МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерних наук

ЯКОВІВ Ярина Василівна

Програмна система для обліку замовлень ательє з пошиття одягу/ Software system for orders accounting in the dressmaking studio

напрям підготовки: 6.050103 - Програмна інженерія
фахове спрямування - Програмне забезпечення систем

Бакалаврська дипломна робота

Виконала студентка групи
ПЗС-41
Я. В. Яковів

_____

Науковий керівник:
викладач ОЛІЙНИК І.С.

_____

Бакалаврську дипломну роботу
допущено до захисту:

"___" _____ 20___ р.

Завідувач кафедри
_____ **А. В. Пукас**

ТЕРНОПІЛЬ - 2016

РЕЗЮМЕ

Дипломна робота містить 92 сторінок, 17 таблиць, 50 рисунів, список використаних джерел із 5 найменувань.

Метою дипломної роботи є розробка інформаційної моделі та програмного додатку для обліку замовлень ательє з пошиття одягу.

Об'єктом дослідження є ательє з пошиття одягу.

Предметом дослідження є створення програмного додатку для обліку замовлень ательє з пошиття одяг. Методи розробки базуються на технології C#, сервер бази даних MS SQL Server.

Одержані результати полягають в розробці програмного додатку для обліку замовлень ательє з пошиття одягу.

Ключові слова: програмний додаток, облік замовлень, ательє з пошиття одягу.

SUMMARY

Thesis contains 92 pages, 17 tables, 50 figures, list of sources with 5 titles

The aim of the thesis is develop information models and software application for accounting for orders of clothing studio.

Object of research is clothing studio.

The subject of research is developing of software application for accounting for orders of clothing studio. Methods of developing technology based on C#, MS SQL Server database server.

The resulting is creating a software application for accounting for orders of clothing studio.

Keywords: software application, accounting orders, clothing studio.

ЗМІСТ

ВСТУП

Впродовж історії жінки завжди потребували і потребують одягу власного особливого стилю. Люди завжди хотіли відрізнятись. Кожен зокрема хотів створити власну річ, унікальну, проте не вміли цього.

Проте є майстри, які виконують цю непросту працю – створюють нові моделі одягу. Але такі завдання потребують хорошої організації і обліку усіх цих замовлень.

Тому саме система «Обліку замовлень ательє з пошиття одягу» - JJclothes - створена для обліку замовлень, де можна зробити замовлення, вказавши параметри клієнта, його вподобання щодо виробу, з якого матеріалу, кольору та фасону він буде зроблений.

Всі ці можливості об’єднує інформаційна система. Дослідження та проектування такої системи і є метою даної роботи.

Робота системи ательє є дуже актуальною в наш час, адже сьогодні важко уявити себе без сучасного одягу, новинок сезону. Люди, які слідкують за трендами, завжди хочуть бути попереду і мати власний стиль. Тому розробка системи обліку замовлень в ательє з пошиття одягу суттєво полегшує роботу, адже допомагає вчасно, якісно вести облік замовлень, що робили клієнти.

Мета роботи – розробити таку систему, яка матиме зручний, сучасний і зрозумілий інтерфейс, а також розробити її на англійській мові, щоб забезпечити продукт на ринку праці для усіх, хто знайомий з англійською мовою.

Галузь застосування даного продукту буде об’єднувати усі ательє з пошиття одягу, які бажатимуть вести облік своїх замовлень в електронному вигляді.

Предмет розробки міститиме трирівневу архітектуру, що включатиме клієнтський додаток, підключений до сервера додатків, який, в свою чергу, підключений до серверу бази даних.

Оскільки рівень клієнта - це власне наша програмна система для кінцевого користувача, то він не повинен мати прямих зв'язків з базою даних, не повинен бути навантаженим основною бізнес-логікою і зберігати стан програми. Рівень клієнта міститиме: інтерфейс програми, перевірка значень, що вводяться, на допустимість і відповідність формату, нескладні операції.

Другий рівень – рівень сервера, на ньому зосереджена більша частина бізнес-логіки. Також тут знаходиться програмний інтерфейс, що зв'язує клієнтські компоненти з прикладною логікою бази даних.

Сервер бази даних, який обраний MS SQL Server 2012, забезпечує зберігання даних і виноситься на третій рівень. Третій рівень являє собою базу даних, разом з збереженими процедурами, тригерами і схемою, яка описує застосунок в термінах реляційної моделі.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Коротка характеристика об'єкту управління

Розробка системи обліку замовлень для ательє є актуальною в цей час, адже існує низка різноманітних програмних систем, і багато з них схожі за своїм вмістом.

Крім того такі системи підвищують ефективність діяльності підприємства й полегшують роботу менеджерам із продажу тих або інших послуг.

На сьогодні існує конкуренція між компаніями, які здійснюють виконання замовлень клієнтів щодо одягу. Тому потрібно мати власну стратегію, щоб забезпечити довге та ефективне існування свого бізнесу на ринку послуг.

Потрібно зазначити, що достатня інформація та легкий доступ до неї приваблює клієнтів. В даному випадку йдеться про розробку автоматизованої системи обліку замовлень для ательє.

Ательє — салон індивідуального пошиття одягу, предметів декору, аксесуарів різних напрямків та стилів.

Система «Обліку замовлень ательє з пошиття одягу» створена для обліку замовлень, де можна зробити замовлення, вказавши параметри клієнта, його вподобання щодо виробу, з якого матеріалу, кольору та фасону він буде зроблений.

Основними напрямками діяльності ательє:

- індивідуальне пошиття чоловічого та жіночого одягу;
- пошиття партій одягу для торговельних точок;
- ремонт одягу будь-якої складності;
- підгонка виробу по фігурі;
- допомога у виборі пошиття одягу трендів сезону.

Визначення кількості виробничих підрозділів - організація має одну установу, в складі якої матимемо склад і майстерню.

Врахування місця розташування окремих підрозділів - підрозділ знаходиться в одному приміщенні.

Визначення потреби в наявності доскональних підрозділів:

- склад потрібен для того, щоб зберігати ресурси для розробки одягу;
- майстерня необхідна як місце розробки цього одягу.

Виділяють два рівні управління – адміністратора і майстра.

Основною проблемою бізнесу є створення такої системи, яка б дозволила вести облік замовлень усіх клієнтів одночасно, допомагала швидко вирішувати питання моделей виробів та зайняла достойне місце серед інших ательє.

1.2. Опис предметної області ательє з пошиття одягу

Предметною областю дипломної роботи є облік замовлень ательє з пошиття одягу. Для швидкого здійснення процесу пошуку та обліку замовлень необхідно розробити програмну систему.

На рисунку 1.1 наведено організаційну структуру об’єкта дослідження, а саме ательє з пошиття одягу.

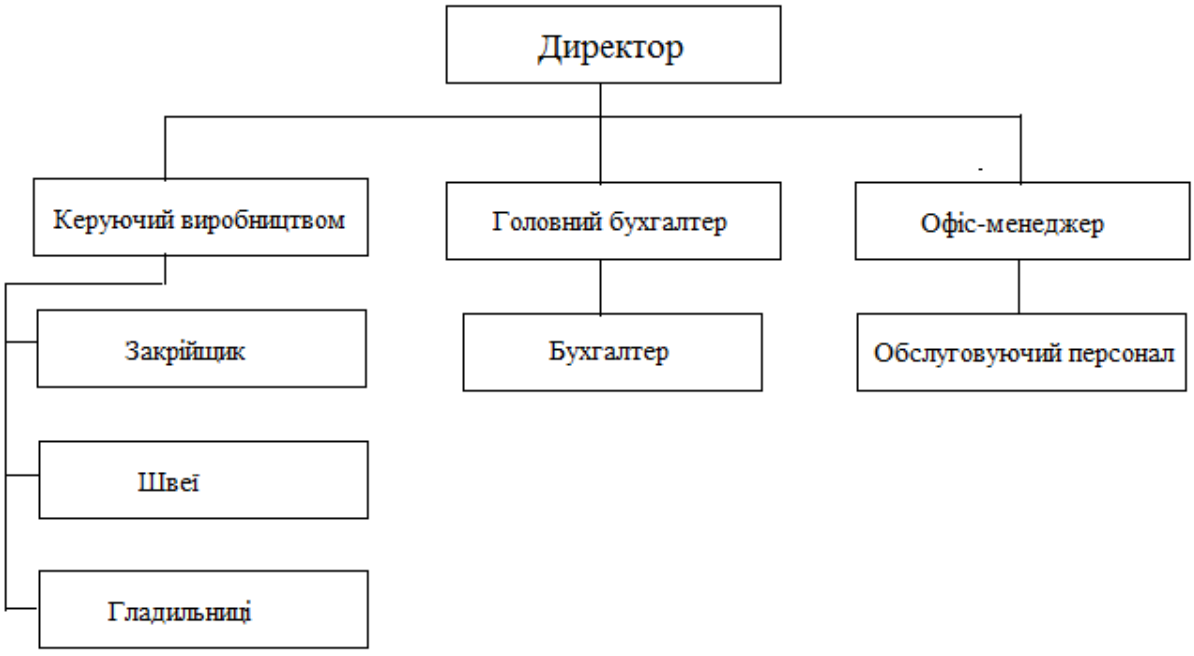

Рисунок 1.1 – Організаційна структура ательє

Для того, щоб повністю представити функціонал програмної системи, виділено такі основні бізнес-процеси:

- процес формування замовлення майстром;
- процес додавання змін (виробів, матеріалів,клієнтів) адміністратором;

- процес здійснення пошуку майстром.

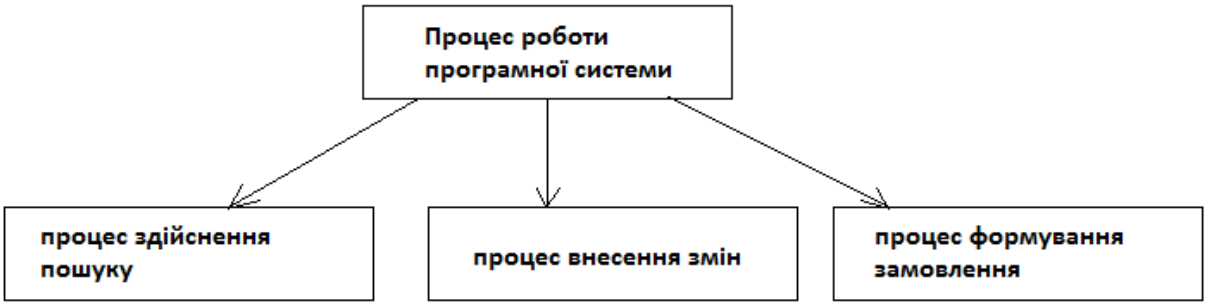

Рисунок 1.2 – Діаграма бізнес-процесів розроблюваного програмного продукту

Тепер детальніше основний бізнес-процес, що буде реалізовуватись у програмному продукті. На рисунку 1.3 зображено процес формування замовлення майстром.

Рисунок 1.3 – Процес формування замовлення майстром

Перше, що потрібно зробити майстру, - це сформувати замовлення, для подальшого додавання її в БД. Замовлення поділяється на такі особливості:

- тип – це сам об’єкт замовлення, наприклад: виріб, матеріал;
- характеристики, ними може виступати для прикладу дата виконання і завершення замовлення, а також ціна та колір;
- дані клієнта – це його прізвище, ім’я, контактні дані та параметри тіла.

Характеристику бізнес-процесу формування замовлення майстром наведено в таблиці 1.1.

Таблиця 1.1

Характеристика бізнес-процесу формування замовлення майстром

Назва характеристики	Значення характеристики
Ім'я бізнес-процесу	Формування замовлення майстром
Основні учасники	Майстер
Вхідна подія	Запит на формування замовлення
Вхідні документи	Дані про продукти і матеріали з бази даних
Вихідна подія	Перегляд сформованого замовлення
Вихідні документи	Сформоване замовлення в базу даних
Клієнт бізнес-процесу	Процес формування замовлення

Характеристику бізнес-процесу здійснення пошуку наведено в таблиці 1.2.

Таблиця 1.2

Характеристика бізнес-процесу здійснення пошуку

Назва характеристики	Значення характеристики
Ім'я бізнес-процесу	Здійснення пошуку
Основні учасники	Майстер
Вхідна подія	Запит на здійснення пошуку
Вхідні документи	Дані про замовлення з бази даних
Вихідна подія	Повідомлення про успішне здійснення пошуку
Вихідні документи	Відфільтрований пошук на екрані
Клієнт бізнес-процесу	Процес додавання категорій адміністратором

Характеристику бізнес-процесу внесення змін адміністратором наведено в таблиці 1.3.

Таблиця 1.3

Характеристика бізнес-процесу внесення змін адміністратором

Назва характеристики	Значення характеристики
Ім'я бізнес-процесу	Внесення змін адміністратором
Основні учасники	Адміністратор

Вхідна подія	Запит на внесення даних
Вхідні документи	Дані про продукт з бази даних
Вихідна подія	Повідомлення про успішно внесені дані
Вихідні документи	Збережений новий продукт в базу даних
Клієнт бізнес-процесу	Процес перегляду інформації

1.3. Огляд і аналіз існуючих аналогів

На сьогодні є достатньо програмних систем, за допомогою яких можна здійснити облік замовлень. Розгляньмо декілька з них.

На рисунку 1.4. наведено сторінку замовлення програми УСУ “Ателье”.

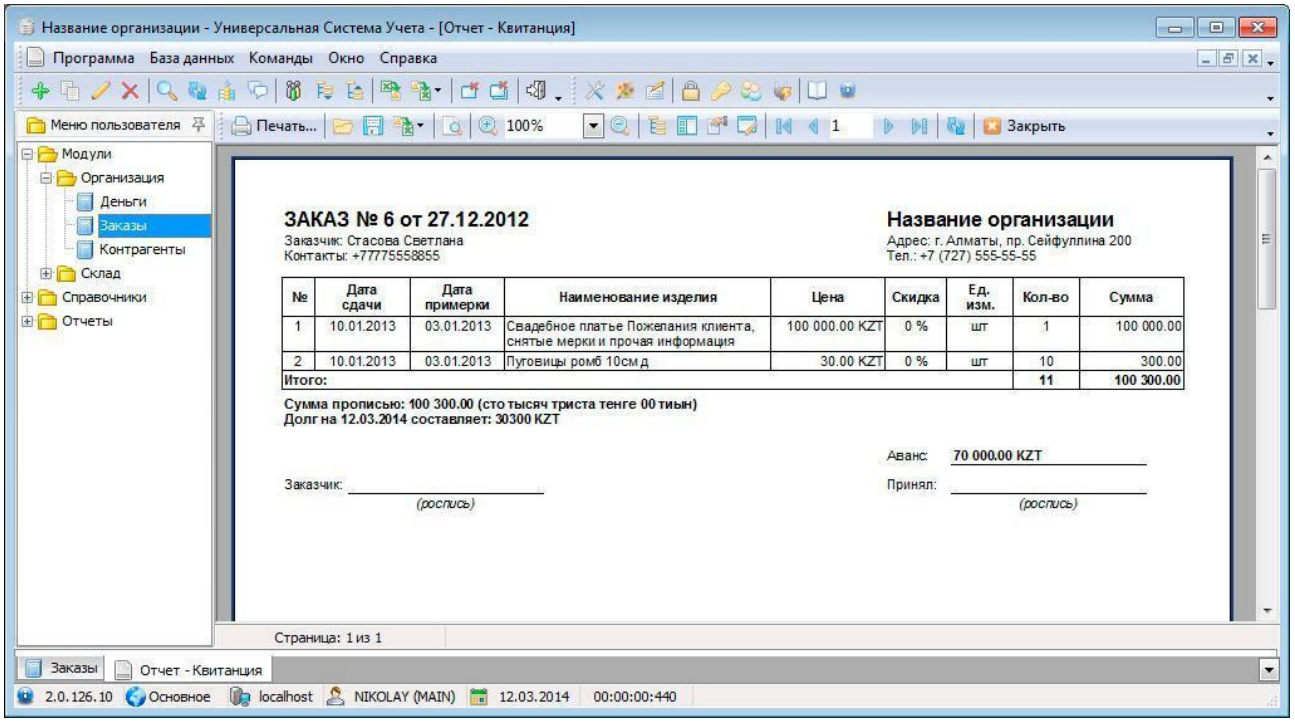

Рисунок 1.4 – Сторінка замовлень програми УСУ «Ателье»

Ця програма має дуже простий і достатньо зручний інтерфейс, за допомогою якого користувач може досить легко зорієнтуватися в подальших діях. Для здійснення обліку користувачу не потрібно проходити реєстрацію. Проте, щоб використовувати додаткові функції потрібно купити програмний продукт.

Дана програмна система має наступні функції:

- облік замовлень;

- історія замовлень;
- нарахування зарплати працівникам;
- контроль якості товарів.

Для порівняння розглянемо програмний продукт Prostoysoft “Ателье” (рисунок 1.5).

