р)/100=$ (11558,71 *10)/100=1155,87 грн (4.4)

Де р – плановий рівень рентабельності (10-15%).

Оптова ціна (кошторисна вартість) визначається по формулі:

$Ц_о = С_{пов}+П =$ 11558,71+ 1155,87 грн =12714,58 грн (4.5)

Податок на додану вартість визначаємо по наступній формулі:

ПДВ = 0.2 * $Ц_о$ = 0,2*12714,58 = 2542,92 грн (4.6)

Виходячи з отриманих даних, ціна реалізації розробленого програмного продукту на основі наступної формули, становитиме:

$Ц_р = Ц_о + ПДВ = 12714,58 + 2542,92 = 15257,50$ грн (4.7)

Цр = Цнір + ПДВ Цр = 15394,02+3078,80 = 18472,82 грн.

3 ОХОРОНА ПРАЦІ

Вступ

Охорона праці як невід ‘ємна складова створення безпеки життєдіяльності людини в умовах виробництва поширюється на всі підприємства, установи і організації незалежно від форм їх власності та видів діяльності, на усіх громадян, які працюють, тощо. Тому за порушення організації охорони праці в однаковій мірі несуть відповідальність перед законом як роботодавець (власник), так і працівник (виконавець).

Розвиток науки і техніки привніс у працю людини свої особливості: вона стала більш продуктивною, але натомість значно зросла ціна помилок, все тяжчими стали наслідки аварій та нещасних випадків.

Міжнародна організація праці у Всесвітній день охорони праці, який щорічно відзначається 28 квітня, представляє доповідь про стан безпеки праці у країнах світу в цифрах і фактах. В одній з останніх зазначено, що безпечні умови праці є, насамперед, економічно вигідними. Тому її фахівці переконані, що впровадження найсуворіших норм охорони праці буде значною мірою відповідати інтересам кожного працівника, кожного роботодавця, кожної країни загалом.

Значення охорони праці полягає в тому, що саме вона є головною умовою збереження здоров'я та захисту людини від впливу шкідливих чинників виробничого середовища.

Дипломним проектом передбачається розробка запитів для обміну даними між сервером і базою даних . Тому для розгляду беремо робоче місце програміста.

Це середовище у приміщеннях (офісах) в основному характеризується такими фізичними параметрами, як температура, вологість та електричний опір підлоги. Фізико-хімічні показники включають інформацію про вміст у

повітрі іонів та різноманітних забруднювачів, а також деякі інші якісні характеристики середовища

3. 1 Аналіз небезпечних та шкідливих чинників, що впливають на працівника.

Безпечні умови праці на підприємстві досягаються за рахунок забезпечення безпеки виробничих процесів, які обґрунтовані і прийняті в технологічній частині дипломного проекту.

Для установлення можливого впливу на здоров'я користувачів ВДТ виробничих чинників має значення ряд якісних характеристик робочого середовища.

Це середовище у приміщеннях (офісах) в основному характеризується такими фізичними параметрами, як температура, вологість та електричний опір підлоги. Фізико-хімічні показники включають інформацію про вміст у повітрі іонів та різноманітних забруднювачів, а також деякі інші якісні характеристики середовища

3.2 Розробка заходів з охорони праці

3.2.1 Виробничі приміщення

Об'ємно-планувальні рішення будівель та приміщень для роботи з ПК мають відповідати вимогам ДСанПіН 3.3.2.007–98. Розміщення робочих місць з ПК у підвальних приміщеннях, на цокольних поверхах заборонено. Площа на одне робоче місце становить не менше ніж 6,0 м3, а об'єм – не менше ніж 20,0 м3.

Виробничі приміщення повинні обладнуватись шафами для зберігання документів, магнітних дисків, полицями, стелажами, тумбами тощо, з урахуванням вимог до площі приміщень. У приміщеннях з ПК слід щоденно робити вологе прибирання. Приміщення із ПК мають бути оснащені аптечками першої медичної допомоги.

