Національний університет «Львівська політехніка»

Кафедра програмного забезпечення

**КУРСОВА РОБОТА**

**з дисципліни «Бази даних»**

на тему:

«База даних для обліку діяльності складу мережі супермаркетів»

Виконав: студент

спеціальності 121

«Інженерія програмного забезпечення»

групи П3-32

Романчук О.Г.

Керівник:

доцент кафедри програмного забезпечення, к.т.н., доцент Павич Н.Я.

Оцінка:

Національна шкала\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів\_\_\_\_\_\_\_Оцінка ECTS\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_ Павич Н.Я.

(підпис)

\_\_\_\_\_\_\_\_\_\_\_\_ Цимбалюк Т.М.

(підпис)

**ЗМІСТ**

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ ОБЛІКУ ДІЯЛЬНОСТІ СКЛАДУ МЕРЕЖІ СУПЕРМАРКЕТІВ.....…………………………………………………………………………...3

* 1. Опис предметної області ...............................………………………………………3
  2. Постановка завдання………………………………………………………………..4
  3. Специфікація вимог до системи обліку cкладу мережі супермаркетів.................................................................…………………….……….5

1.3.1. Вступ…………………………………………………………………………...5

1.3.2. Загальний опис….……………………………………………………………..6

РОЗДІЛ 2. ПРОЕКТУВАННЯ БАЗИ ДАНИХ……………………………………………….12

2.1. Концептуальне моделювання предметної області ………………………………12

2.2. Логічне проектування бази даних………………………………………………...19

2.3. Проектування типових запитів і транзакцій……………………………………..21

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ РОБОТИ З БАЗОЮ ДАНИХ…..23

3.1. Реалізація доступу до даних………………………………………………………23

3.2. Реалізація функціональних характеристик системи…………………………….25

3.2.1. Авторизація…………………………………………………………………...25

3.2.2. Вибірка даних…………………………………………………………………27

3.2.3. Додавання даних……………………………………………………………...28

3.2.4. Видалення даних……………………………………………………………...28

3.2.5. Редагування даних……………………………………………………………29

ВИСНОВКИ…………………………………………………………………………………….30

СПИСОК ЛІТЕРАТУРИ…………………………………………………………………….....31

**РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ ОБЛІКУ ДІЯЛЬНОСТІ МЕРЕЖІ АВТОЗАПРАВНИХ СТАНІВ**

* 1. **Опис предметної області “Керування** **складом мережі супермаркетів”**

Продаж і організація товарів мережі супермаркетів як і будь-яка сфера бізнесу не може обійтися без програм для керування. Керування супермаркетом має на увазі облік товарів, виробників, складів, працівників та постачальників. Ведення обліку дозволяє не тільки контролювати матеріальну та сировинну базу, а й так само здійснювати контроль над діями персоналу.

Базові функції програми управління складом мережі супермаркетів: облік і контроль приходу та продажу продуктів, формування суворої звітності і документів, зокрема поставок та переміщень товарів між складами. Крім цього важливий критерій - централізоване управління мережею супермаркетів, швидка зміна цін і отримання інформації в найкоротші терміни, без залучення людини.

Головне завдання впровадження програмного забезпечення для керування складом мережі супермаркетів - оптимізувати роботу всіх складів, шляхом ефективного розподілу прав для персоналу. Зробити звітність швидкою, своєчасною і максимально точною. Подальша робота і аналіз діяльності окремих складів, допоможе виявити слабкі місця, не ефективні або збиткові послуги, в тому числі допоможе скоординувати роботу всієї мережі супермаркетів.

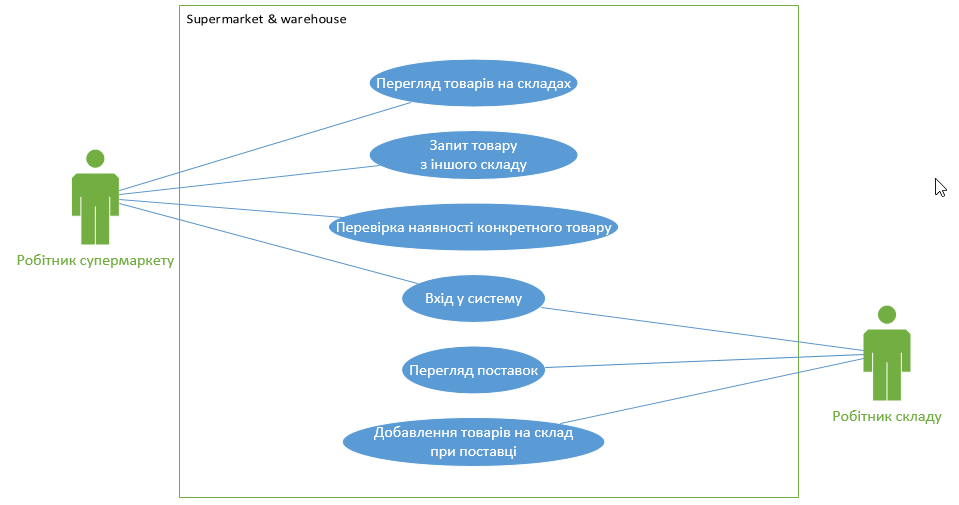
* 1. **Постановка завдання**

Основний функціонал програми – це перегляд інформації про товари на складі, поставки та постачальників. Основні сутності програми – це продукти та постачальники товарів, склади.