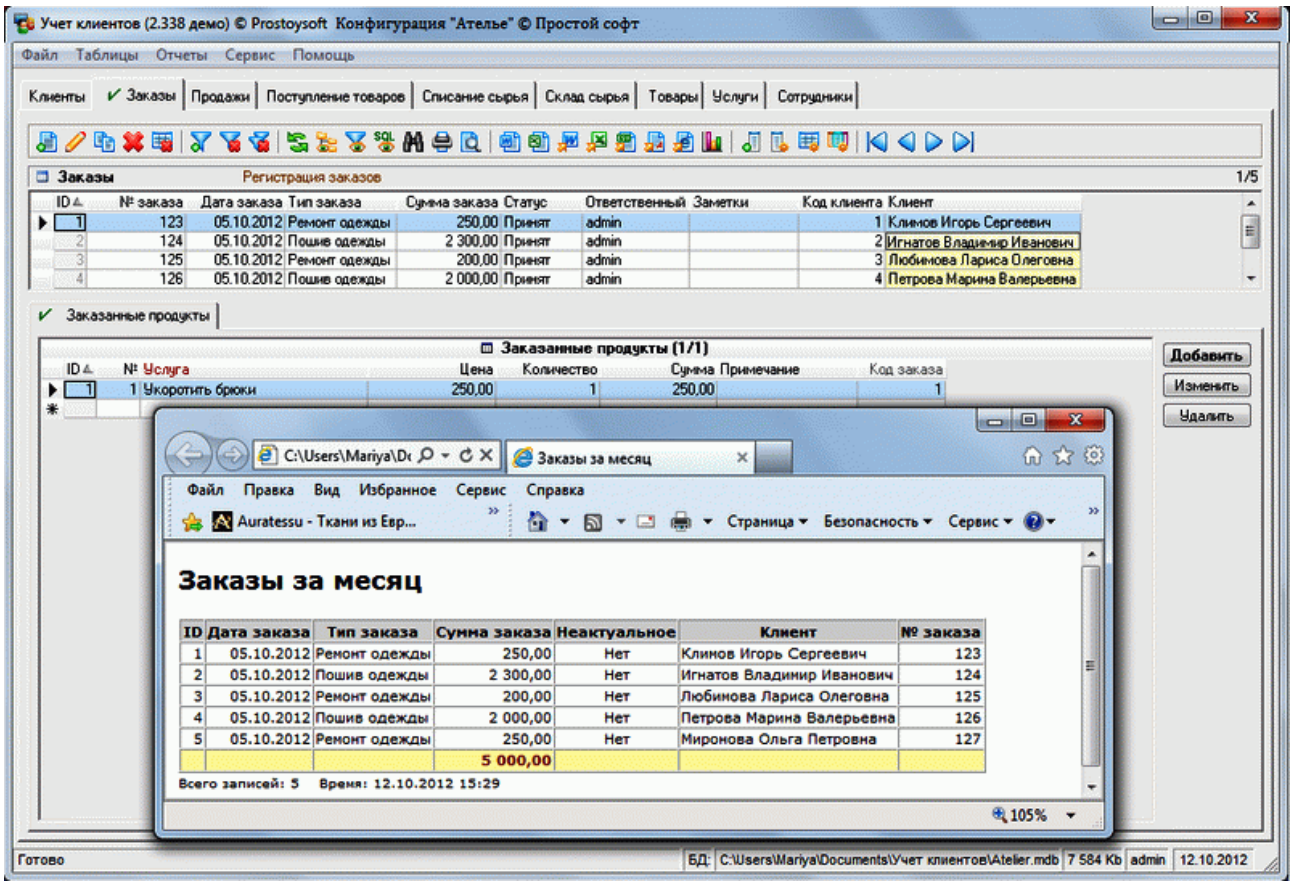

Рисунок 1.5 – Сторінка замовлень програми Prostoysoft “Ателье”

Продукт дозволяє проводити облік замовлень, реєструвати їх та проводити зміни. Цей продукт має багато різноманітних функцій, таких як:

- облік даних клієнтів;
- облік замовлень;
- облік товарів;
- оформлення продаж;
- пошук і групування даних.

Хоча програмний продукт і має велику кількість хороших функцій, він доступний лише у платній версії.

Розглянемо ще один програмний продукт **RedCafe** (рисунок 1.6).

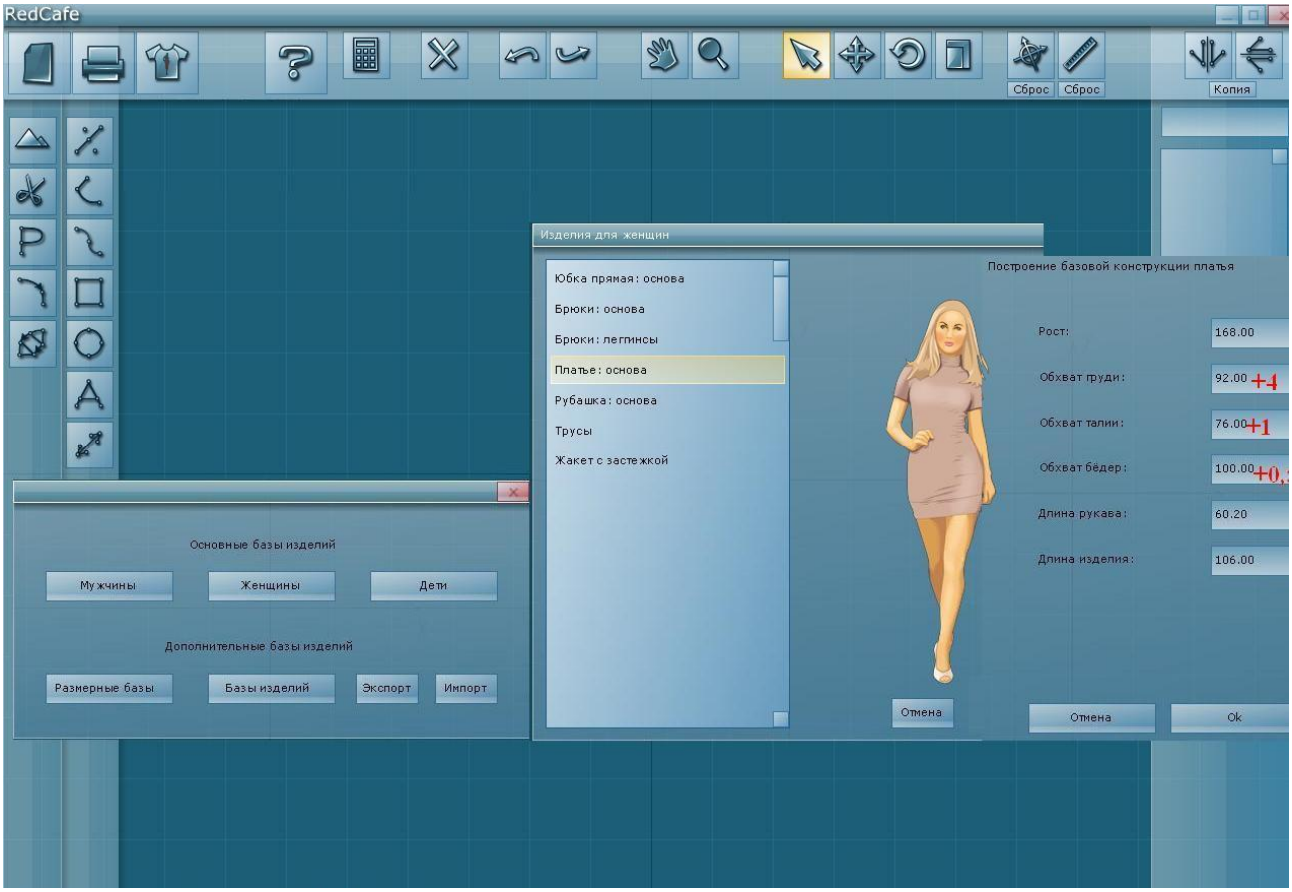

Рисунок 1.6 – Сторінка замовлень програми **RedCafe**

Інтерфейс даної програми є дуже зрозумілий і цікавий для користувача. Перевагою є доступність програми у офлайн-онлайн режимах, безкоштовній та платних версіях.

Основними функціями є:

- створення викрійок;
- редагування одягу;
- історія роботи;
- власний кабінет викрійок.

Оглянувши та зробивши аналіз існуючих аналогів, можна зробити висновок, що вищеперелічені програмні системи є хорошими для проведення обліку замовлень ательє для пошиття одягу. Кожна програма функціонує досить добре. Але, переглянувши функції кожного програмного продукту, можна побачити, що кожна система різна і орієнтована лише на російськомовних

користувачів, тому оптимальним рішенням даної проблеми є створення продукту для здійснення обліку замовлень, орієнтованих на іноземних користувачів зі зручним і сучасним інтерфейсом.

Здійснивши аналіз альтернатив, створено таблицю 1.4, де відображається порівняльна характеристика аналогів.

Таблиця 1.4

Порівняльна характеристика програмних продуктів

Порівняльна характеристика програмних продуктів			
Фірма-розробник	УСУ - Универсальная Система Учета	Общество с ограниченной ответственность ю «Редкафе»	ООО "Простой софт"
Назва програмного продукту	Программа «Ателье»	RedCafe	Конфигурация "Ателье"
Версії продукту	2.0.123.10 2.0.126.10	dShape v.0.0.1 dShape v.0.4.0 RedCafe v.0.5.0 RedCafe v.1.0.0 RedCafe v.1.3.2	1.18 1.136 *2.0* 2.236 2.829
Допомога користувачу	Відео-огляд про програму та презентація	Повністю наявна вся документація по роботі з програмою	Сайт з допоміжними фото та зауваженнями
Ціна	Існує платна і демо-версія	Онлайн, офлайн версії. Платна і безплатна версії	Тільки платна версія
Сайт, де можна скачати	http://usu.kz/app_ate le.php	http://redcafestore .com/download.ht ml	http://prostoysof t.ru/Atelier.htm

1.4. Специфікація вимог до системи

Специфікація вимог для програмної системи - це повний опис поведінки системи, що розробляється. Вона включає множину прецедентів, які описують всі взаємодії, які користувачі мають з програмним забезпеченням. Прецеденти

також відомі як функціональні вимоги. На додачу до прецедентів також включає нефункціональні вимоги. Нефункціональні вимоги є вимогами, які накладають обмеження на проект чи реалізацію. Специфікація вимог до системи включає: глосарій проекту, опис варіантів використання.

У додатку А подано глосарій основних використовуваних термінів при створенні системи обліку замовлень для ательє з пошиття одягу

На рисунку 1.7. наведено діаграму варіантів використання для акторів системи (майстер, адміністратор).

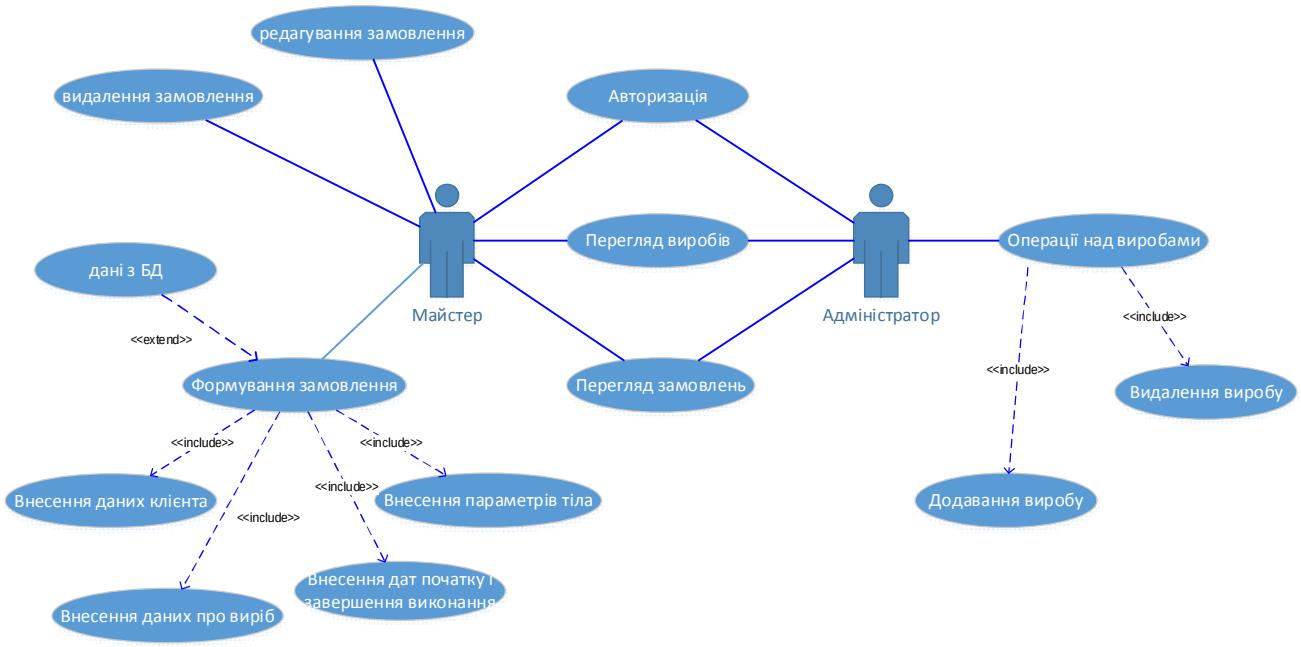

Рисунок 1.7 – Діаграма варіантів використання

Наступним кроком у визначенні специфікацій вимог до системи буде опис варіантів використання.

Варіанти використання представленні у таблицях 1.5-1.12.

Варіант використання «Авторизація» подано у таблиці 1.5.

Таблиця 1.5

Варіант використання «Авторизація»

Контекст використання	Авторизація
Дійові особи	Неавторизований майстер, адміністратор
Передумова	Користувач знаходиться на сторінці авторизації
Тригер	Відкрита сторінка для авторизації
Сценарій	1.Заповнити поле «Enter login»;

	2.Заповнити поле «Enter password»; 3.Натиснути кнопку «Login»
Пост-умова	Майстер, адміністратор авторизований

Варіант використання «Перегляд виробів» подано у таблиці 1.6.

Таблиця 1.6

Варіант використання «Перегляд виробів»

Контекст використання	Перегляд виробів
Дійові особи	Майстер, адміністратор
Передумова	Авторизація майстра, адміністратора
Тригер	Відкрита головна сторінка
Сценарій	1.Клікнути на вкладку “View Products”;
Пост-умова	Відкрита сторінка з виробами

Варіант використання «Перегляд замовлень» подано у таблиці 1.7.

Таблиця 1.7

Варіант використання «Перегляд замовлень»

Контекст використання	Перегляд замовлень
Дійові особи	Майстер, адміністратор
Передумова	Авторизація майстра, адміністратора
Тригер	Відкрита головна сторінка
Сценарій	1.Клікнути на вкладку “View Orders”;
Пост-умова	Відкрита сторінка з замовленнями

Варіант використання «Додавання виробу» подано у таблиці 1.8.

Таблиця 1.8

Варіант використання «Додавання виробу»

Контекст використання	Додавання виробу
Дійові особи	Адміністратор
Передумова	Авторизація адміністратора
Тригер	Відкрита сторінка виробів
Сценарій	1.Заповнити поле «Name»;

	2.Вибрати зі списку «Material»; 3. Заповнити поле «Image»; 4. Вибрати зі списку «Type»; 5. Заповнити поле «Price»; 6. Натиснути кнопку «Add»
Пост-умова	Збереження даних в БД

Варіант використання «Видалення виробу» подано у таблиці 1.19.

Таблиця 1.19

Варіант використання «Видалення виробу»

Контекст використання	Видалення виробу
Дійові особи	Адміністратор
Передумова	Авторизація адміністратора
Тригер	Відкрита сторінка виробів
Сценарій	1.Натиснути на кнопку «Delete»; 2. Обрати бажаний продукт; 2. Натиснути на кнопку «Delete»
Пост-умова	Збереження даних в БД

Варіант використання «Формування замовлення» подано у таблиці 1.10.

Таблиця 1.10

Варіант використання «Формування замовлення»

Контекст використання	Формування замовлення
Дійові особи	Майстер
Передумова	Авторизація майстра
Тригер	Відкрита сторінка замовлень

Сценарій	1. Обрати дату початку замовлення; 2. Обрати дату завершення замовлення; 3. Заповнити поле «First, Last Name»; 4.Заповнити поле «Email»; 5. Заповнити поле «Size 6. Заповнити поле «Product»; 7. Заповнити поле «Material»; 8. Заповнити поле «Color»; 9. Заповнити поле «Count»; 10. Натиснути кнопку «Price»; 11. Натиснути кнопку «Save»
Пост-умова	Збереження даних в БД

Варіант використання «Редагування замовлення» подано у таблиці 1.11.

Таблиця 1.11

Варіант використання «Редагування замовлення»

Контекст використання	Редагування замовлення
Дійові особи	Майстер
Передумова	Авторизація майстра
Тригер	Відкрита сторінка замовлень
Сценарій	1. Обрати дату початку замовлення; 2. Обрати дату завершення замовлення; 3. Натиснути кнопку «Edit»; 4. Заповнити поле «First, Last Name»; 5.Заповнити поле «Email»; 6. Заповнити поле «Size 7. Заповнити поле «Product»; 8. Заповнити поле «Material»; 9. Заповнити поле «Color»; 10. Заповнити поле «Count»; 11. Натиснути кнопку «Price»; 12. Натиснути кнопку «Save»
Пост-умова	Збереження даних в БД

Варіант використання «Видалення замовлення» подано у таблиці 1.12.

Таблиця 1.12

Варіант використання «Видалення замовлення»

Контекст використання	Видалення замовлення
Дійові особи	Майстер
Передумова	Авторизація майстра
Тригер	Відкрита сторінка перегляду замовлень
Сценарій	1. Натиснути на поле вибраного замовлення; 2. Натиснути на кнопку «Delete»; 3. Натиснути на кнопку «Yes»
Пост-умова	Збереження даних в БД

На рисунку 1.8 наведено розкадровку варіантів використання прототипу для функції «Авторизація».