При приміщеннях із ПК мають бути обладнані побутові приміщення для відпочинку під час роботи, кімната психологічного розвантаження.

3.2.2 Мікроклімат робочої зони працівників, вентиляція.

Мікроклімат середовища суттєво впливає на стан організму людини, її працездатність протягом робочого дня. Показники температури , відносної вологості, швидкості руху повітря, теплового випромінювання нагрітих поверхонь характеризують клімат внутрішнього середовища виробничого приміщення. В процесі трудової діяльності людина перебуває у тепловій взаємодії з виробничим середовищем.

За оптимальних мікрокліматичних умов в організмі працівника, завдяки терморегуляції, підтримується постійна температура тіла (36,6 °С). Кількість тепла, що утворюється в організмі, залежить від фізичного навантаження працівника, а рівень тепловіддачі – від мікрокліматичних умов виробничого середовища.

Згідно з діючими у нашій країні нормативними документами (ДСанПіН 3.3.2-007-980 у холодні періоди року температура повітря, швидкість його руху та відносна вологість повітря повинні відповідно складати: 22-24^{0}С; 0,1 м/с; 40-60%. Температура повітря може коливатись у межах від 21 до 25^{0}С при збереженні інших параметрів мікроклімату.

В теплі періоди року температура повітря, його рухливість та відносна вологість повинні відповідно становити: 23-25^{0}С; 0,1-0,2 м/с; 40-60 %.

Оптимальним рівнем аероіонізації у зоні дихання користувача вважається вміст легких аерофонів обох знаків від 150 до 5000 у 1 см^3 повітря.

Нормалізуючий вплив на склад повітря робочої зони справляють примусова вентиляція, захисні екрани (оснащені заземленням) та застосування іонізаторів.

3.2.3 Освітлення робочого місця, шум, вібрація

Приміщення для роботи з ПК повинні мати природне та штучне освітлення відповідно до СНиП П-4–79/ Природне освітлення має здійснюватись через світлові прорізи, орієнтовані переважно на північ чи

північний схід, і забезпечувати коефіцієнт природної освітленості (КПО) не нижче, ніж 1,5%.

Природне освітлення повинно здійснюватись у вигляді бічного освітлення та відповідати нормам ДБН В.2.5-28-2006 «Природне і штучне освітлення».

При природному освітленні слід передбачити наявність сонцезахисних засобів, що знижують перепади яскравостей між природним світлом та свіченням екрана ВДТ. З цією метою можна використовувати плівки з металізованим покриттям або жалюзі з вертикальними лам елями, що регулюються.

Штучне освітлення у приміщеннях з ВДТ треба здійснювати у вигляді комбінованої системи освітлення з використовуванням люмінесцентних джерел світла у світильниках загального освітлення. На робочих місцях має бути забезпечена рівномірна освітленість за допомогою переважно відбитого або розсіяного світлорозподілу. Світлових відблисків з клавіатури, екрана та від інших частин ВДТ у напрямку очей користувача не повинно бути.

Норма освітленості на робочих місцях складає 300-500лк.

3.2.4 Електробезпека.

Для попередження поразок електричним струмом необхідно чітко й у повному обсязі виконувати правила провадження робіт і правил технічної експлуатації. Необхідно виключити можливість доступу оператора до частин устаткування, що працює під небезпечною напругою, до неізольованим частинам, призначеним для роботи при малій напрузі й не підключеним до захисного заземлення, а також підводити електроживлення до ПЕОМ від розетки за допомогою спеціальної вилки із заземлюючим контактом.

3.2.5 Організація робочого місця користувача ПК

Обладнання і організація робочого місця з ВДТ мають забезпечувати відповідність конструкцій всіх елементів робочого місця та їх взаємного

розташування, ергономічним вимогам, з урахуванням характеру і особливостей трудової діяльності (ДСанПіН 3.3.2.-007-98).