У системі буде два типи користувачів, а саме: робітник супермаркету та робітник складу.

Робітник супермаркету може переглядати товри на складах, оформлювати запит товару з іншого складу, перевіряти наявність конкретного товару.

В свою чергу робітник складу може переглядати інформацію поставок та добавляти товари на склад при поставці.

В ході постановки завдання була спроектована діаграма прецедентів на малюнку.

Мал.1.1 Діаграма прецедентів

* 1. **Специфікація вимог до системи обліку “ Керування складом мережі супермаркетів ”**
     1. **Вступ**

Призначенням даного продукту є автоматизація роботи супермаркету, можливість організації товарів на складі та у самому супермаркетів для працівників. Забезпечення всієї необхідної інформації про товари; про склади, їх розташування; про постачальників та виробників, їх цінову політику та контактну інформацію.

* + 1. **Загальний опис**

**1.3.2.1. Перспективи продукту**

Даний продукт являється самодостатньою системою, покликаний систематизувати роботу супермаркетів. Ідея виникла на основі систем управління великими мережами гіпермаркетів, але на відміну від таких систем вона повинна бути простою і зрозумілою навіть недосвідченим користувачам.

**1.3.2.2. Характеристика продукту**

Продукт є повністю незалежним і повністю самодостатнім програмним забезпеченням. Основні обмеження на продукт накладає реалізація бази даних та зв'язків у ній.

**1.3.2.3. Класи користувачів та їх характеристики**

Оскільки дана система є загальнодоступна, користувачами будуть працівники цієї організації, клієнти та звичайні користувачі. Поділ користувачів буде відбуватись за рівнями доступу, як наведено у наступній таблиці.

Таблиця 1.1. Класи користувачів

|  |  |
| --- | --- |
| Тип користувача | Права доступу |
| Робітник супермаркету | Можливість перегляду продуктів на складах. Можливість оформлення запиту на продукти з інших складів. Можливість перевірки наявності конкретного товару. |
| Робітник складу | Можливість перегляду поставок. Можливість добавлення товарів на склад при поставці. |

**1.3.2.4. Середовище функціонування**

Для функціонування системи необхідна платформа повинна відповідати наступним вимогам:

|  |  |
| --- | --- |
| Операційна система | ОС windows XP і вище |
| Процесор | Intel Pentium і вище |
| Оперативна пам’ять | 512 МБ |
| Обсяг вільної пам’яті | 300 МБ |

**1.3.2.5. Обмеження проектування та реалізації**

Продукт повинен бути реалізований впродовж 6 місяців після старту розробки. Система не повинна вимагати наявності додаткових програмних компонентів крім Microsoft SQL Server.

**1.3.2.6. Документація користувача**

У комплекті з системою надаватиметься набір користувацьких інструкцій, рекомендацій по роботі з системою згідно із стандартом IEEE Std 1063-2001 «IEEE Standard for Software User Documentation».

* + 1. **Характеристики системи**

**1.3.3.1. Авторизація користувачів**

Пріоритет *високий*. Авторизація користувача відповідно до його типу.

**Послідовність дія/відгук**

**База даних**

**Форма**

**входу**

**Користувач**

**Функціональні вимоги**

REQ-1: Вибір типу користувача при вході в систему.

**1.3.3.2. Керування товарами**

Пріоритет *високий*. Додавання, редагування даних про товари.

**Послідовність дія/відгук**

**База даних**

**Форма редагування даних**

**Користувач**

**Функціональні вимоги**

REQ-1: Додавання товару на склад. Передбачити можливість неправильного введення даних.

REQ-2: Змінювати кількість товару на складі.

**1.3.3.3. Керування клієнтами**

Пріоритет *високий*. Добавляння товарів на склад при поставці.

**Послідовність дія/відгук**

**База даних**

**Форма редагування даних**

**Користувач**

**Функціональні вимоги**

REQ-1: Добавляння товарів на склад при поставці. Передбачити можливість неправильного введення даних.

**1.3.3.4. Керування працівниками**

Пріоритет *високий*. Запит товару з іншого складу.

**Послідовність дія/відгук**

**База даних**

**Форма редагування даних**

**Користувач**

**Функціональні вимоги**

REQ-1: Оформлення запиту товару з іншого складу. Передбачити можливість неправильного введення даних. Одночасне видалення товарів зі складу, в якому беруть товар.