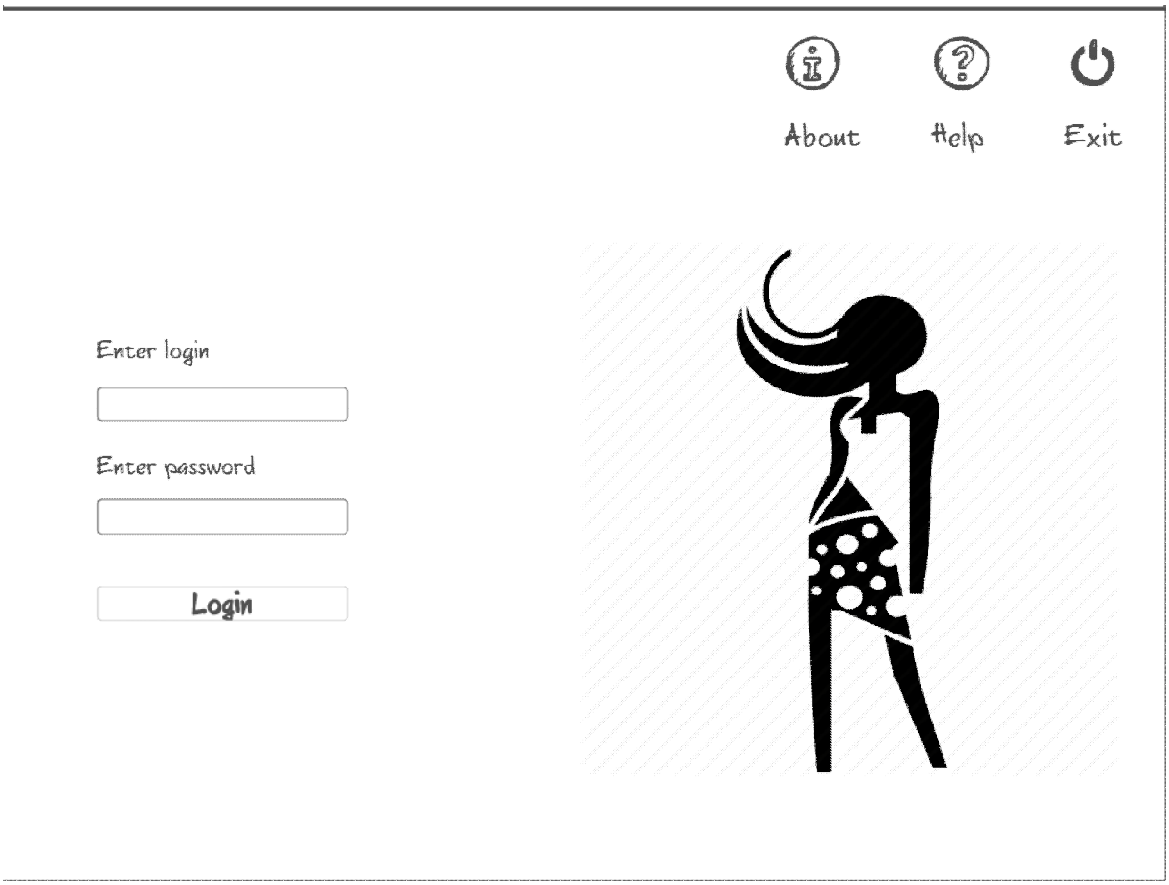

Рисунок 1.8 – прототип для функції «Авторизація»

Прототип для функції «Перегляд виробів» на рисунку 1.9

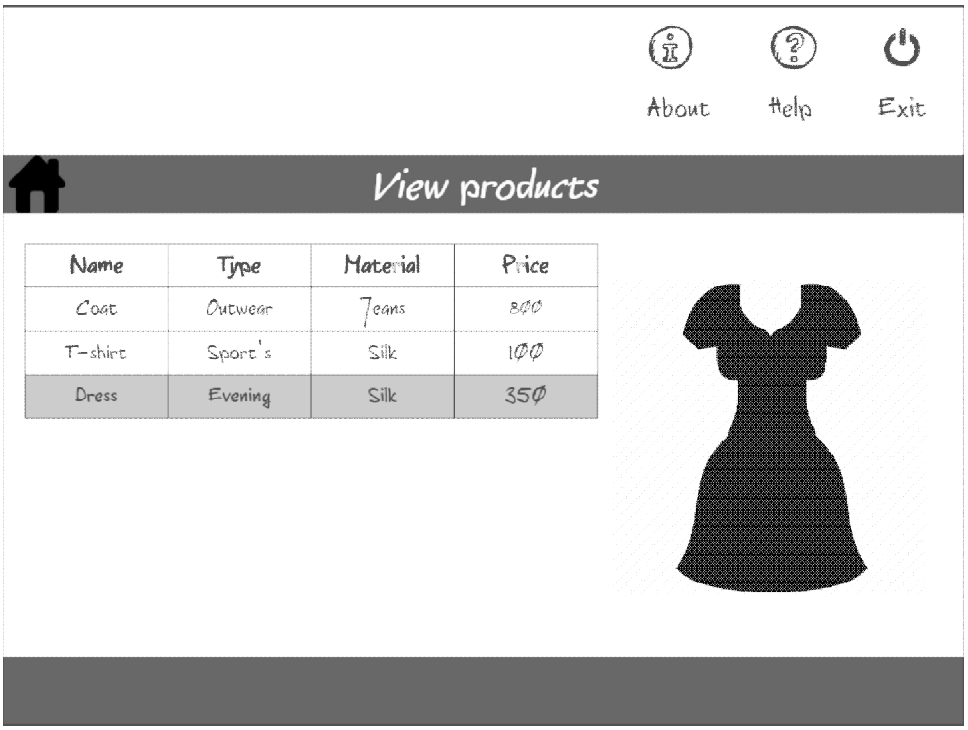

Рисунок 1.9 – прототип функції «Перегляд виробів»

Прототип для функції «Перегляд замовлень» на рисунку 1.10

Рисунок 1.10 - прототип для функції «Перегляд замовлень»

Прототип для функції «Додати виріб» на рисунку 1.11

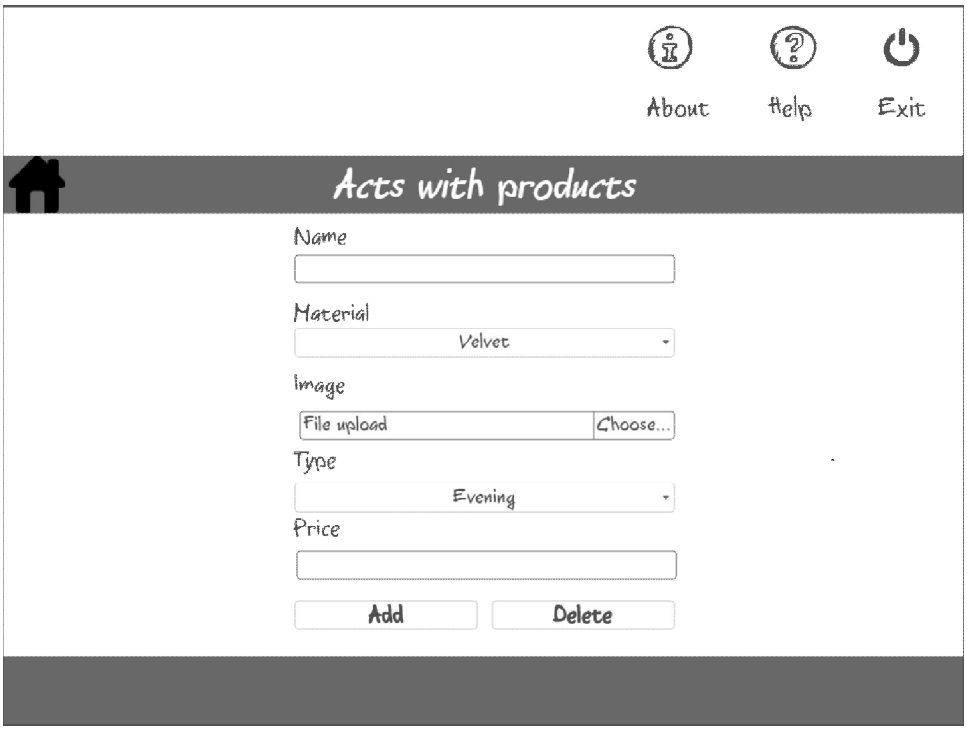

Рисунок 1.11 - прототип функції для «Додати виріб»

Прототип для функцій «Видалити виріб» на рисунку 1.12

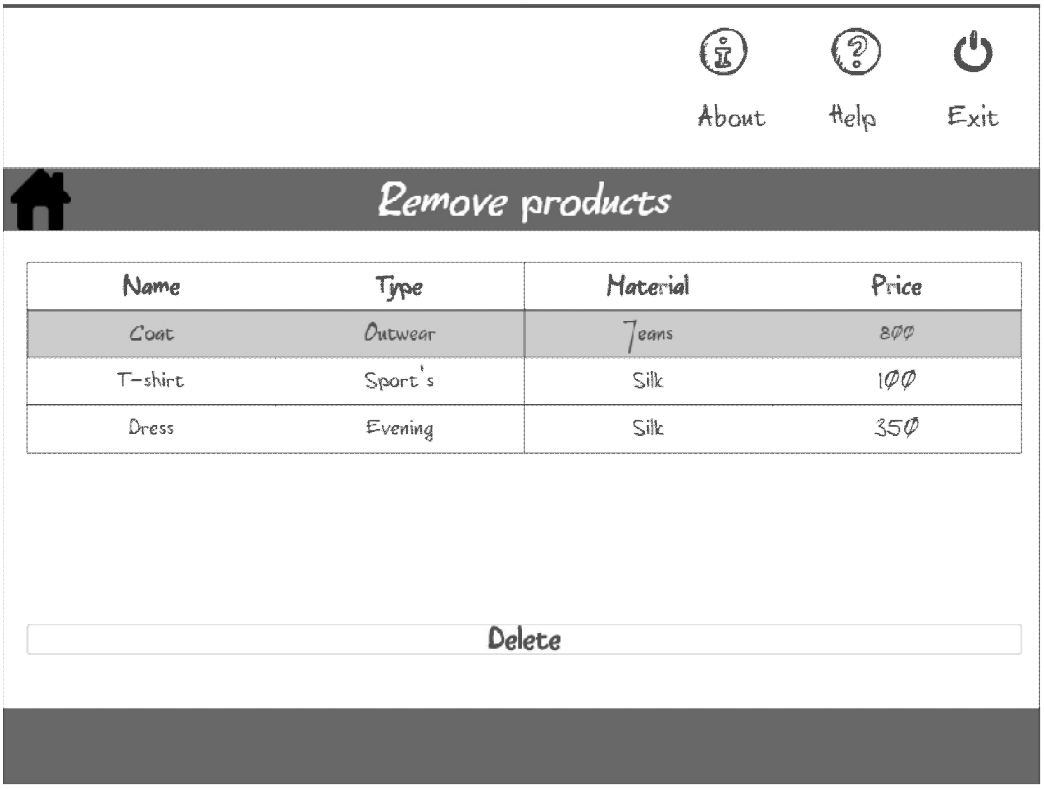

Рисунок 1.12 - прототип функцій «Видалити виріб»

Прототип для функції «Формування замовлення» на рисунку 1.13

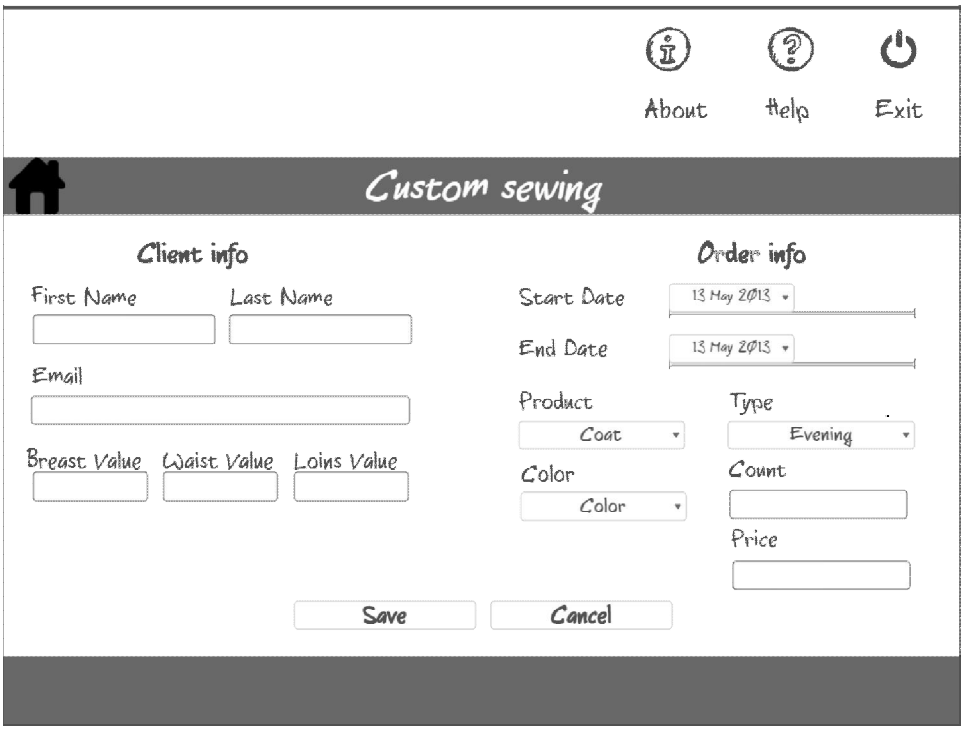

Рисунок 1.13 – прототип функції «Формування замовлення»

Прототип для функцій «Редагувати замовлення», «Видалити замовлення» на рисунку 1.14

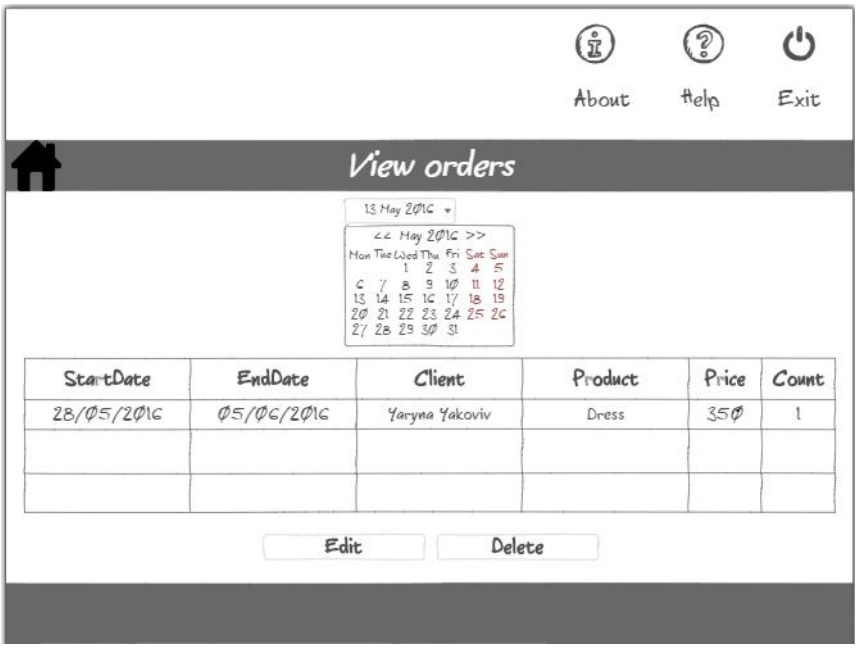

Рисунок 1.14 – прототип функцій для «Редагувати замовлення», «Видалити замовлення»

Специфікація функціональних вимог наведена у таблиці 1.13.

Таблиця 1.13

Специфікація функціональних вимог

Ідентифікатори вимоги	Назва вимоги	Атрибут вимог		
		Пріоритет	Складність	Контакт
1	Авторизація	Обов'язкове	Середня	Адміністратор Майстер
2	Перегляд виробів	Рекомендоване	Низька	Адміністратор Майстер
3	Перегляд замовлень	Обов’язкове	Висока	Адміністратор Майстер
4	Додавання виробу	Обов’язкове	Висока	Адміністратор
6	Видалення виробу	Рекомендоване	Низька	Адміністратор
7	Формування замовлення	Обов’язкове	Висока	Майстер
8	Редагування замовлення	Рекомендоване	Середня	Майстер
9	Видалення замовлення	Рекомендоване	Низька	Майстер

Специфікація функціональних вимог наведена у таблиці 1.14.

Таблиця 1.14

Специфікація нефункціональних вимог

Індетифікатор вимоги	Назва вимоги	Атрибути вимог		
		Пріоритет	Складність	Контакт
1. Застосовність				
1.1	Основні вимоги застосовності нової системи відносно інших систем, які знають користувачі	Рекомендована	Низька	Адміністратор, майстер

Продовження таблиці 1.14

1.2	Вимоги по відповідальності стандартам графічного інтерфейсу користувача	Обов’язкова	Низька	Адміністратор, майстер
1.3	Час, необхідний для навчання звичайних і досвідчених користувачів	Рекомендована	Середня	Адміністратор, майстер
2. Надійність				
2.1	Доступність	Обов’язкова	Середня	Адміністратор
2.2	Середній час безвідмовної роботи	Обов’язкова	Середня	Адміністратор
2.3	Точність	Обов’язкова	Середня	Адміністратор
3. Робочі характеристики				
3.1	Використання ресурсів	Рекомендована	Середня	Адміністратор
4. Проектні обмеження				
4.1	Вимоги до технології програмування	Рекомендована	Середня	Адміністратор

Значення нефункціональних вимог:

- час, необхідний для навчання звичайних користувачів – 30 хвилин;

- час, необхідний для навчання досвідчених користувачів – 15 хвилин;

- основні вимоги застосовності нової системи відносно інших систем, які знають користувачі – всі функції системи є легкими у виконанні, а структура програми не відрізняється від існуючих аналогів;

- вимоги по відповідності загальним стандартам застосовності та стандартам графічного інтерфейсу користувача – програма повинна працювати в операційній системі Windows XP і вище;

- доступність – час, що витрачається на обслуговування системи не повинен перевищувати 3% від загального часу роботи;

- середній час безвідмовної роботи – 3години;

- використання ресурсів – мінімальні системні вимоги:

  - Об’єм HDD: > 20 Мб
  - Об’єм RAM: > 512 Мб
  - Відео пам'ять: > 128 Мб
  - Процесор: > 1 ГГц
  - Клавіатура, маніпулятор миша

Висновки до першого розділу

У процесі розробки даного продукту було виконано наступні завдання:

1. Наведено коротку характеристику об’єкту управління, описано напрями його діяльності, розроблено схему організаційної структури.

2. Визначено склад функцій, що входять до бізнес-процесу на основі яких розроблено схему управління бізнес-процесом.

3. Проведено критичний аналіз існуючих аналогів, що реалізують аналогічні функції.

4. Розроблено специфікацію вимог до програмної системи для моделювання роботи скінченних автоматів.

5. Розроблено структурну схему програмної системи, побудовано UML діаграми, що реалізують логіку основних класів.

6. Наведено прототипи для розроблюваного продукту.

РОЗДІЛ 2
ПРОЕКТУВАННЯ

2.1 Розробка архітектури програмної системи

Для розробки системи було обрано трирівневу архітектуру. Трирівнева архітектура включає в себе такі компоненти як: клієнтський додаток , підключений до сервера додатків, який в свою чергу підключений до серверу бази даних .

Структурну схему програмної системи зображено на рисунку 2.1.

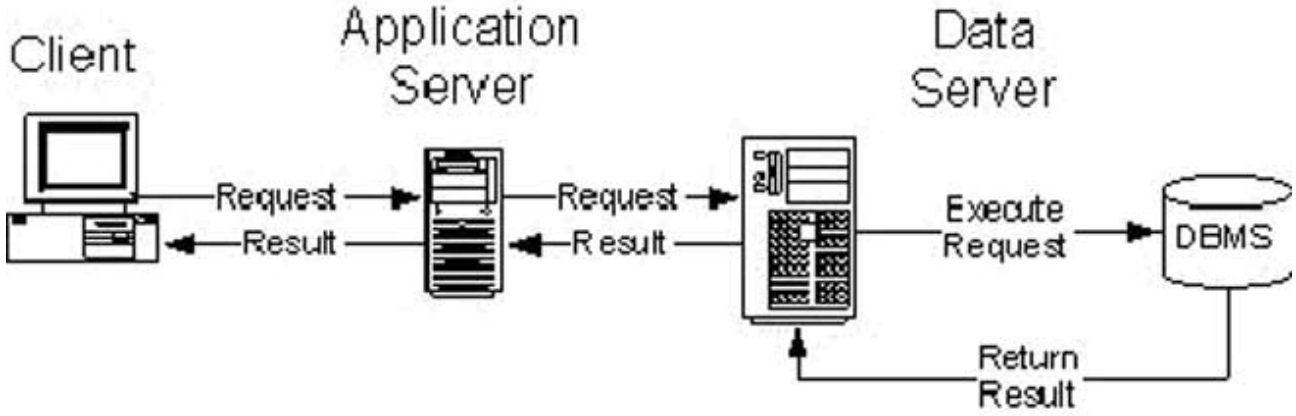

Рисунок 2.1 – Структурна схема програмної системи

Рівень клієнта - це власне розроблена програмна система для кінцевого користувача, тому він не повинен мати прямих зв'язків з базою даних і не повинен бути навантаженим основною бізнес-логікою. На першому рівні даної системи винесено найпростіша бізнес-логіка, а саме: авторизація користувача в систему, перегляд продуктів та замовлень.