Конструкція робочого місця й взаємне розташування всіх його елементів (сидіння, органи керування, засобу відображення інформації) відповідають антропометричним, фізіологічним і психологічним вимогам, а також характеру роботи. Конструкція робочих меблів дає можливість забезпечувати можливість індивідуального індивідуального регулювання їх відповідно до потреб працівника для підтримки зручної пози. Робочий стіл повинен бути пофарбований матовою фарбою. Дисплей розташований так, що його верхній край перебуває на рівні очей, на відстані близько 70 см, що укладається в припустимі рамки від 60 до 90 см. Частота мерехтіння екрана дорівнює 100 Гц, що відповідає умові більше 70 Гц.

Для зниження нервово-емоційного напруження, стомлювання, поліпшення мозкового кровообігу, подолання несприятливих наслідків гіподинамії, запобігання втомі доцільно впроваджувати виконання комплексу вправ, які наведені у Державних санітарних правилах і нормах роботи з візуальними терміналами електронно-обчислювальних машин ДСанПіН 3.3.2.007-98.

3. 3 Пожежна безпека.

Протипожежний захист приміщення забезпечується застосуванням автоматичної установки пожежної сигналізації, наявністю засобів пожежогасіння, застосуванням основних будівельних конструкцій будинку з регламентованими межами вогнестійкості, організацією своєчасної евакуації людей.

До засобів гасіння пожежі відносяться внутрішні пожежні водопроводи (крани –ПК), вогнегасники (вуглекислотні та порошкові), сухий пісок тощо.

У випадку виникнення пожежі необхідно відключити електроживлення, викликати по телефону 101 пожежну команду, евакуювати людей із приміщення відповідно до плану евакуації і приступити до ліквідації пожеж.

ВИСНОВКИ

У процесі роботи над дипломним проектом було поставлено мету розробки запитів системи БД управління спілкування з серверної частиною.

Приступаючи до автоматизації структури БД, було проведено передпроектне обстеження для врахування особливостей та вимог до управління даними.

1. Розглянуто існуючу технологію обробки інформації та виявлено її недоліки;

2. Сформульовані цілі та призначення автоматизованого варіанту вирішення комплексу завдань;

3. Формалізовані розрахунки;

4. Зроблено вибір технології проектування та технічного, програмного, інформаційного, а також технологічного забезпечення.

5. Було розроблено інформаційне та програмне забезпечення комплексу завдань, які дозволяють скоротити трудомісткість виконуваних робіт та вартісні витрати на їх виконання.

Система була розроблена на основі фреймворку ASP.NET MVC Core 6 з використанням СУБД MS SQL Server, а також ORM-системи Entity Framework Core.

Розроблена система надала такі переваги порівняно із звичайним документообігом:

1. Можливість оперативного контролю за достовірністю інформації;

2. Зменшення кількості можливих помилок при генеруванні похідних даних;

3. Можливість швидкого доступу до будь-яких даних;

4. Економія пам'яті в процесі складання запиту для зв'язування та управління даними звіту.

У ході виконання роботи всі цілі були досягнуті, всі поставлені завдання було вирішено.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Рихтер, Джефри. CLR via C#. Программирование на платформе Microsoft NET Framework 2.0 на языке C#. – Питер, Русская Редакция, 2007 г. – 656 с.

2. Троелсен, Эндрю. C# 2008 и платформа .NET 3.5 Framework. 4-е изд. - М.: Вильямс, 2009. – 1168 с.

3. Рихтер Джефри. Программирование на платформе Microsoft NET Framework. – Питер, Русская Редакция, 2005 г. – 486 с.

4. Новые возможности NET Framework [Электронный ресурс] / MSDN – Электронные данные – Режим доступа: http://msdn.microsoft.com/ru-ru/library/ms171868.aspx.

5. Джон Скит C# для профессионалов. Тонкости программирования. – М.:Вильямс, 2014. – 408 с.

6. Джозеф Албахари, Бен Албахари C# 6.0. Справочник. Полное описание языка– М.:Вильямс, 2016. – 1040 с.

7. Фримен А. ASP.NET 4.5 с примерами на C# 5.0 для профессионалов. – М.:Вильямс, 2014. – 1120 с.