* + 1. **Вимоги зовнішніх інтерфейсів**

**1.3.4.1. Апаратні інтерфейси**

Комп'ютер чи смартфон на якому буде виконуватись дана програма повинен мати наступні апаратні характеристики:

* 32-х бітний або 64-х бітний процесор з тактовою частотою 1 ГГц, або більше;
* 512 ГБ ОЗП та 300 МБ ЗСД

**1.3.4.2. Програмні інтерфейси**

Для повноцінної роботи необхідно встановити наступні програмні компоненти:

* ОС Windows 7 / Windows 8 / Windows 10 / Linux / Mac OS / Android

**1.3.4.3. Комунікаційні інтерфейси**

Для коректної роботи ПК повинен мати порт RJ-45 або ж модуль Wi-Fi, а локальна мережа підтримувати стандарт Fast Ethernet.

* + 1. **Нефункціональні вимоги до проекту**

**1.3.5.1. Вимоги продуктивності**

Система повинна максимально ефективно використовувати апаратні ресурси платформи, на якій вона запущена. При виконанні запитів до бази даних не повинна виникати затримка.

**1.3.5.2. Вимоги надійності**

Система повинна виконувати періодичне збереження даних для того, що запобігти великим втратам у разі непередбачуваних ситуацій (перебої електроенергії тощо).

**1.3.5.3. Вимоги безпеки**

Кожен користувач має певний тип доступу прив’язаний до облікового запису.

**1.3.5.4. Атрибути якості програмного продукту**

Навчання роботі із системою повинно займати мінімальну кількість часу, система повинна бути стабільною та ефективно використовувати апаратні ресурси.

**РОЗДІЛ 2. ПРОЕКТУВАННЯ БАЗИ ДАНИХ**

## 2.1. Концептуальне моделювання предметної області

### 2.2.1. Опис сутностей

Задля вирішення задачі керування АЗС у БД було створено наступні сутності: «Клієнт», «Працівник», «Користувач», «Роль користувача», «Пальне», «Тип платежу», «Постачальник пального», «Продаж», «Станція», «Склад».

Сутність *Клієнт*

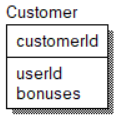


Рис.2.1. Таблиця «Клієнт»

Дана сутність містить інформацію про клієнтів АЗС:

* customerId (PK) — ідентифікаційний номер клієнта;
* userId (FK) – посилання на таблицю всіх користувачів, з якої витягуватиметься логін, пароль та роль користувача;
* bonuses – бонуси, що були нараховані даному клієнту.

Сутність *Працівник*

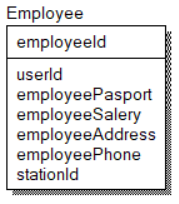


Рис.2.2. Таблиця «Працівник»

Дана сутність містить інформацію про працівників АЗС:

* employeeId (PK) — ідентифікаційний номер працівника;
* userId (FK) – посилання на таблицю всіх користувачів, з якої витягуватиметься логін, пароль та роль користувача;
* employeePasport – номер паспорта працівника;
* employeeSalery – зарплата працівника;
* employeeAdress – адреса, за якою проживає працівник;
* employeePhone – телефон працівника;
* stationId (FK) – ідентифікаційний номер станції, на якій працює працівник.

Сутність *Користувач*

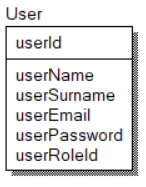


Рис.2.3. Таблиця «Користувач»

Дана сутність містить узагальнену інформацію про користувачів ПЗ:

* userId (PK) — ідентифікаційний номер користувача;
* userName – ім’я користувача;
* userSurname – прізвище користувача;
* userEmail – електронна пошта користувача;
* userPassword – пароль користувача;
* userRoleId (FK) – роль користувача.

Сутність *Пальне*

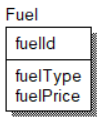


Рис.2.4. Таблиця «Пальне»

Дана сутність містить інформацію про пальне в мережі АЗС:

* fuelId (PK) — ідентифікаційний номер пального;
* fuelType – тип пального;
* fuelPrice – ціна пального в мережі АЗС.

Сутність *Тип оплати*

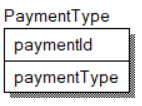


Рис.2.5. Таблиця «Тип оплати»

Дана сутність містить інформацію про тип оплати, за яким можна здійснити покупку на АЗС:

* paymentId (PK) — ідентифікаційний номер типу оплати;
* paymentType – тип оплати покупки.

Сутність *Постачальник*

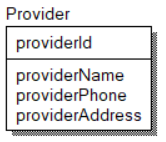


Рис.2.6. Таблиця «Постачальник»

Дана сутність містить інформацію про постачальників пального:

* providerId (PK) — ідентифікаційний номер постачальника;
* providerName – ім’я постачальника;
* providerPhone – телефон постачальника;
* providerAddress – адреса головного офісу постачальника.

Сутність *Продаж*

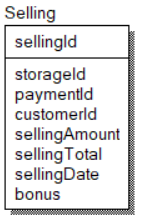


Рис.2.7. Таблиця «Продаж»

Дана сутність містить інформацію про здійснений продаж пального:

* sellingId (PK) — ідентифікаційний номер продажу;
* storageId (FK) – посилання на склад, з якого можна буде дізнатись тип пального та станцію на якій його продали;
* sellingAmount – кількість проданого пального;
* sellingTotal – загальна сума проданого пального;
* bonus – бонус нарахований клієнту за покупку;
* sellingDate – дата продажу;
* paymentId (FK) – тип платежу;
* customerId (FK) – клієнт, якому продали пальне.