Другий рівень – рівень сервера. На ньому зосереджена більша частина бізнес-логіки. Тут відображається програмний інтерфейс, що зв'язує клієнтські компоненти з прикладною логікою бази даних. У даному продукті це реалізації функцій додавання нового продукту та нового замовлення.

Сервер бази даних, який обраний MS SQL Server 2012, забезпечує зберігання даних і виноситься на третій рівень. Цей рівень являє собою базу даних, разом з збереженими процедурами, тригерами і схемою, яка описує продукт в термінах реляційної моделі.

Функціональну структуру системи умовно можна розділити на такі модулі:

- модуль обробки інформації про користувачів:
    1) авторизація.
- модуль обробки інформації про вироби:
    1) додавання виробу;
    2) видалення виробу;
    3) перегляд виробів.
- модуль обробки інформації про замовлення:
    1) формування замовлення;
    2) редагування замовлення;
    3) видалення замовлення;
    4) перегляд замовлень.

Для того, щоб краще зрозуміти процес функціонування системи побудовано діаграми станів для кожного модуля.

Діаграма активності авторизації користувача зображено на рисунку 2.2.

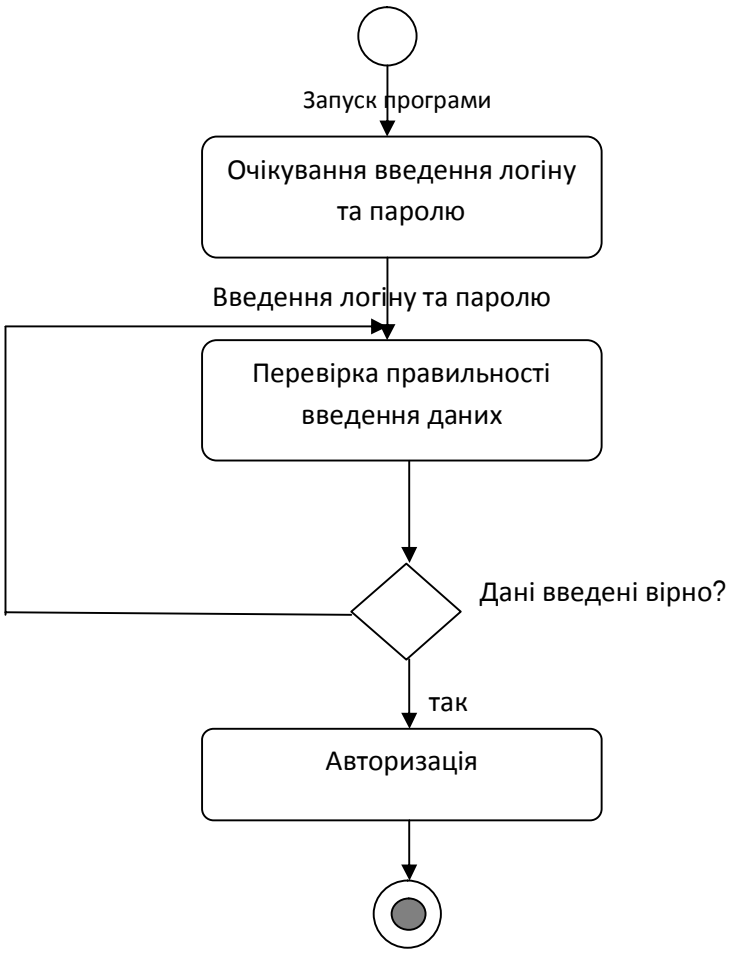

Рисунок 2.2 – Діаграма активності процесу авторизації користувача

Діаграму активності для модуля обробки інформації про вироби зображено на рисунку 2.3

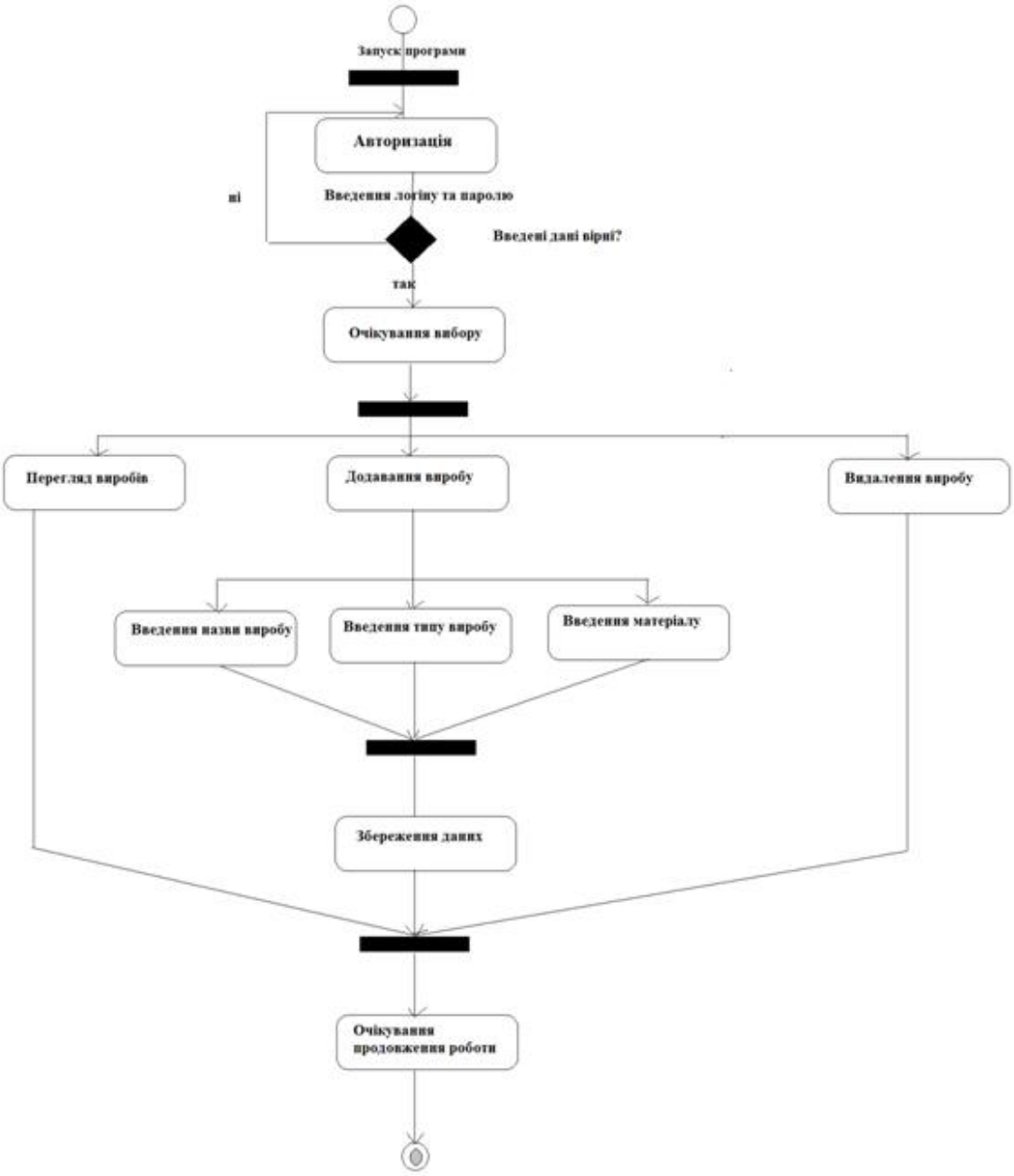

Рисунок 2.3 – Діаграма активності модуля обробки інформації про вироби

Діаграму активності для модуля обробки інформації про замовлення зображено на рисунку 2.4.

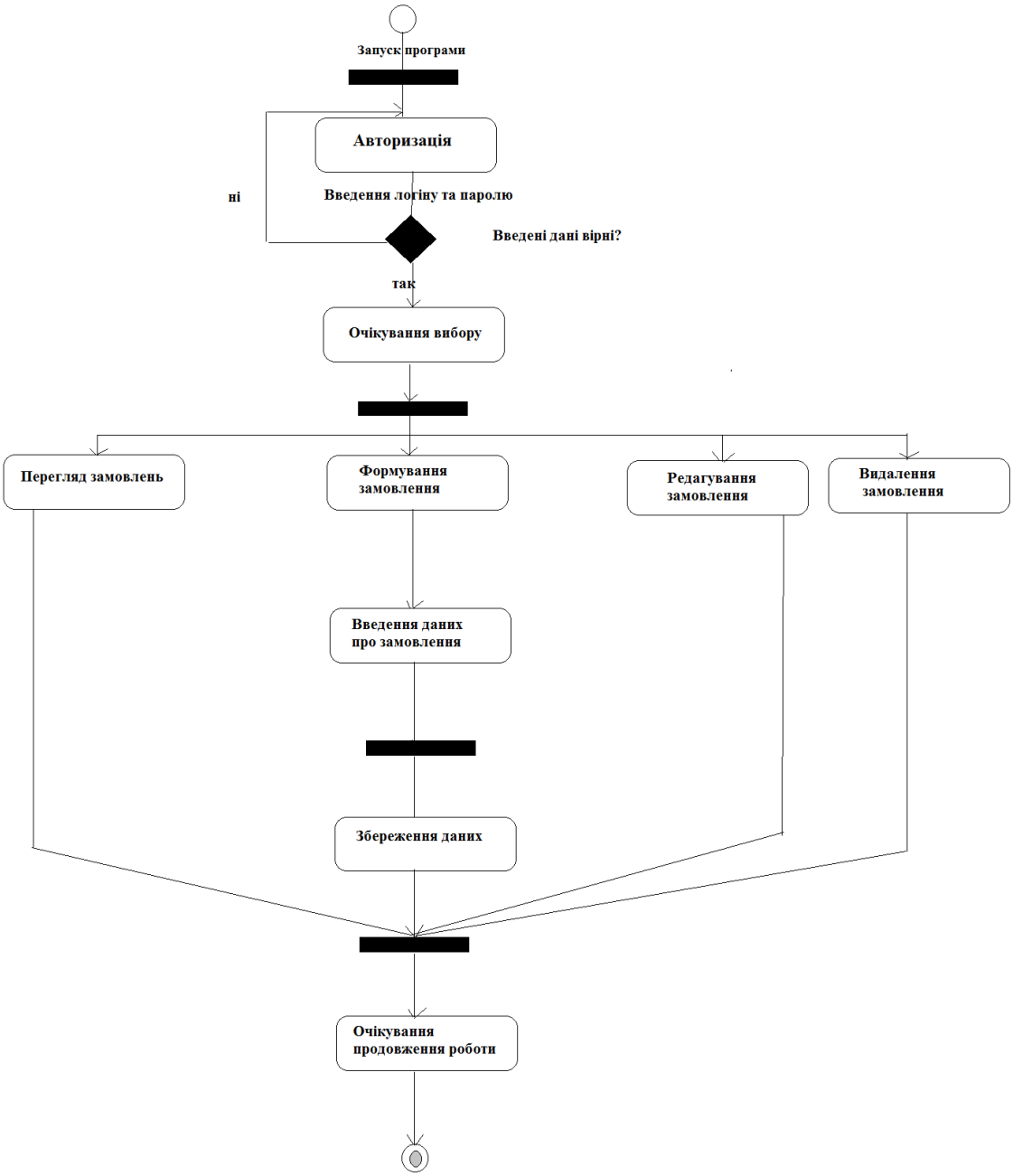

Рисунок 2.4 – Діаграма активності для модуля обробки інформації про замовлення

2.2 Проектування структури бази даних

Етап проектування бази даних вважається одним із найскладніших етапів створення БД, який не має явно вираженого початку та закінчення.

Проектування бази даних починається з моменту прийняття стратегічних рішень і триває на етапах реалізації та тестування.

Процес проектування БД охоплює кілька основних сфер:

- проектування об'єктів БД (таблиці, подання, індекси, тригери, збережені процедури, функції, пакети) для подання даних ПО в БД;
- проектування інтерфейсу взаємодії з БД (форми, звіти й т.д.), тобто проектування додатків, які будуть супроводжувати дані в БД і реалізовувати питально-відповідні відношення на цих даних;
- проектування БД під конкретне обчислювальне середовище або інформаційну технологію;
- проектування БД під призначення.

Даний проект розробляється для користувача. І стосується розробки настільного додатку для оформлення замовлення ательє з пошиття одягу. Для досягнення цієї мети потрібно забезпечити обмін інформацією між користувачем та додатком.

Для формалізації і опису бізнес-процесів в системі було використано мову моделювання IDEF0. В цій методології система представляється, як сукупність взаємодіючих робіт і функцій. Тут об'єкти представлені ієрархічно, що значно полегшує розуміння предметної області. В IDEF0 розглядаються логічні зв'язки між роботами, а не послідовність їх виконання в часі.

В результаті застосування методології отримана модель, яка побудована з чітко сформульованою метою і з єдиною точкою зору, яка складається з діаграм, фрагментів текстів, що мають посилання один на одного. На рисунку 2.5 зображена контекстна діаграма, а на рисунку 2.6 зображена діаграма декомпозиції. В додатку Б наведено детальніше модель IDEF0.

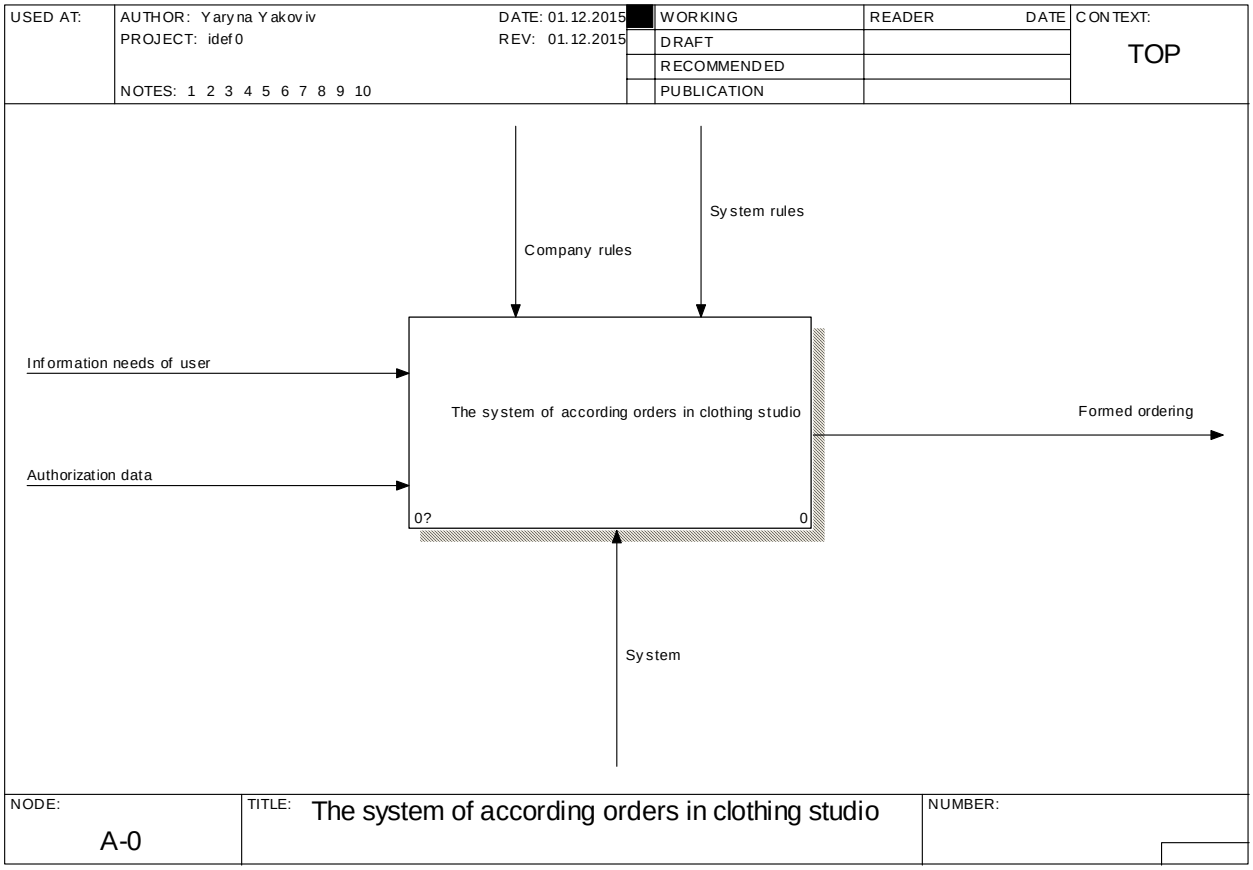

Рисунок 2.5 - Контексна діаграма

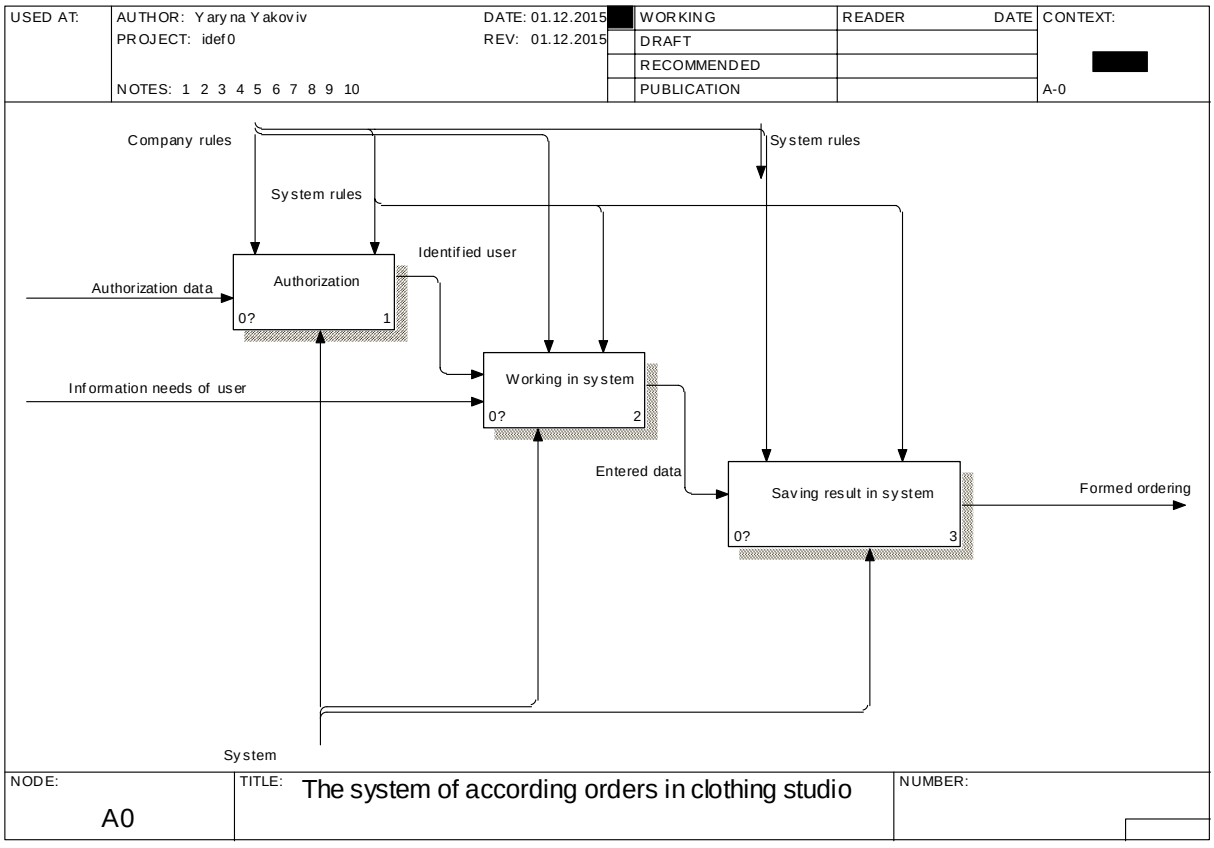

Рисунок 2.6 - Діаграма декомпозиції

Інформацію про замовлення та клієнта користувач вводить самостійно. Взаємодію інформації між користувачем та додатком зручно подати у вигляді моделі потоків даних, що зображена на рисунках 2.7-2.8.

Для опису документообігу і обробки інформації в системі побудуємо діаграму потоків даних. Діаграма являє собою графічне представлення потоків даних і обробку даних в інформаційній системі. На діаграмі показано джерела і споживачі інформації, а також процеси, які здійснюють обробку інформації.

В додатку В наведена діаграма потоків даних, яка відображає характер інформаційних процесів та потоків даних в системі.

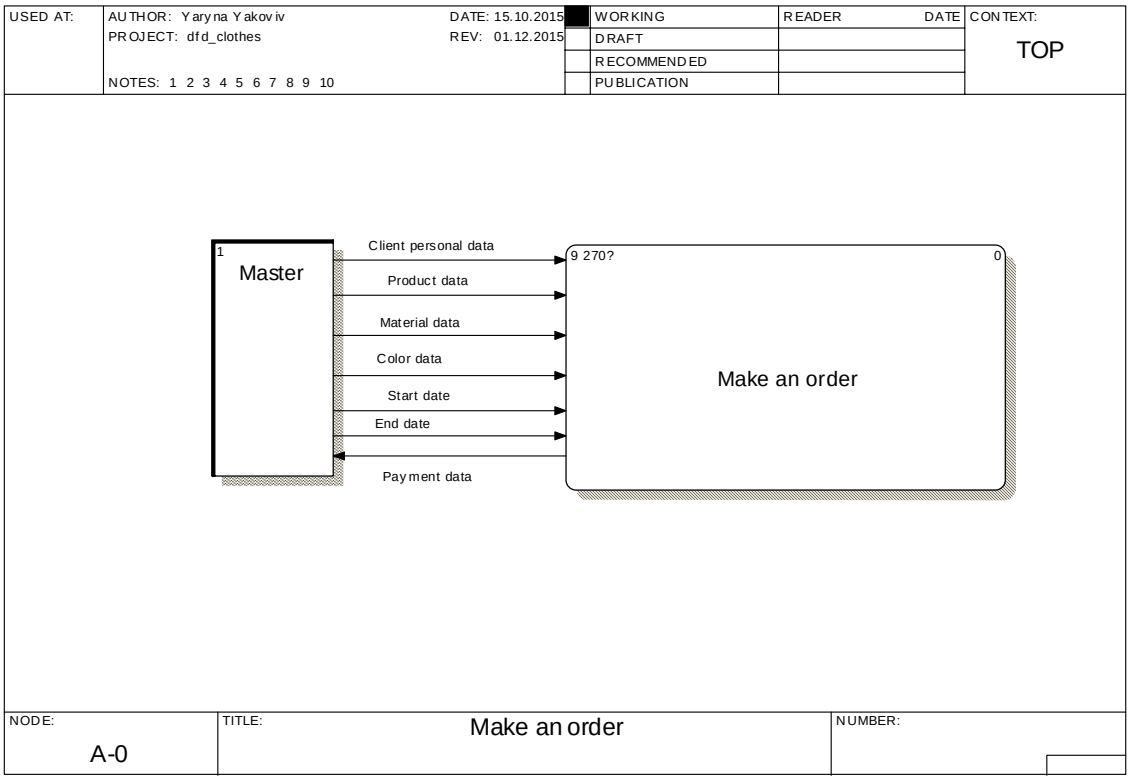

Рисунок 2.7 – Діаграма потоків даних

Діаграма декомпозиції зображена на рисунку 2.6.

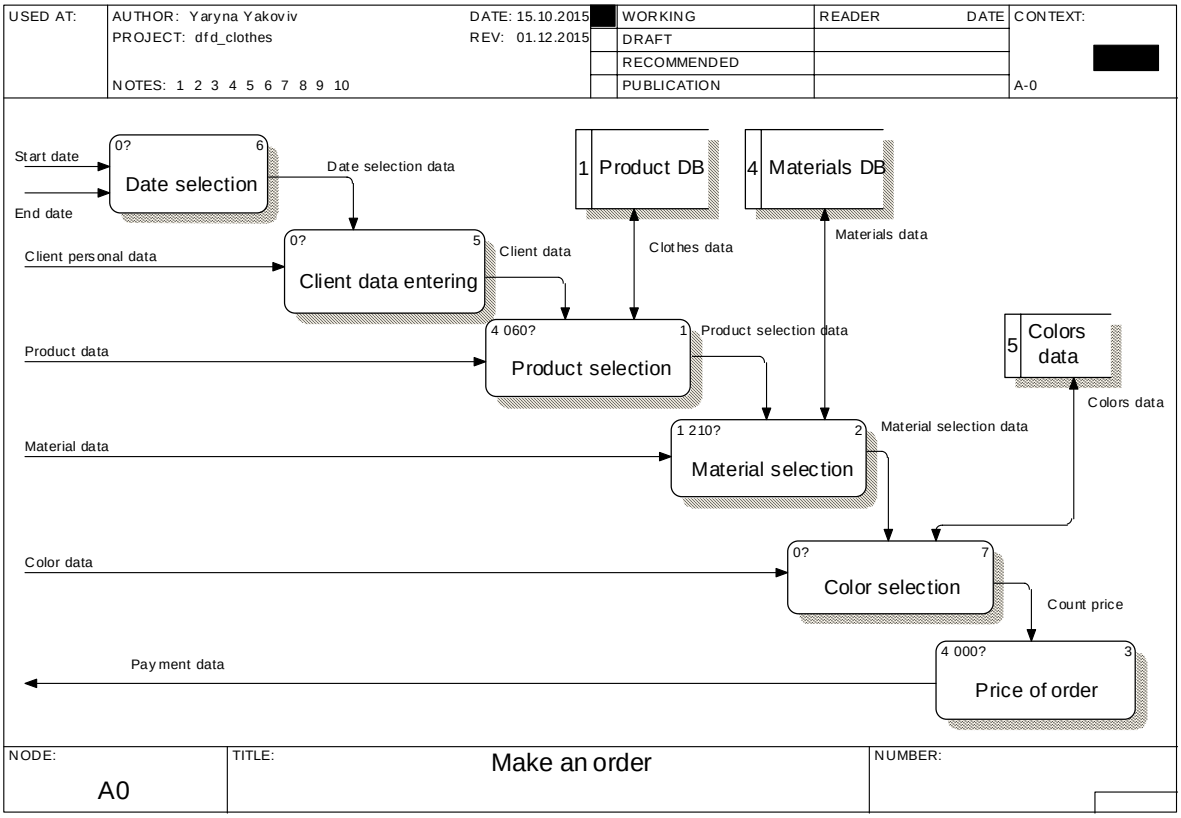

Рисунок 2.8 – Діаграма декомпозиції

Як показано на рисунку 2.8 робота програмної системи здійснюється наступним чином :

1. Входи (input), потоки, які поступають у систему і переробляються нею у вихідні величини, на діаграмі зображені ліворуч:

- ✓ Criteria (критерії пошуку);
- ✓ Information about task (інформація про завдання);
- ✓ List categories (список категорій).

2. Виходи (output), продукти діяльності системи, тобто результати перетворення вхідних величин тощо, на діаграмі зображені праворуч:

- ✓ Result (результат);
- ✓ Completed task (виконане завдання).

Процес формування замовлення відбувається за допомогою майстра, який взаємодіє з програмною системою. Майстер вибирає дату початку виконання замовлення та дату завершення виконання замовлення, щоб визначити межі виконання. Після цього він вводить дані про клієнта – прізвище, ім’я, email.

Наступним кроком є внесення даних про виріб, а також матеріал, з якого він буде зроблений та колір. Після цього ми визначаємо ціну сформованого замовлення.

На завершення формуємо остаточне замовлення і відправляємо його на виконання. Після цього усі дані записуються в базу даних.

Діаграма IDEF3 будується для моделювання процесів, що відбуваються в системі. IDEF3 – метод документування технологічних процесів, що надає механізм документування та збору інформації про процеси. IDEF3 показує причинно-наслідкові зв'язки між ситуаціями і подіями в зрозумілій експерту формі, використовуючи структурний метод вираження знань про те, як функціонує система. Створена модель наведена в додатку В.

Наступним кроком є виявлення основних сутностей та зв’язків між ними. Для цього створено діаграму елементів та зв’язків (див. рис. 2.7).

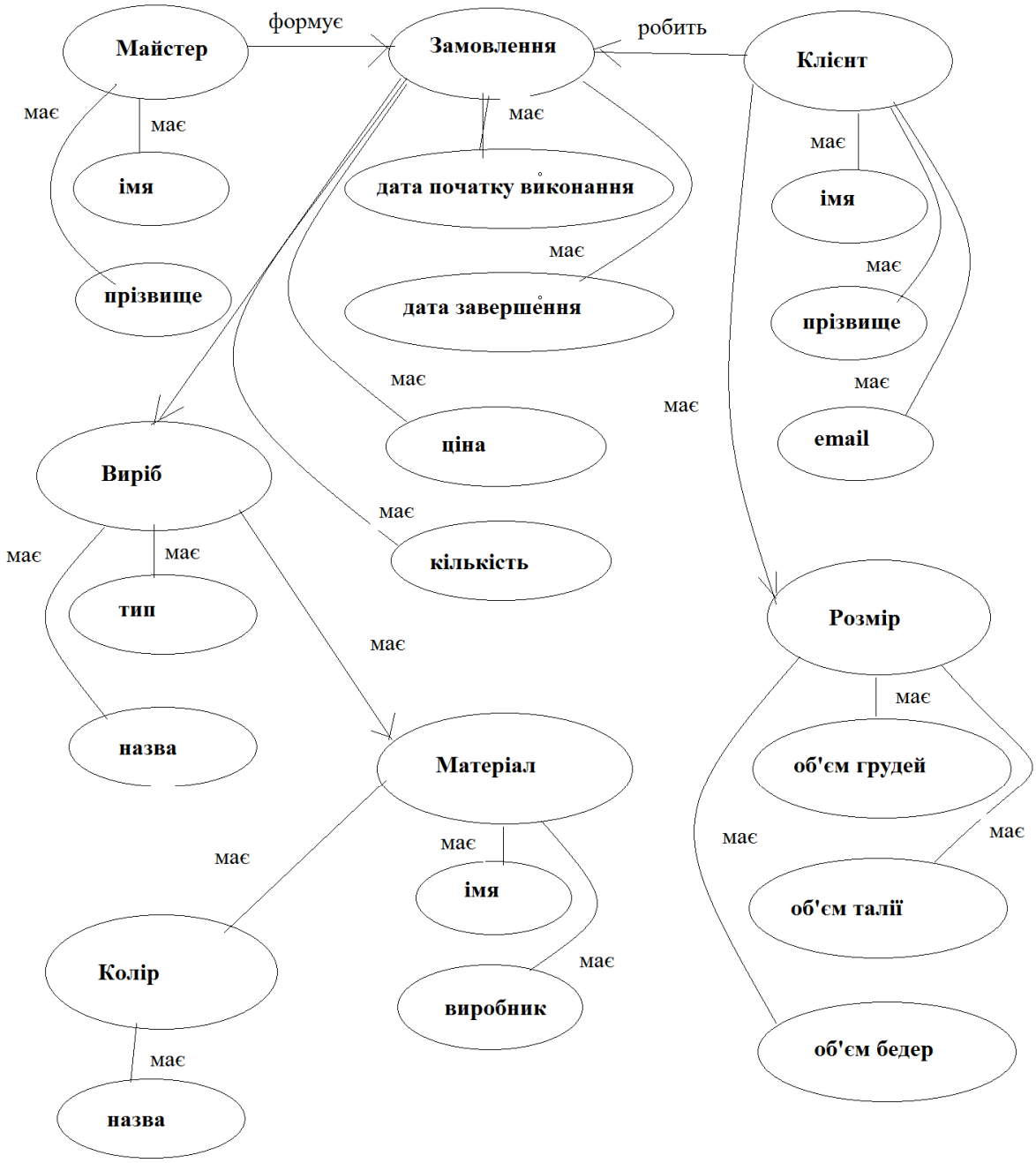

Рисунок 2.9 – Діаграма елементів та зв’язків

Після цього було створено таблицю ідентифікаторів, яку подано у таблиці 2.1.

Таблиця 2.1

Таблиця ідентифікаторів

Об’єкт	Властивість	Тип	Розмірність	Ідентифікатор
Замовлення				Ordering
	Дата початку виконання	Date		DateStart
	Дата завершення	Date		DateEnd
	Ціна	String	10	Cost
	Кількість	Integer	5	Count
Виріб				Product
	Назва	String	100	Name
	Тип	String	50	Type
Матеріал				Material
	Назва	String	100	Name
	Виробник	String	100	Manufacturer
Клієнт				Client
	Ім’я	String	100	FirstName
	Прізвище	String	100	LastName
	E-mail	String	50	Email
Розмір				Size
	Об’єм талії	String	10	WaistVolume
	Об’єм грудей	String	10	BreastVolume
	Об’єм бедер	String	10	HipVolume
Колір	Назва	String	100	Color

Наступним кроком є створення діаграми класів UML (див. рис. 2.8).

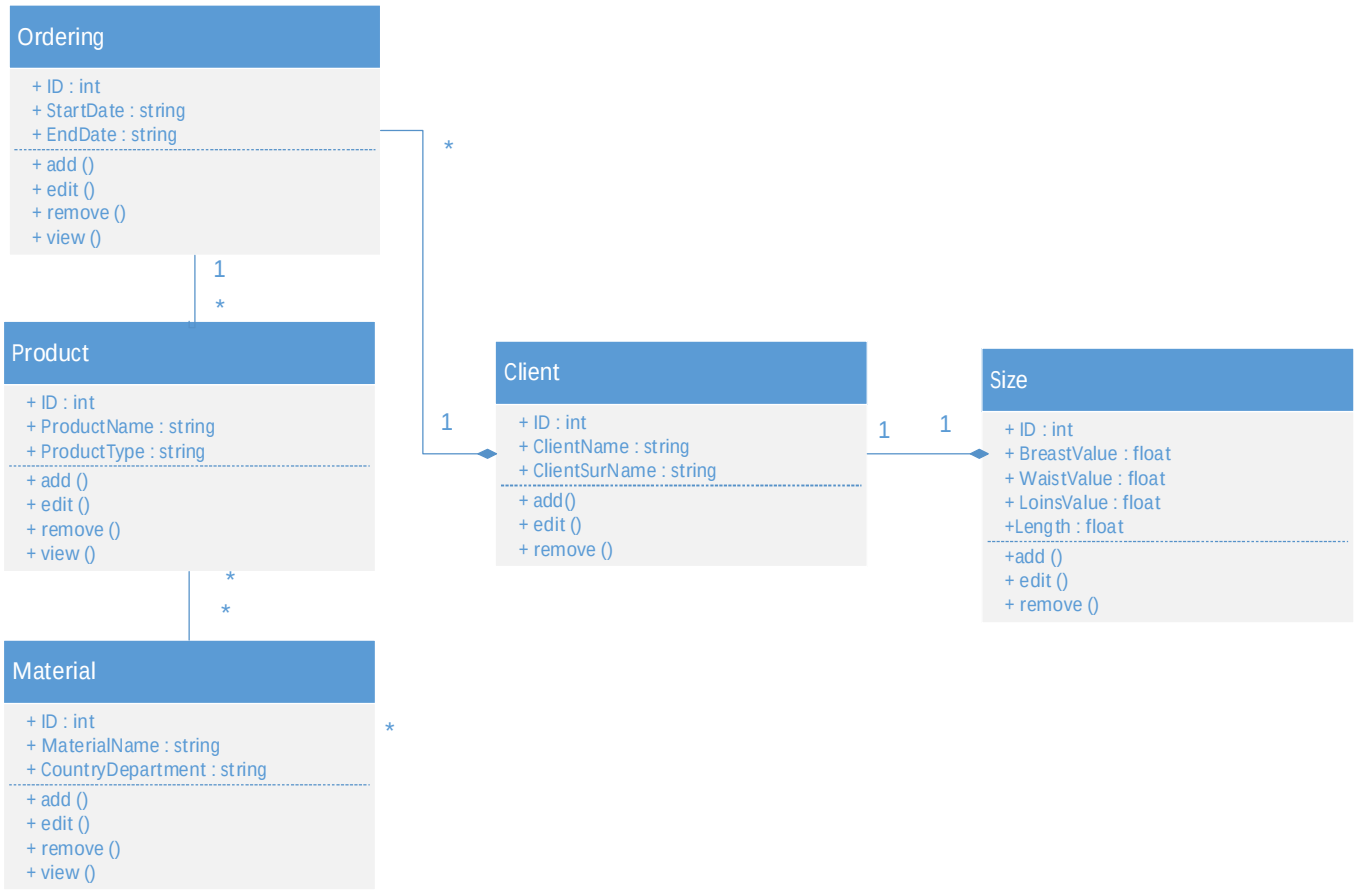

Рисунок 2.10 – Діаграма класів UML

Найбільше поширеним засобом моделювання даних є діаґрами “сутність-взаємозв'язок” (ERD). З їхньою допомогою визначаються важливі для предметної області об'єкти (сутності), їх властивості (атрибути) і відношення один з одним (зв'язки). ERD безпосередньо використовуються для проектування реляційних баз даних.