Сутність *Станція*

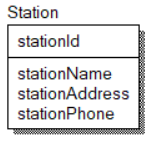


Рис.2.8. Таблиця «Станція»

Дана сутність містить інформацію про станції АЗС:

* stationId (PK) — ідентифікаційний номер станції;
* stationName – назва станції;
* stationAddress – адреса станції;
* stationPhone – телефон станції.

Сутність *Склад*

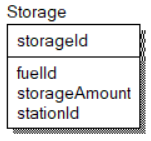


Рис.2.9. Таблиця «Склад»

Дана сутність містить інформацію про склади пального на АЗС:

* storageId (PK) — ідентифікаційний номер складу;
* fuelId (FK) – тип пального на складі;
* storageAmount – кількість пального даного типу на складі;
* stationId (FK) – посилання на станцію, на якій розташований склад.

Сутність *Постачальк пального*

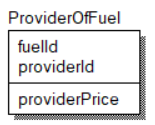


Рис.2.10. Таблиця «Постачальник пального»

Дана сутність містить інформацію про постачальників пального на АЗС:

* fuelId (FK) — ідентифікаційний номер пального;
* providerId (FK) – ідентифікаційний номер постачальника;
* providerPrice – ціна, за якою можна купити пальне у постачальника.

Сутність *Роль користувача*

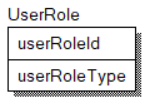


Рис.2.11. Таблиця «Роль користувача»

Дана сутність містить інформацію про ролі користувачів в системі:

* userRoleId (PK) — ідентифікаційний номер ролі користувача;
* userRoleType – роль користувача.

### 2.1.2. Опис зв’язків між сутностями

Між сутностями було встановлено наступні зв’язки:

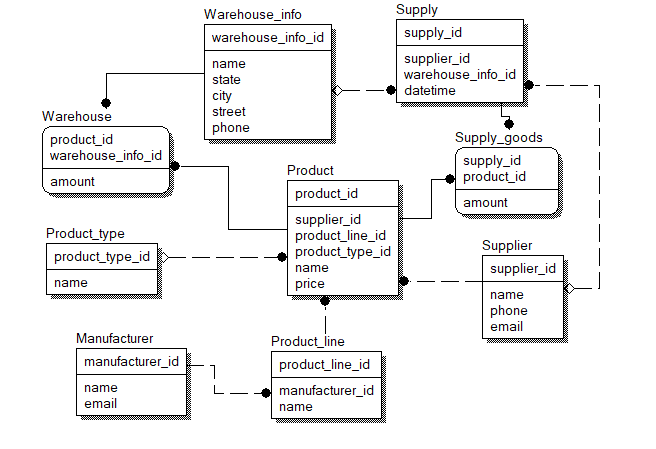
* Між сутностями *Тип платежу* та *Продаж* було встановлено зв’язок один-до-багатьох. (Продаж може мати лише один тип платежу, а платіж може мати багато продажів.)
* Між сутностями *Клієнт* та *Продаж* було вставновлено зв’язок один-до-багатьох. (Продаж може мати лише одного відповідного клієнта, в свою чергу клієнт може мати багато продажів.)
* Між сутностями *Склад* та *Продаж* було встановлено зв’язок один-до-багатьох.
* Між сутностями *Склад* та *Пальне* було встановлено зв’язок один-до-багатьох.
* Між сутностями *Пальне* та *Постачальник на фірмі* було встановлено зв’язок багато-до-багатьох. (Для цього створилась допоміжна таблиця *Постачальник пального.*)
* Між сутностями *Станція* та *Склад* було встановлено зв’язок один-до-багатьох. (Станція може мати декілька складів, а склад може належати тільки до однієї станції.)
* Між сутностями *Працівник* та *Станція* було встановлено зв’язок один-до-багатьох. Тому що на одній станції може працювати декілька працівників, але один працівник не може працювати на декількох станціях.
* Між сутностями *Працівник* та *Користувач* було встановлено зв’язок один-до-одного. Тому що працівник це і є користувач.
* Між сутностями *Клієнт* та *Станція* було встановлено зв’язок один-до-одного. Тому що клієнт це і є користувач.
* Між сутностями *Користувач* та *Роль користувача* було встановлено зв’язок багато-до-одного. Тому що один користувач може мати тільки одну роль, але одна роль може належати декільком користувачам.