Тому на підставі аналізу предметної області у роботі спроектовано структуру бази даних у вигляді ERD діаграми (рисунок 2.11).

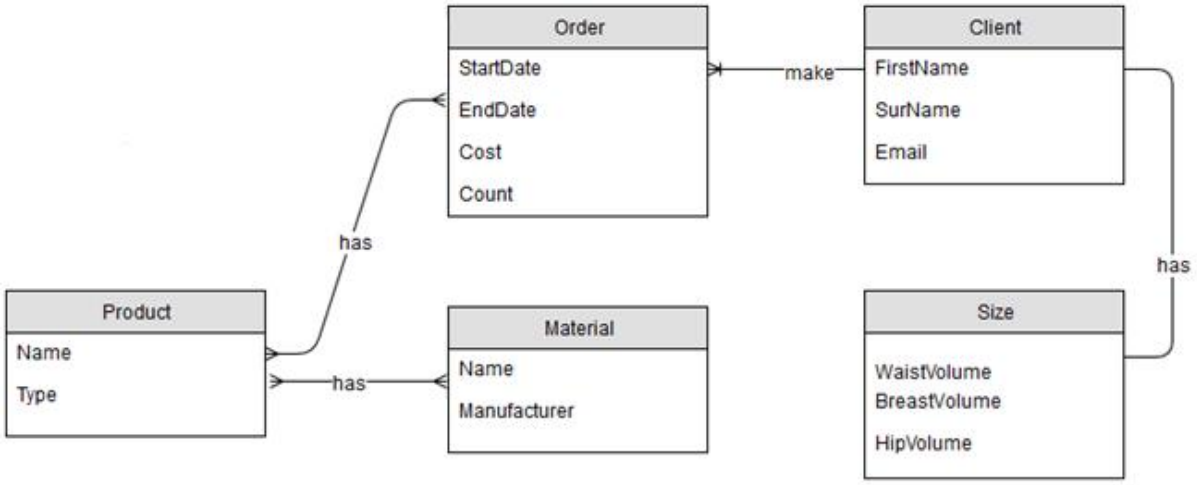

Рисунок 2.11 – ERD-діаграма

Проектування структури бази даних дозволяє визначити та повністю описати всі відношення та поля і є завершальною ланкою перед етапом програмної реалізації бази даних.

Висновки до другого розділу

В ході проектування програмного продукту було обрано архітектуру для її розробки, її актуальність. Було проведено проектування бази даних програмної системи в рамках якого спроектовано її фізичну модель.

Було створено наступні діаграми для реалізації проекту:

- Діаграма активності модуля системи;
- Діаграма декомпозиції та контекстна діаграма;
- Діаграма елементів та зв’язків;
- Діаграма класів;
- ERD – діаграма.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Програмна реалізація проекту

При створенні програмного продукту важливо, яким чином він буде реалізований, на якій платформі, на якій мові програмування, адже від цього залежить надійність, продуктивність і якість роботи розроблюваної програмної системи.

Оскільки система є платформною, то в процесі розробки буде використовуватись мова C# — об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET.

Призначено C# програмування для використання з програмною платформою .NET Framework. Платформа .NET Framework грає роль своєрідної операційної системи всередині операційної системи, яка дозволить створити нашу програму.

Організація інтерфейсу з користувачем відбувається за допомогою Windows Forms Application, створеної у середовищі Visual Studio 2013.

Visual Studio 2013 включає .NET Framework 4.5. У Visual Studio 2013 упроваджено наступне покоління інструментів ASP.NET, є підтримка динамічних розширень в мовах програмування C# і Visual Basic, використовуються нові шаблони проектів, інструментарій для документування тестових сценаріїв і велика кількість нових бібліотек. Саме ця версія підтримує платформи десктопної розробки Windows 8.1, на якій розроблятиметься програмний продукт.

Це комплексне інтегроване середовище з широкими функціональними можливостями має удосконалений інтерфейс і містить нові інструменти для підтримки багатьох процесів. Тому це дозволить створювати інноваційний і якісний програмний продукт з привабливим інтерфейсом.

Першим етапом створення програмного продукту є створення Windows Forms Application. Це процес формування основної базової частини

програмного продукту. Він включає в собі створення основних форм та елементарних функцій.

На рисунку 3.1 зображено основні класи та модулі системи.

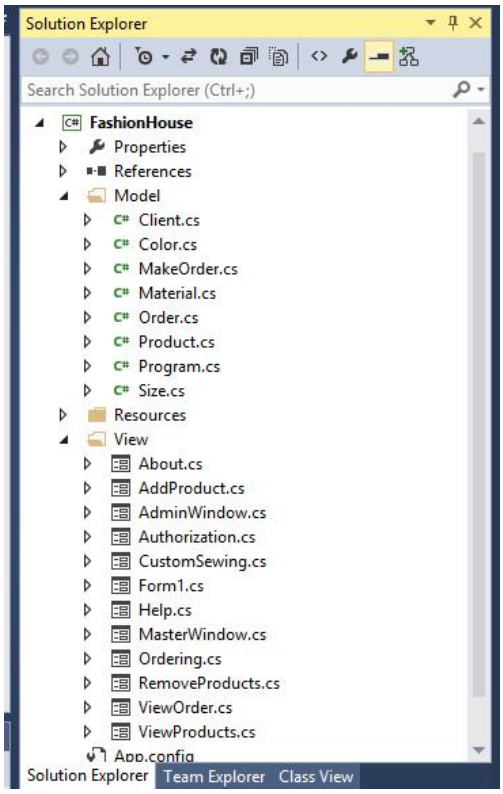

Рисунок 3.1 - Основні класи та модулі системи

Даний програмний продукт міститиме такі програмні модулі:

- Form1 – головна форма програми, через яку здійснюється вхід в систему;
- Authorization – модуль, через який користувачі можуть увійти в систему, ввівши відповідний логін та пароль;
- AdminWindow – головне вікно адміністратора і його основні функції;
- MasterWindow - головне вікно майстра і його основні функції;
- ViewProducts – модуль здійснення перегляду виробів, що додані до даної системи;
- AddProduct – модуль здійснення додавання та видалення виробів, дозволений адміністратором;

- ViewOrder – модуль перегляду усіх замовлень, сортування замовлень по бажаній даті та редагування, видалення замовлень по бажанню;
- Ordering – модуль вибору типу замовлення, яке буде здійснюватись;
- CustomSewing – модуль формування замовлення індивідуального та оптового пошиття;
- Trends – модуль формування замовлення згідно новинок сезону;
- Repairing – модуль ремонту одягу;
- RemoveProducts – модуль видалення продуктів;
- About – модуль перегляду інформації про створений продукт;
- Help – модуль перегляду інформації для експлуатації даного продукту.

У додатку Г наведені усі класи реалізованого програмного продукту.

3.2. Програмна реалізація бази даних

Для збереження даних в програмну систему використовується система управління реляційними базами даних MS SQL SERVER 2012.

SQL Server 2012 включає низку вдосконалень для роботи з критичними бізнес-застосунками і бізнес-аналітикою.

Microsoft SQL Server включає також Common Language Runtime (CLR) Microsoft .NET, що дозволяє застосункам, розробленим на мовах платформи .NET (також обрана мова C#), реалізовувати процедури, що зберігаються та різні функції.

Середовищем для написання коду було обрано Visual Studio 2013 - інтегроване рішення для управління життєвим циклом додатків. Завдяки Visual Studio 2013 підвищується ефективність роботи під час розробки рішень для бізнесу і споживчих переваг.

Оскільки програмний продукт буде використовувати СУБД MS SQL Server 2012, то потрібно встановити цю базу даний на свій робочий комп’ютер та підключити її до нашого проекту.

Нижче наведено код підключення проекту до бази даних:

```
public class DataBaseConnector
    {
        private static String connectionString
        {
            get
            {
                return "Data Source = HP; database=FashionHouse; Integrated
Security=SSPI; Trusted_Connection=Yes;";
            }
        }
     }
```

Щоб створити таблицю, скористаємося виразом CREATE TABLE, в якому задамо ім'я нашої нової таблиці. Вираз починається з CREATE TABLE, після якої слідує ім'я таблиці. Потім в дужках вказується список стовпців, і інформація про ключі. Кожному стовпцю дається ім'я, вказується тип даних, і значення за замовчуванням, якщо воно доречно. Нижче наведено код створення таблиці «Client»:

```
USE FashionHouse;
CREATE TABLE Client (
Id UNIQUEIDENTIFIER PRIMARY KEY,
FirstName NVARCHAR(50),
LastName NVARCHAR(50),
Email NVARCHAR(320),
Size_Id uniqueidentifier NOT NUll)
```

Стовпець Id - це первинний ключ (primary key), колонка, яка унікально ідентифікує кожного клієнта. Інші стовпці використовують в якості типів даних рядки змінної довжини (NVARCHAR). На рисунку 3.2 проілюстровано структуру таблиці «Client».

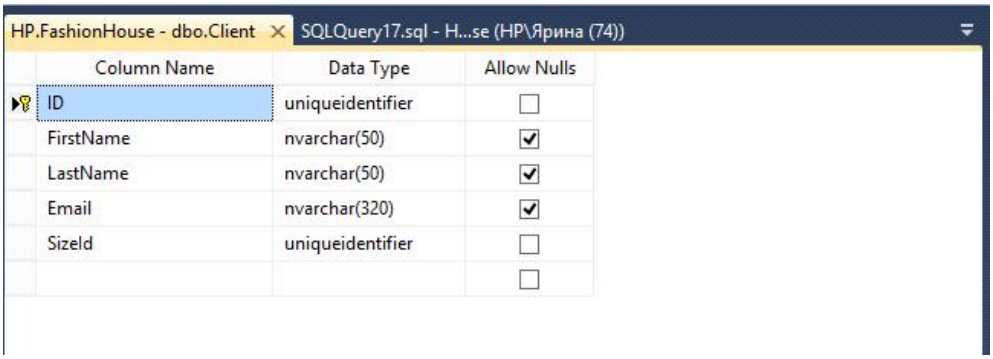

Рисунок 3.2. Структура таблиці «Client»

Далі додаємо в таблицю запис – клієнта з прізвищем, ім’ям та емейлом.

Нижче наведено SQL код даного запиту:

```
INSERT INTO Client (FirstName, LastName, Email)
VALUES ('Viktoria','Nesmiyanova','vika@gmail.com');
```

Після частини INSERT INTO, пишеться ім'я таблиці. Потім відкриваємо першу дужку, перераховуємо стовпці в які буде здійснена вставка, розділяючи їх один від одного комами. Після перерахування списку стовпців дужка закривається і вказується зарезервоване слово VALUES, після якого в дужках перераховуються значення які потрібно вставити в таблицю, причому перераховуються в такому самому порядку, що і назви стовпців. Якщо значення мають символьний тип даних, необхідно укладати їх в лапки. Результат даного запиту зображено на рисунку 3.3.

	c393666c-b405-...	Yaryna	Yakoviv	yy@y.y	89223d91-0595-...
▶	7d3a4d7d-ccd7...	Viktoria	Nesmiyanova	vika@gmail.com	89223d91-0595-...
*	NULL	NULL	NULL	NULL	NULL

Рисунок 3.3. Таблиця «Client» із заповненими полями

Аналогічно створюємо таблиці «Material», «Ordering», «Product», «Size», та записуємо в них відповідні дані.

```
USE FashionHouse;
CREATE TABLE Material (
Id UNIQUEIDENTIFIER PRIMARY KEY,
MaterialName NVARCHAR(50),
CountryDepartment NVARCHAR(50),
Price FLOAT)

USE FashionHouse;
CREATE TABLE Ordering (
Id UNIQUEIDENTIFIER PRIMARY KEY,
StartDate DATE,
EndDate DATE,
Color NVARCHAR(50),
Price FLOAT,
Kount INT,
Client_Id uniqueidentifier NOT NUll
Product_Id uniqueidentifier NOT NUll
Material_Id uniqueidentifier NOT NUll)

USE FashionHouse;
CREATE TABLE Product (
Id UNIQUEIDENTIFIER PRIMARY KEY,
ProductName NVARCHAR(50),
ProductType NVARCHAR(50),
Image NVARCHAR(50),
Price FLOAT,
Material_Id uniqueidentifier NOT NUll)
```

```
USE FashionHouse;
CREATE TABLE Size (
Id UNIQUEIDENTIFIER PRIMARY KEY,
BreastValue FLOAT,
WaistValue FLOAT,
LoinsValue FLOAT)
```

Якщо виникає необхідність видалити певний запис з таблиці можна скористатись оператором DELETE. Синтаксис тут дуже простий, і включає тільки назву таблиці, і умову при якому буде виконана операція видалення. SQL код даного запису наведений нижче:

```
DELETE FROM Client
WHERE FirstName='i';
```

Нижче на рисунку 3.4 зображено створені таблиці в базі даних.

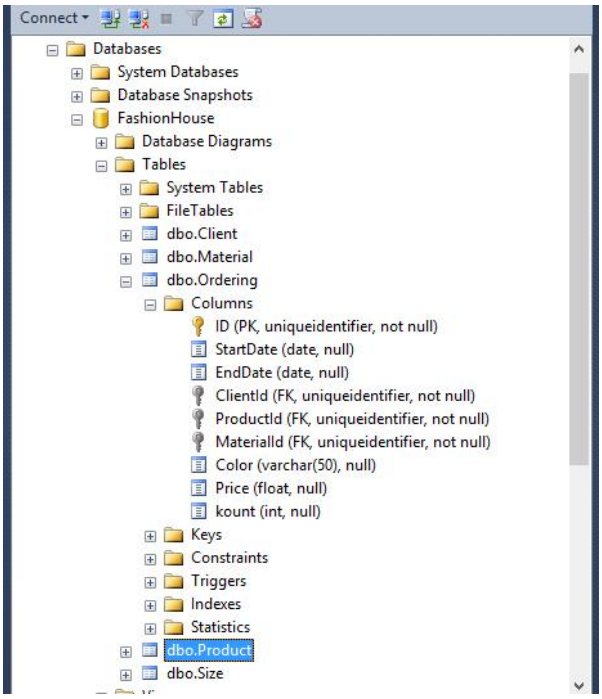

Рисунок 3.4 Створені таблиці в БД

Щоб реалізувати функцію для адміністратора, а саме «Додавання продукту» - потрібно отримати дані з БД, та створити функцію додавання даних до БД.

Щоб отримати дані з БД створимо StoredProcedure – GetMaterials

```
USE [FashionHouse]
GO
/****** Object:  StoredProcedure [dbo].[GetMaterials]    Script Date: 28.04.2016 19:11:25
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

```
-- Batch submitted through debugger:
SQLQuery4.sql|7|0|C:\Users\1D4F~1\AppData\Local\Temp\~vs15B5.sql