## 2.2. Логічне проектування БД

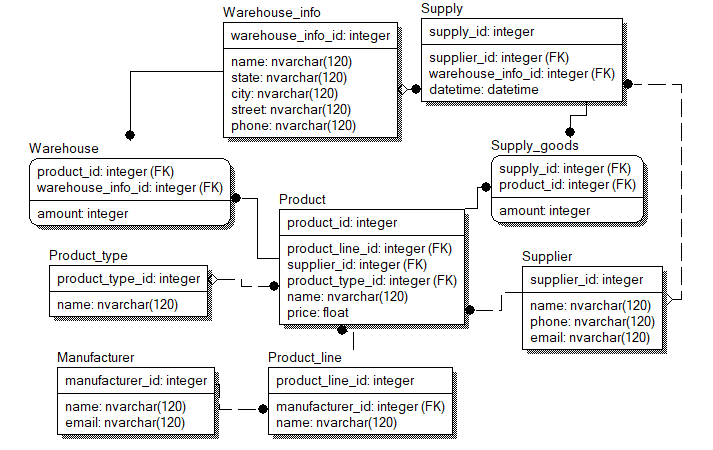
Логічна модель — це загальний погляд на дані, модель даних в пе­вній предметній області. Інколи її називають концептуальною моделлю даних. Такою моделлю і є модель типу сутність-зв' язок.

Фізична модель — це модель даних, реалізована у вигляді опису бази даних. Вона може бути реалізована на двох рівнях, а саме у вигляді трансформаційної моделі і моделі в системі керування базами даних.

В рамках даної роботи було створено логічну та фізичну моделі БД для обліку складу мережі супермаркетів.



Мал.2.12. Логічна модель БД



Мал.2.13. Фізична модель БД

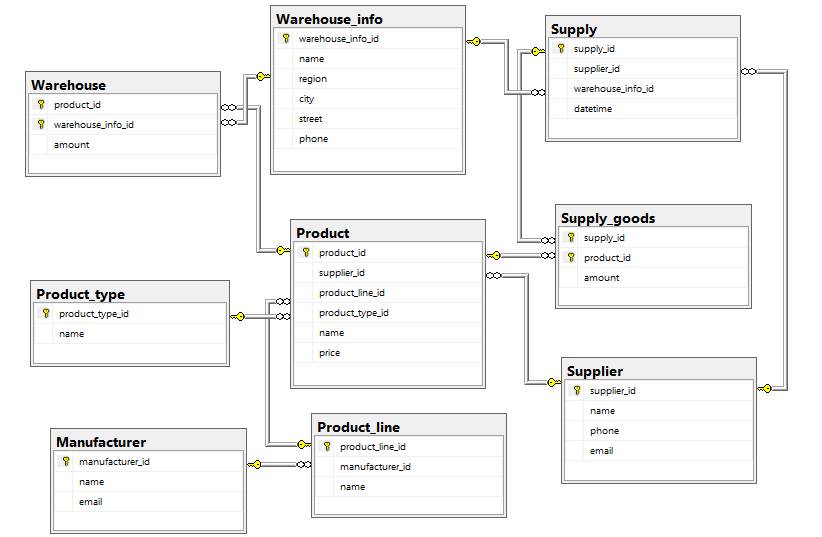


Рис.2.14. Діаграма побудована в середовищі SQL Server

Були додані наступні обмеження:

* Усі первинні ключі не можуть бути рівні NULL (Обмеження SQL) та менші 0.
* На усі атрибути з вторинними ключами було накладено обмеження з умовою більшою за 0.
* На усі атрибути, що позначають кількість компонентів було накладено обмеження з умовою більшою за 0.

## 2.3. Проектування типових запитів і транзакцій

В ході проектування системи управління базою даних було, спроектовано ряд запитів, а саме:

* Вибірка всіх наявних на станції типів пального (Таблиця «Station»).
* Запит на додавання нового типу пального (Таблиця «Fuel»).
* Запит на створення нового клієнта (Таблиця «Customer»).
* Запит на видалення даних про продаж кінцевому користувачу (Таблиця «Selling»).
* Запит на додавання нового постачальника (Таблиця «Provider»).
* Запит на оновлення даних про постачальника (Таблиця «Provider»).
* Запит на видалення даних про постачальника (Таблиця «Provider»).

Індекси в даній системі будуть використовуватися виключно за замовчуванням середовища SQL Server, через відсутність великих потоків даних для зберігання.

**РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ РОБОТИ З БАЗОЮ ДАНИХ**

## 3.1. Реалізація доступу до бази даних

Для реалізації доступу до даних БД обрано .NET Entity Framework. Це усуває необхідність в написання більшої частини коду для доступу до даних, який зазвичай потрібно розробникам. Використовуючи Entity Framework, розробники видають запити, що використовують LINQ, а потім витягують і маніпулюють даними, як сильно типізованими об'єктами. Entity Framework надає можливість взаємодії з об'єктами як за допомогою LINQ у вигляді LINQ to Entities, так і з використанням Entity SQL.

Реалізація платформи Entity Framework ORM надає такі послуги, як:

* відстеження змін;
* дозвіл ідентичності;
* відкладеності завантаження;
* переклад запитів.

Внаслідок цього, розробники можуть зосередитися на бізнес-логіці своїх програм, а не приділяти значну увагу основам доступу до даних. Entity Framework – це удосконалена технологія, що дає розробникам автоматизований механізм для доступу і зберігання даних в базі даних.