CREATE PROCEDURE [dbo].[GetMaterials]
AS
BEGIN
SELECT *
FROM Material
END
```

Результат виконання даної процедури можна подивитись на рисунку 3.5.

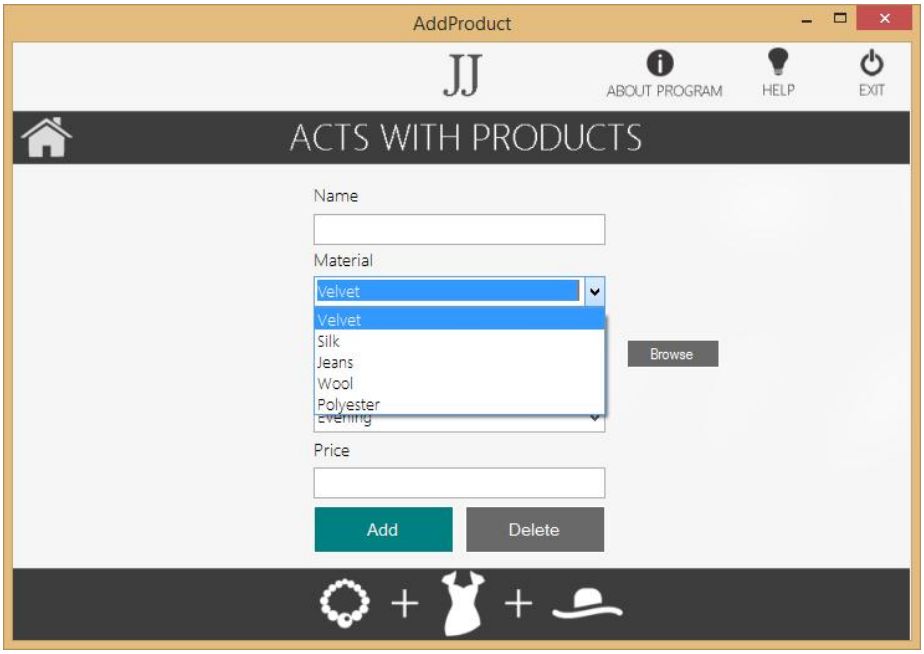

Рисунок 3.5. Результат виконання StoredProcedure – GetMaterials

Аналогічно для роботи з програмним продуктом потрібно створити наступні StoredProcedure. Усі StoredProcedure наведені в додатку Г.

Висновки до третього розділу

В ході програмної реалізації було реалізовано програмний продукт повністю з усіма необхідними функціями. Було створено базу даних з відповідними наборами даних. В ході даного розділу було виявлено основні модулі системи, їхня реалізація та взаємодія один з одним, а також з обраною базою даних MS SQL Server 2012.

РОЗДІЛ 4
ТЕСТУВАННЯ ТА ДОСЛІДНА ЕКСПЛУАТАЦІЯ

4.1. Тестування

Тестування програмного забезпечення — це процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись. Техніка тестування також включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою оцінки.

Оскільки число можливих тестів навіть для нескладних програмних компонент практично нескінченне, тому стратегія тестування полягає в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів. Як результат програмне забезпечення тестується стандартним виконанням програми з метою виявлення дефектів.

При розробці програмного продукту було проведено модульне та системне тестування. Було проведено функціональне тестування та тестування графічного інтерфейсу. Під час розробки системи та її тестування було налаштовано та наповнено БД тестовими даними.

Функціональне тестування розглядає заздалегідь вказане поведінку і ґрунтується на аналізі специфікацій функціональності компонента або системи в цілому. Функціональні види тестування розглядають зовнішню поведінку системи.

Оскільки перевагою цього тестування є те, що воно імітує фактичне використання системи, тому воно було проведене в першу чергу. На основі даного тестування були розроблені тестові випадки.

Під час фази розробки тестів було спроектовано 24 функціональних тестових випадків. Таблиця показує розподіл функціональних тестових випадків і наборів тестових даних для цих випадків за варіантами використання. Таблиця тест-кейсів прикріплена у додатку Д «Специфікація тестів для функціонального тестування».

Таблиця 4.1

Розподіл функціональних тестових

Варіанти використання	Тестові випадки	Тестові дані
Check logging	3	3
Check option “Exit”	2	2
Check option “About Program”	1	1
Check option “Help”	1	1
Check option “Change User”	2	2
Check option “View Products”	1	1
Check option “Products Acts”	1	1
Check option “View Orders”	1	1
Check option “Ordering”	1	1
Check option “Add Product”	3	3
Check option “Delete Product”	1	1
Check option “Custom Sewing”	4	4
Check option “Select Date”	2	2
Check option “Select Product”	1	1
Загалом	24	24

Результати тестування

Підсумок тестування: 23 тести з 24 функціональних тестів наведених в Додатку Д «Специфікація тестів для функціонального тестування» пройшли, отже функціональне тестування розглядається як частково успішне – 97% тестових випадків пройшли, 99.94% наборів тестових даних пройшли.

Відомі дефекти

Дефект #1 Коли користувач завантажує зображення невідомого типу, зображення не відображаються у вікні «View Orders».

Опис: Відповідно до тестового випадку Add New Product система повинна інформувати користувача про можливі типи. Дефект полягає в тому, що система не інформує користувача про те, що користувач не може завантажувати інші типи, крім типів зображень.

Подолання дефекту: Цей дефект неможливо виправити без шкоди для програмної системи. Проблема полягає в тому, що доступ є обмеженим.

Тестування графічного інтерфейсу десктопного додатку проводилось вручну на ПК HP ProBook 4535s із версією Windows 8.1. Під час тестування було перевірено коректність розміщення усіх елементів інтерфейсу та їх відповідність заданим стилям й призначеному функціоналу.

В ході даного тестування було перевірено:

- продуктивність, ефективність - скільки часу і кроків знадобиться користувачеві для завершення основних завдань програми, такі як оформлення замовлення, додавання продукту;
- правильність - скільки помилок зробив користувач під час роботи з додатком;
- емоційна реакція - як користувач почувається після завершення завдання.

4.2. Розгортання програмного продукту

Розроблений програмний продукт орієнтований на інтеграцію з операційною системою Windows. Для коректної роботи клієнтської частини, серверна частина системи не повинна бути повністю налаштована. Усі дані, які потрібні для роботи додатку мають бути присутніми у БД програмного продукту.

Для функціонування клієнтської частини на комп’ютер потрібно відкрити FashionHouse.exe файл, надавши йому потрібні дозволи. Відкриття програми є стандартною процедурою для користувачів персональних комп’ютерів із операційною системою Windows та не потребує спеціальних знань чи вмінь.

Для коректної роботи пакету необхідна користувацька машина з процесором не менше 200 MHz, оперативною пам’яттю не менше 256 Mb.

Розроблений продукт буде повністю сумісний із операційними системами Windows, починаючи із Windows 7.

Вимоги до складу і параметрів технічних засобів

Системні вимоги до сервера

Вимоги з боку апаратного забезпечення серверної частини

- Оперативна пам’ять – 8 Гб.

- Дискова пам’ять – 150Гб.
- Швидкодія процесора – 2GHz

Вимоги з боку програмного забезпечення серверної частини

- Операційна система - OS Windows 8, Microsoft
- .NET Framework 4.5, Microsoft
- Sql Server 2012, Microsoft

Системні вимоги до клієнта

Вимоги з боку апаратного забезпечення клієнтської частини

- Оперативна пам’ять – 4Гб.
- Швидкодія процесора – 1GHz

Вимоги з боку програмного забезпечення клієнтської частини

- Операційна система - OS Windows 8, Microsoft
- .NET Framework 4.0, Microsoft

4.3. Інструкція користувача

Далі наведемо інструкцію для користувача із детальним описом способів використання основних функціональних можливостей розробленого програмного продукту для обліку замовлень ательє з пошиття одягу.

Для авторизації у програмі користувачу потрібно ввести свій логін та пароль та натиснути кнопку підтвердження.

На рисунку 4.1 наведено загальний вигляд вікна, яке реалізує функцію авторизації користувача в програмі.

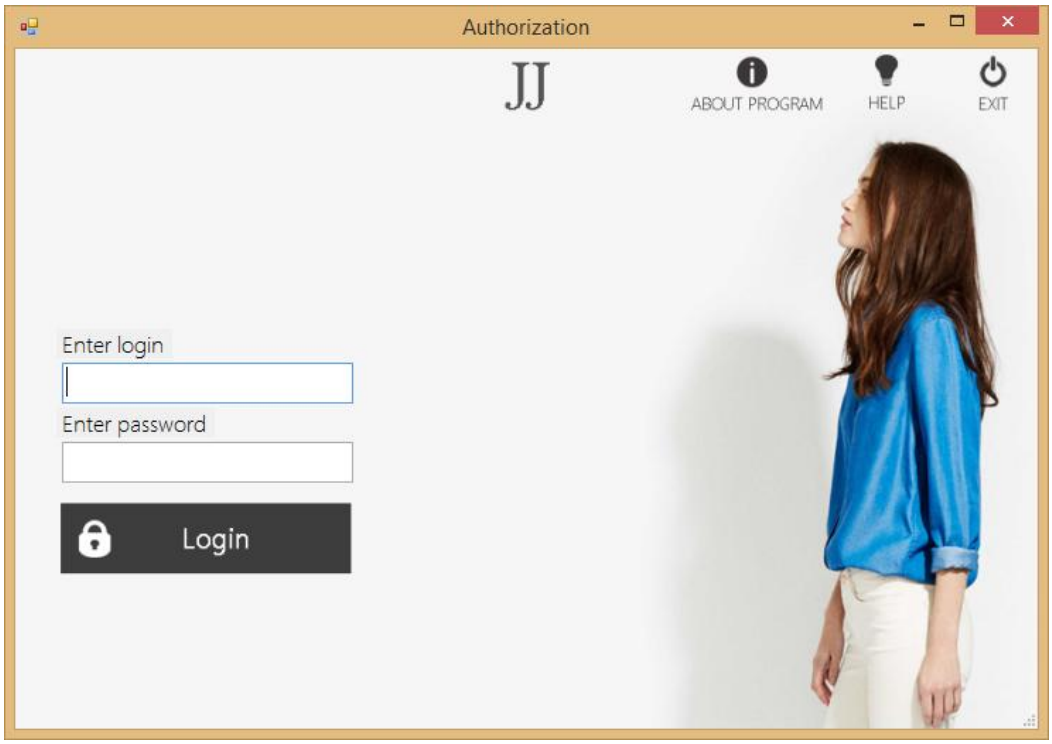

Рисунок 4.1 - Вікно для авторизації в програмі

Залежно від введення відповідних даних, майстра чи адміністратора, програма відкриває головну сторінку користувача. Головне вікно та основні кнопки у ньому зображено на рисунку 4.2.

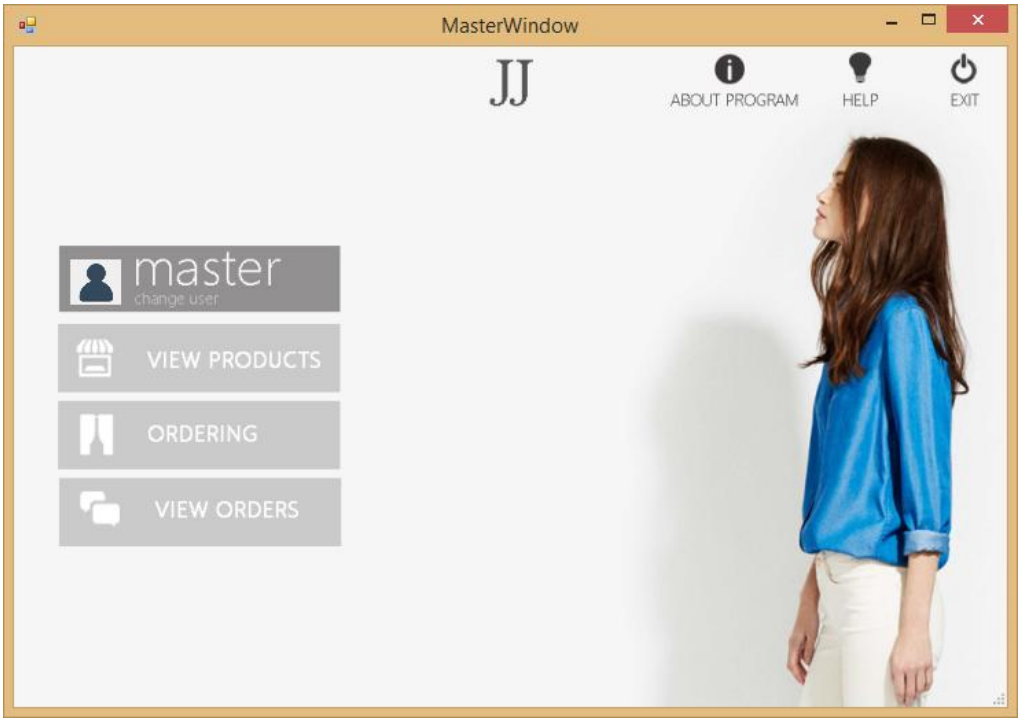

Рисунок 4.2 - Основне вікно майстра

Щоб змінити користувача натисніть кнопку “Change user”, як зображено на рисунку 4.3.

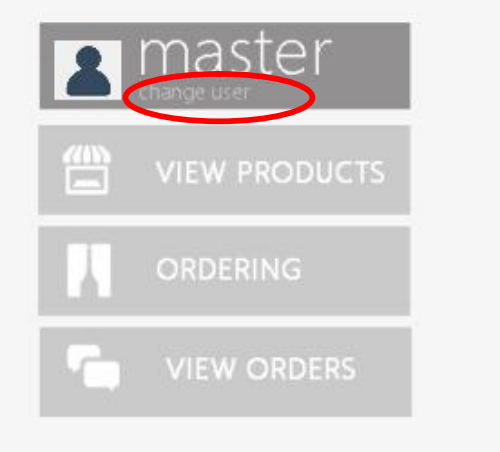

Рисунок 4.3 - Вікно з кнопкою зміни користувача

Щоб переглянути наявність усіх продуктів натисніть на головному меню кнопку “View products” та оберіть бажаний продукт, як наведено на рисунку 4.4.

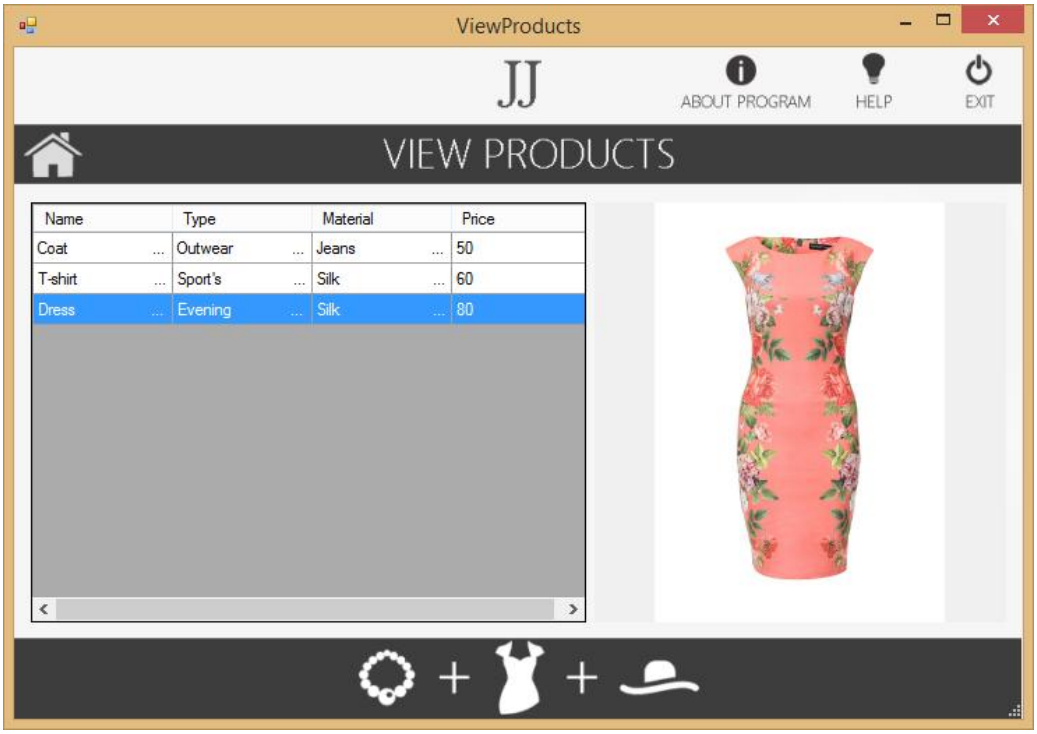

Рисунок 4.4 - Вікно перегляду продуктів