Entity Framework є корисним в трьох сценаріях. По-перше, якщо у вас уже є готова база даних або ж ви хочете спроектувати вашу базу даних на фоні інших частин програми. По-друге, якщо ви хочете зосередити свою увагу на доменних класах, а потім створити на основі існуючих доменнних класів базу даних. По-третє, якщо ви хочете створити схему бази даних на візуальному конструкторі, а потім на основі цього створити базу даних і класи.

Застосування Entity Framework дають наступні переваги:

* застосування можуть працювати з концептуальною моделлю в термінах наочної області – зокрема з успадкованими типами, складними елементами і зв'язками;
* зіставлення між концептуальною моделлю і схемою, специфічною для конкретного сховища, можуть мінятися без зміни коду застосування;
* застосування звільняються від жорстких залежностей від конкретного ядра СУБД або схеми зберігання;
* розробники мають можливість працювати з узгодженою моделлю об'єктів застосування, яка може бути зіставлена з різними схемами зберігання, які, можливо, реалізовані в різних системах управління даними;
* декілька концептуальних моделей можуть бути зіставлені з єдиною схемою зберігання;
* підтримка інтегрованих в мову запитів (LINQ) забезпечує під час компіляції перевірку синтаксису запиту щодо концептуальної моделі.

Платформа Entity Framework є компонентом .NET Framework, тому застосування Entity Framework можна запускати на будь-якому комп'ютері, на якому встановлений .NET Framework.

Зв’язок з сутностями БД у цьому проекті було здійснено за допомогою Entity Framework 6 у Visual Studio, який має назву CourseWork.

C:\Users\Ivan\Downloads\component.png

Мал.3.1. Діаграма компонентів

## 3.2. Реалізація функціональних характеристик системи

### 3.2.1. Авторизація

Авторизація реалізована наступним чином:

За допомогою елементу input користувач вводить свій email і пароль. Якщо введені дані вірні та при спробі під'єднатися до SQL Server, не виникло виняткової ситуації, тоді відкривається головне вікно програми, при завантаженні якого форма авторизації закривається, якщо ж виникла помилка, або введені дані були хибними згенерується повідомлення про помилку за допомогою елемента alert.

public async Task<User> GetUserByEmail(string email)

{

var user = (await userRepo.GetAsync(u => u.Email == email)).FirstOrDefault();

if (user == null)

return null;

if (user.EmailConfirmed == true)

{

var dto = \_mapper.Map<User, UserAccountDTO>(user);

return user;

}

return null;

}

public async Task<UserAccountDTO> Authenticate(string email, string password)

{

var user = await GetUserByEmail(email);

if (user == null)

return null;

var dto = \_mapper.Map<User, UserAccountDTO>(user);

if (password == dto.Password)

return dto;

return null;

}

Було створено клас User, який відповідає за авторизацію користувачів, редагування дозволів для них та перевірку цих дозволів.

public string UserId { get; set; }

public string UserName { get; set; }

public string UserSurname { get; set; }

public string UserEmail { get; set; }

public string UserPassword { get; set; }

public string UserRoleId { get; set; }

Клас User має у собі наступні поля: логін користувача, пароль користувача та його роль, які використовуються при автентифікації та авторизації.

Для авторизації користувача використовується JWT.

var response = new

{

access\_token = encodedJwt,

name = identity.Name,

id = user.Id,

email = user.Email,

role = identity.Claims

.Where(c => c.Type == ClaimTypes.Role)

.Select(c => c.Value)

};

string token = JsonConvert.SerializeObject(response, new JsonSerializerSettings { Formatting = Formatting.Indented });

return token;

### 3.2.2. Вибірка даних

Вибірка даних за id:

public async Task<TEntity> FindByIdAsync(int id)

{

return await \_dbSet.FindAsync(id);

}

Вибірка всіх даних з певної таблиці (TEntity):

public async Task<IEnumerable<TEntity>> GetAsync()

{

return await \_dbSet.AsNoTracking().ToListAsync();

}

Вибірка всіх даних, які відповідають заданій умові:

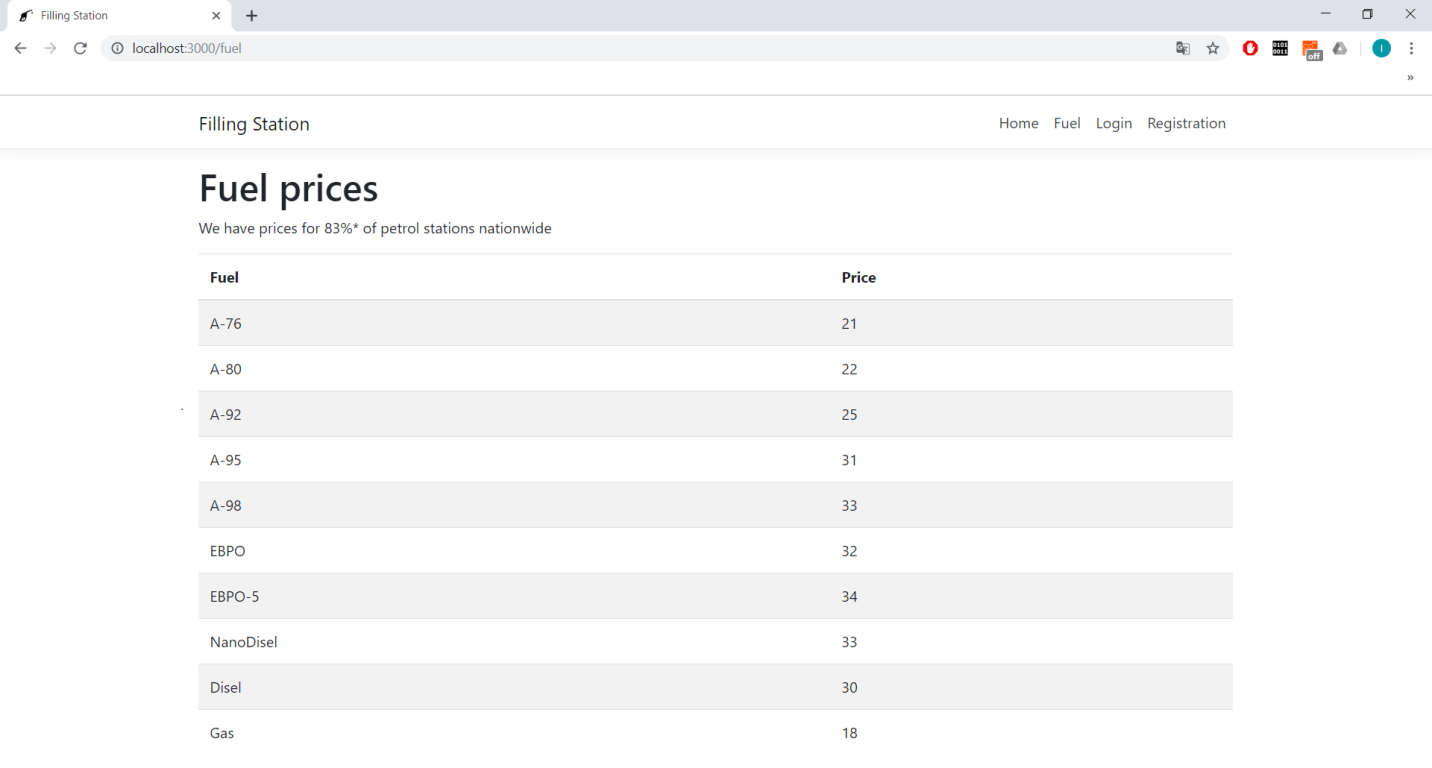
public async Task<IEnumerable<TEntity>> GetAsync(Expression<Func<TEntity, bool>> predicate)

{

return await \_dbSet.AsNoTracking().Where(predicate).ToListAsync();

}

На мал.3.2. зображена вибірка даних методом GetAsync():



Мал.3.2. Вибірка даних з таблиці Fuel

### 3.2.3. Додавання даних

public async Task CreateAsync(TEntity item)

{

\_dbSet.Add(item);

await \_context.SaveChangesAsync();

}

### 3.2.4. Видалення даних

public async Task RemoveAsync(TEntity item)

{

\_dbSet.Remove(item);

await \_context.SaveChangesAsync();

}

### 3.2.5. Редагування даних

public async Task UpdateAsync(TEntity item)

{

\_context.Entry(item).State = EntityState.Modified;

await \_context.SaveChangesAsync();

}

# ВИСНОВКИ

Результатом даної курсової роботи є база даних для керування АЗС та програмне забезпечення для роботи з нею.

В ході роботи над проектом було використано такі програмні продукти як Microsoft Visual Studio 2017 Сommunity Edition, Microsoft SQL Server Management Studio 2012. Visual Studio являє собою середовище розробки даного проекту. Сама ж база даних була реалізована за допомогою MS SQL Server. В якості технології зв’язку між додатком і базою даних використано технологію Entity Framework. Користувацький інтерфейс побудований на основі React і Redux.

Виконуючи цю курсову роботу я оволоділа навичками концептуального проектування баз даних, а також організації взаємодії бази даних з програмним забезпеченням за допомогою технології Entity Framework, навчилася проектувати типові запити та транзакції.

Основним недоліком розробленого додатку є неможливість його взаємодії з іншим типом бази даних без внесення необхідних правок.

В програмі реалізовано всі функціональні вимоги, описані в специфікації вимог до ПЗ.

# СПИСОК ЛІТЕРАТУРИ

1. SQL Server Tutorials. Web page: https://msdn.microsoft.com/en-us/library/ms167593(v=sql.105).aspx
2. C# Programming Guide. Web page: https://msdn.microsoft.com/en-us/library/67ef8sbd.aspx
3. ComboBox DataSource And Entity FrameWork. Web page: http://stackoverflow.com/questions/15898632/combobox-datasource-and-entity-framework
4. Beginning SQL Server for Developers: From Novice to Professional 2012 / Robin Dewson. – Appress,2012. – 823 ст.
5. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# /Дж.Рихтер. – Питер, 2016. – 896 ст.

**Додаток А. Скрипт створення БД**

CREATE TABLE Manufacturer

(

manufacturer\_id integer NOT NULL ,

name nvarchar(120) NULL ,

email nvarchar(120) NULL

)

Go

ALTER TABLE Manufacturer

ADD CONSTRAINT XPKManufacturer PRIMARY KEY CLUSTERED (manufacturer\_id ASC)

go

CREATE TABLE Product

(

product\_id integer NOT NULL ,

supplier\_id integer NOT NULL ,

product\_line\_id integer NOT NULL ,

product\_type\_id integer NULL ,

name nvarchar(120) NULL ,

price float NULL

)

go

ALTER TABLE Product

ADD CONSTRAINT XPKProduct PRIMARY KEY CLUSTERED (product\_id ASC)

go

CREATE TABLE Product\_line

(

product\_line\_id integer NOT NULL ,

manufacturer\_id integer NOT NULL ,

name nvarchar(120) NULL

)

go

ALTER TABLE Product\_line

ADD CONSTRAINT XPKProduct\_line PRIMARY KEY CLUSTERED (product\_line\_id ASC)

go

CREATE TABLE Product\_type

(

product\_type\_id integer NOT NULL ,

name nvarchar(120) NULL

)

go

ALTER TABLE Product\_type

ADD CONSTRAINT XPKProduct\_type PRIMARY KEY CLUSTERED (product\_type\_id ASC)

go

CREATE TABLE Supplier

(

supplier\_id integer NOT NULL ,

name nvarchar(120) NULL ,

phone nvarchar(120) NULL ,

email nvarchar(120) NULL

)

go

ALTER TABLE Supplier

ADD CONSTRAINT XPKSupplier PRIMARY KEY CLUSTERED (supplier\_id ASC)

go

CREATE TABLE Supply

(

supply\_id integer NOT NULL ,

supplier\_id integer NULL ,

warehouse\_info\_id integer NULL ,

datetime datetime NULL

)

go

ALTER TABLE Supply

ADD CONSTRAINT XPKSupply PRIMARY KEY CLUSTERED (supply\_id ASC)

go

CREATE TABLE Supply\_goods

(

supply\_id integer NOT NULL ,

product\_id integer NOT NULL ,

amount integer NULL

)

go

ALTER TABLE Supply\_goods

ADD CONSTRAINT XPKSupply\_goods PRIMARY KEY CLUSTERED (supply\_id ASC,product\_id ASC)

go

CREATE TABLE Warehouse

(

product\_id integer NOT NULL ,

warehouse\_info\_id integer NOT NULL ,

amount integer NULL

)

go

ALTER TABLE Warehouse

ADD CONSTRAINT XPKWarehouse PRIMARY KEY CLUSTERED (product\_id ASC,warehouse\_info\_id ASC)

go

CREATE TABLE Warehouse\_info

(

warehouse\_info\_id integer NOT NULL ,

name nvarchar(120) NULL ,

state nvarchar(120) NULL ,

city nvarchar(120) NULL ,

street nvarchar(120) NULL ,

phone nvarchar(120) NULL

)

go

ALTER TABLE Warehouse\_info

ADD CONSTRAINT XPKWarehouse\_info PRIMARY KEY CLUSTERED (warehouse\_info\_id ASC)

go

ALTER TABLE Product

ADD CONSTRAINT R\_11 FOREIGN KEY (supplier\_id) REFERENCES Supplier(supplier\_id)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE Product

ADD CONSTRAINT R\_13 FOREIGN KEY (product\_line\_id) REFERENCES Product\_line(product\_line\_id)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE Product

ADD CONSTRAINT R\_16 FOREIGN KEY (product\_type\_id) REFERENCES Product\_type(product\_type\_id)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE Product\_line

ADD CONSTRAINT R\_3 FOREIGN KEY (manufacturer\_id) REFERENCES Manufacturer(manufacturer\_id)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE Supply

ADD CONSTRAINT R\_14 FOREIGN KEY (supplier\_id) REFERENCES Supplier(supplier\_id)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE Supply

ADD CONSTRAINT R\_15 FOREIGN KEY (warehouse\_info\_id) REFERENCES Warehouse\_info(warehouse\_info\_id)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE Supply\_goods

ADD CONSTRAINT R\_8 FOREIGN KEY (supply\_id) REFERENCES Supply(supply\_id)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE Supply\_goods

ADD CONSTRAINT R\_10 FOREIGN KEY (product\_id) REFERENCES Product(product\_id)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE Warehouse

ADD CONSTRAINT R\_1 FOREIGN KEY (product\_id) REFERENCES Product(product\_id)

ON DELETE NO ACTION

ON UPDATE NO ACTION

Go

ALTER TABLE Warehouse

ADD CONSTRAINT R\_2 FOREIGN KEY (warehouse\_info\_id) REFERENCES Warehouse\_info(warehouse\_info\_id)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go