Щоб повернутись до попереднього меню, натисніть кнопку будиночка, яка зображена на рисунку 4.5.

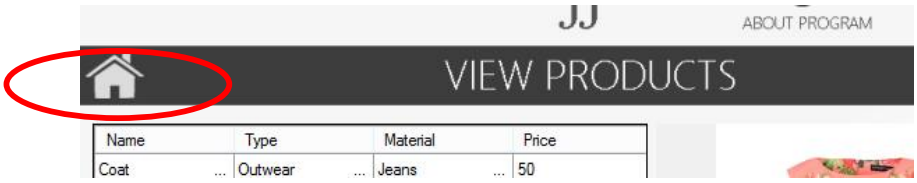

Рисунок. 4.5 - Вікно для переходу до попереднього меню

Щоб зробити замовлення натисніть на головному меню кнопку “Ordering” та оберіть “Custom Sewing”, як проілюстровано на рисунку 4.6

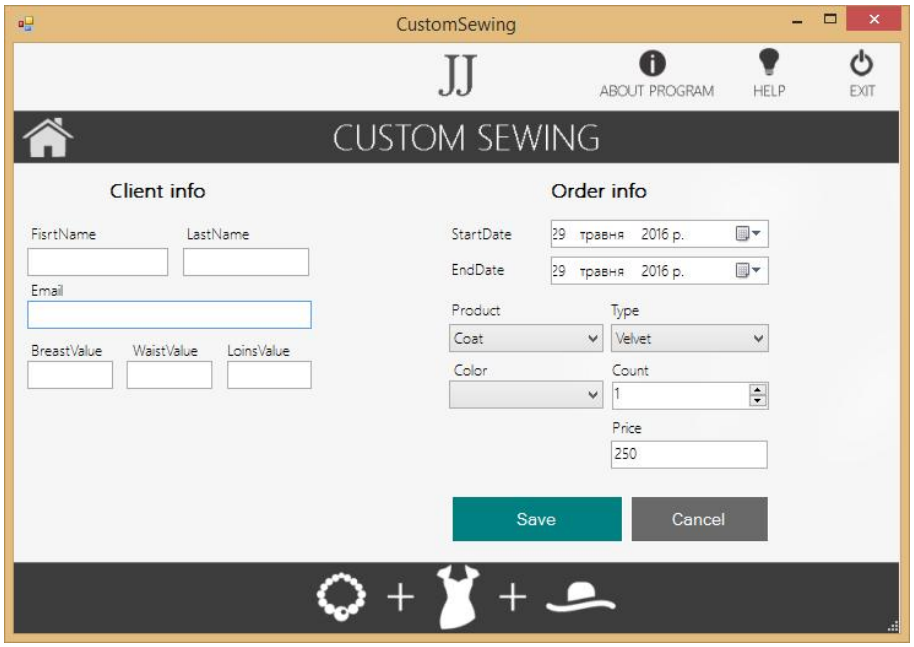

Рисунок 4.6 - Вікно формування замовлення

Щоб підтвердити замовлення, яке заповнено, натисніть на кнопку “Save” або відмінити – кнопку “Cancel”. Дані кнопки зображено на рисунку 4.7.

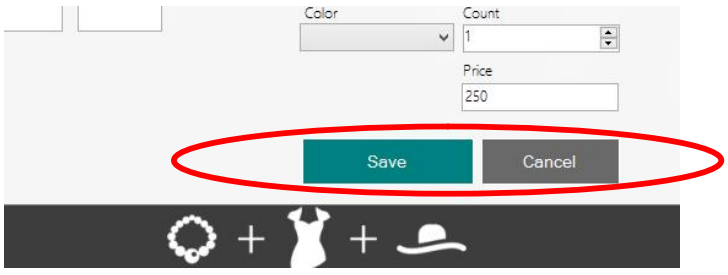

Рисунок 4.7 - Кнопки замовлення продукту

Щоб переглянути існуючі замовлення натисніть на головному меню кнопку “View Orders” та оберіть бажану дату, як проілюстровано на рисунку 4.8

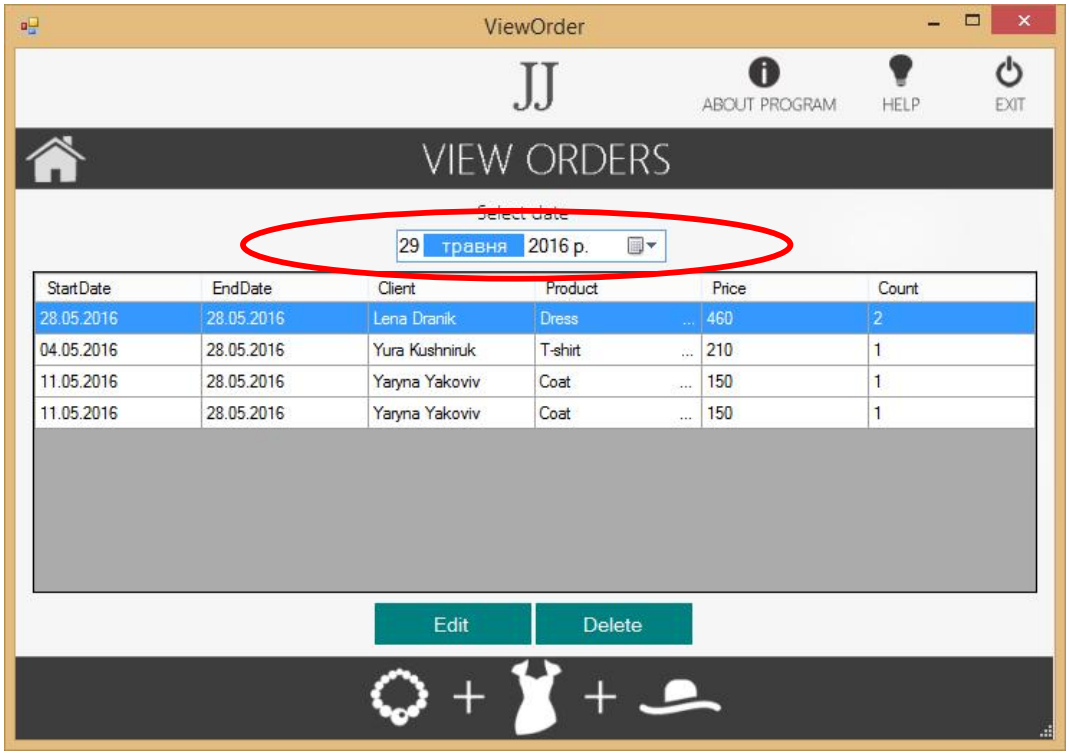

Рисунок 4.8 - Вікно перегляду замовлень

Щоб відредагувати чи видалити замовлення натисніть на відповідні кнопки, проілюстровані на рисунку 4.9.

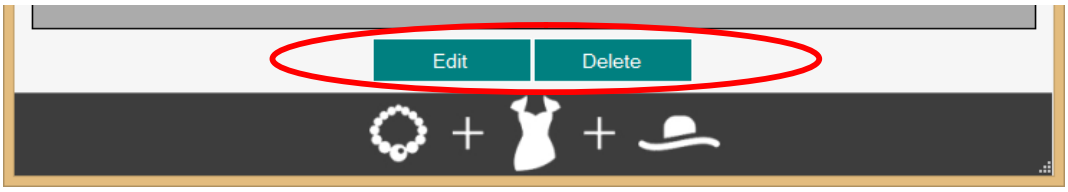

Рисунок 4.9 - Кнопки перегляду замовлень

Щоб додати новий продукт, потрібно бути в системі авторизованим як адміністратор та на головному меню натиснути кнопку “Acts with products” та заповнити відповідні поля, як відображено на рисунку 4.10.

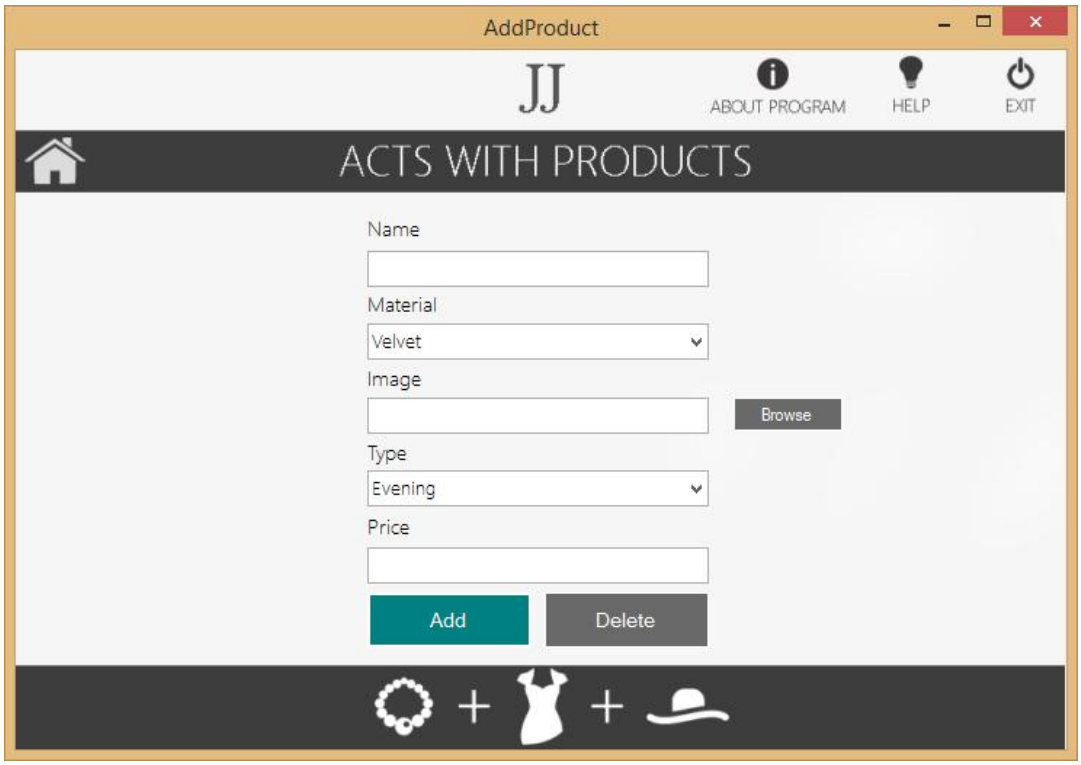

Рисунок 4.10 - Вікно додавання нового продукту

Щоб видалити будь-який продукт натисніть на кнопку “Delete” та оберіть бажаний продукт, як наведено на рисунку 4.11.

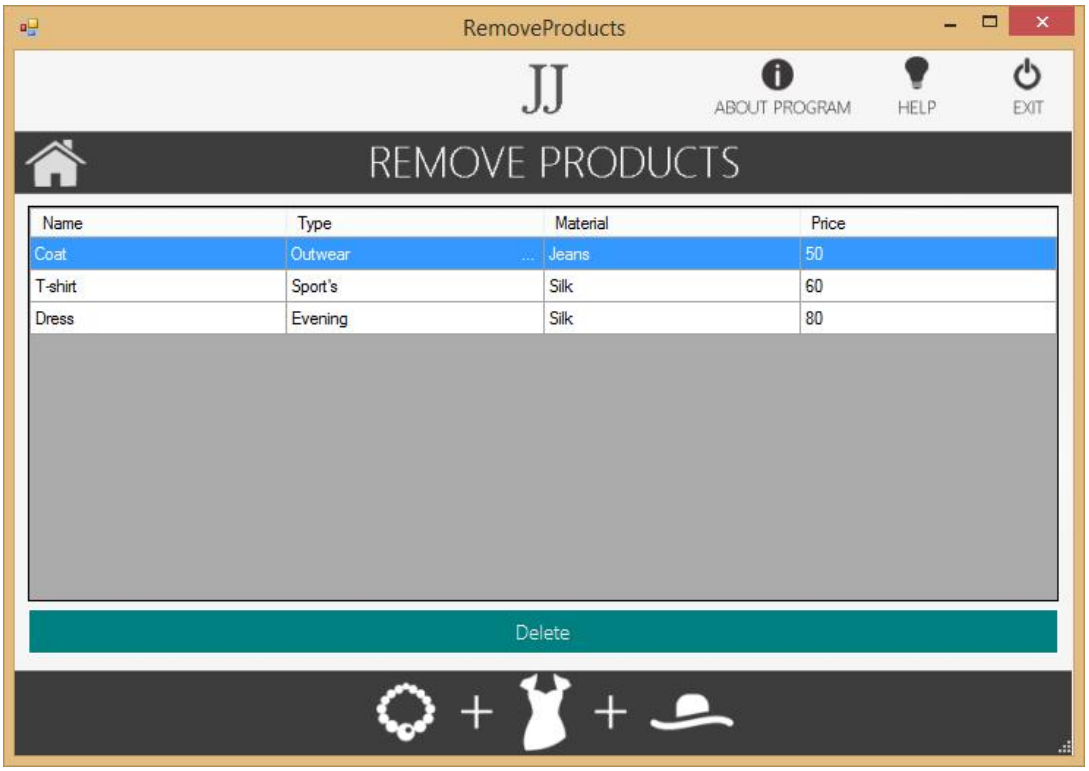

Рисунок 4.11- Вікно видалення продукту

У вікні «Help» наводяться основні функції для користувача та їх пояснення у користуванні, вигляд яких проілюстровано на рисунку 4.12.

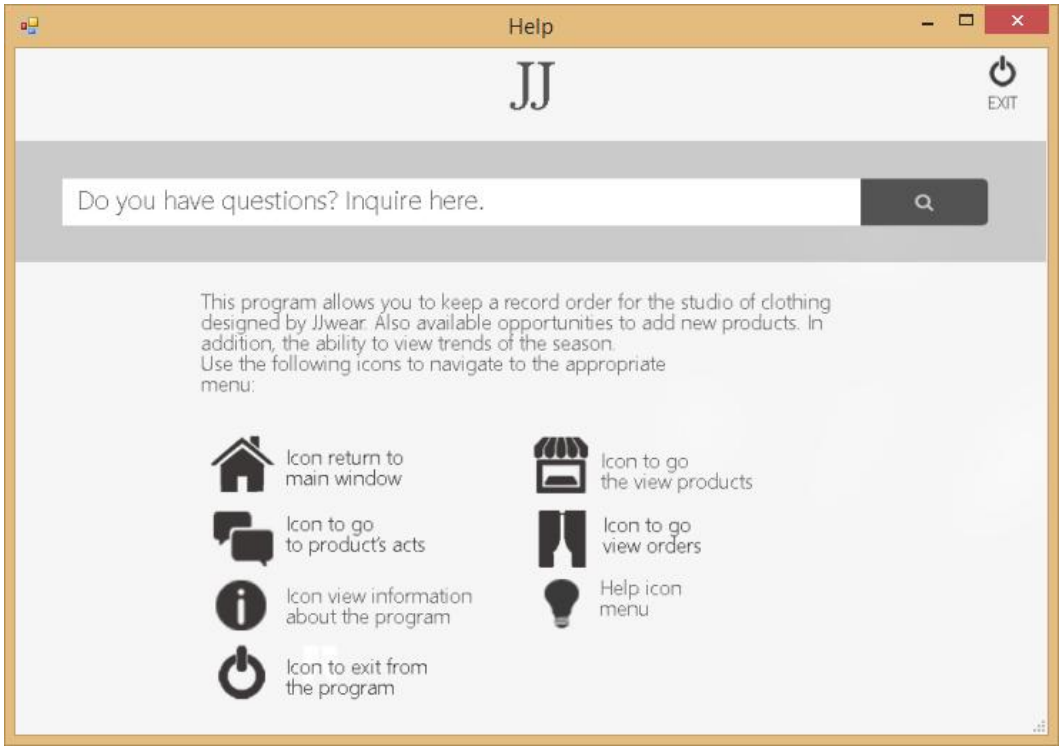

Рисунок 4.12 - Вікно допомоги користувачеві

Щоб вийти з програми натисніть на кнопку “Exit”, як зображено на рисунку 4.13.

Рисунок 4.13 - Вікно виходу з програми

Висновки до четвертого розділу

Даний розділ описує результати проведеного тестування над реалізованим програмного продукту для обліку замовлень ательє з пошиття одягу, описує апаратні та програмні засоби, які необхідні для функціонування розробленого програмного продукту та містить створену інструкцію користувача.

Умови тестування, які визнавалися успішними були наступні:

Розробка тестів:

- Всі заплановані тестові випадки розроблено;
- Покриття тестами програмних вимог досягає 100%;
- Покриття тестами варіантів використання досягає 100%;

Тестування:

- Всі розроблені тестові випадки виконано;
- Виконано тестування продуктивності, вимоги продуктивності задоволено;
- Всі внутрішні дефекти виправлені і виправлення підтверджено. Всі наведені умови задоволено, проект вважається успішним.

ВИСНОВКИ

Під час створення програмного продукту було виявлено його важливість і актуальність в сучасному суспільстві. Облік замовлень для ательє з пошиття одягу є дуже популярним в наш час і хорошим методом для спрощення роботи з документами.

У процесі розробки даного продукту було виконано наступні завдання:

1. Наведено коротку характеристику об’єкту управління, описано напрями його діяльності, розроблено схему організаційної структури.

2. Визначено склад функцій, що входять до бізнес-процесу на основі яких розроблено схему управління бізнес-процесом.

3. Проведено критичний аналіз існуючих аналогів, що реалізують аналогічні функції.

4. Розроблено специфікацію вимог до програмної системи для моделювання роботи скінченних автоматів.

5. Розроблено структурну схему програмної системи, побудовано UML діаграми, що реалізують логіку основних класів.

6. Проведено проектування бази даних програмної системи в рамках якого спроектовано її фізичну модель.

7. Розроблено програмний продукт та визначення актуальність мови програмування та бази даних.

8. Проведено тестування та експлуатацію даного продукту.

Отже, даних програмний продукт на основі усіх вище згаданих розділів виконаний і готовий до експлуатації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Електронні ресурси	1. Бібліотека MSDN - цінне джерело інформації для розробників, які використовують засоби, продукти, технології та служби корпорації Майкрософт [Електронний ресурс]. — Режим доступу: http://msdn.microsoft.com, для доступу до інформаційних ресурсів авторизація не потрібна. — Назва з екрану. Библиотека MSDN
	2. uk.wikipedia.org – Вільна енциклопедія [Електронний ресурс]. – Режим доступу: http://uk.wikipedia.org для доступу до інформаційних ресурсів авторизація не потрібна — Назва з екрану. ВікіпедіЯ. Вільна енциклопедія.
	3. w3schools.com – the world’s largest development site. [Електронний ресурс]. — Режим доступу: http://w3schools.com/, для доступу до інформаційних ресурсів авторизація не потрібна. — Назва з екрану. W3Schools – Educate Yourself.
	4. C# Language Specification Version 4.0. [Електронний ресурс]: офіційна специфікація мови програмування / Microsoft Corporation. — Систем. вимоги: Pentium-266 ; 32 Mb RAM ; Windows 98/2000/NT/XP/7, Microsoft Office Word — Назва з титул. екрану. C# Language Specification Version 4.0.
	5. HTML specification Version 4.01 [Електронний ресурс]: офіційна специфікація мови розмітки гіпертекстових посилань / W3C. Режим доступу: http://www.w3.org/TR/REC-html40/, для доступу до інформаційних ресурсів авторизація не потрібна. — Назва з екрану. World Wide Web Consortium (W3C).

ДОДАТОК А

Таблиця 1.1

Глосарій основних термінів

Термін	Опис терміну
1. Основні поняття та категорії предметної області та проекту	
Ательє	салон індивідуального пошиття одягу
Виріб	предмет або набір предметів, що виготовляються на підприємств
Матеріал	речовина, або суміш речовин, первинний предмет праці, який використовують для виготовлення виробу
Ремонт	процес зміни, відновлення, покращення будь-чого, доведення об'єкта до початкових характеристик
Тренд	загальна тенденція при різнонаправленому русі, визначена загальною спрямованістю змін показників часового ряду
Фасон	Крій, модель, зразок, за якими шиють (або пошито) одяг, взуття, головні убори тощо
Інтерфейс	засіб зручної взаємодії користувача з інформаційною системою
Фірма	Торгівельне підприємство, що користується правом юридичної особи, під маркою якої продаються товари або надаються послуги
Стиль	усталена форма художнього самовизначення епохи, регіону, нації, соціальної або творчої групи або окремої особистості
2. Користувачі системи	
Майстер	особа, що використовує систему обліку замовлень для того, щоб формувати замовлення та вести їхній облік.
Адміністратор	користувач системи, який здійснює підтримку системи для обліку системи замовлень
3. Вхідні та вихідні документи	
Замовлення	комерційний документ у процесі закупівлі, який видається покупцем постачальнику й зазначає тип, кількість, якість, ціну та іншу інформацію про товари чи послуги, яку робить покупець
Звіт	письмове повідомлення про виконання певної роботи
База даних	впорядкований набір логічно взаємопов'язаних даних, що використовуються спільно та призначені для задоволення інформаційних потреб користувачів

ДОДАТОК Б

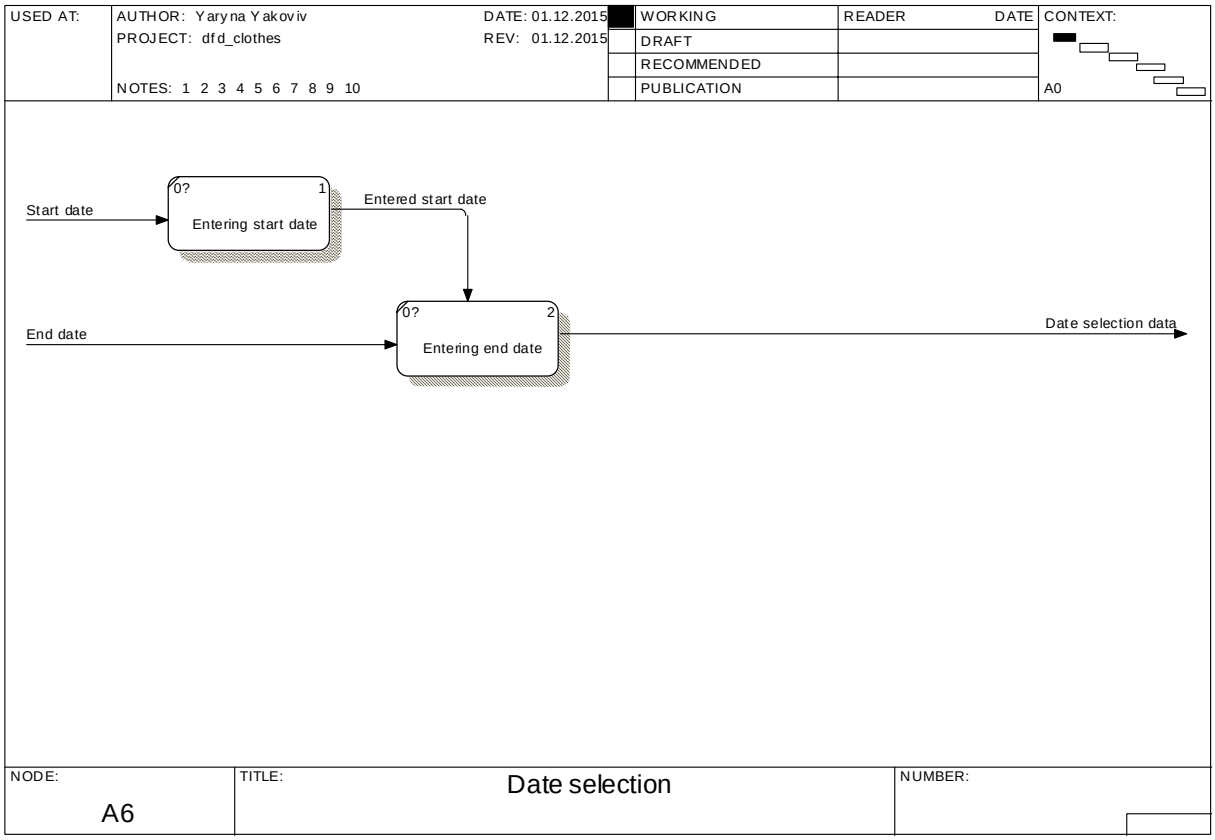

Рисунок 1.1 – Діаграма декомпозиції для вибору дати

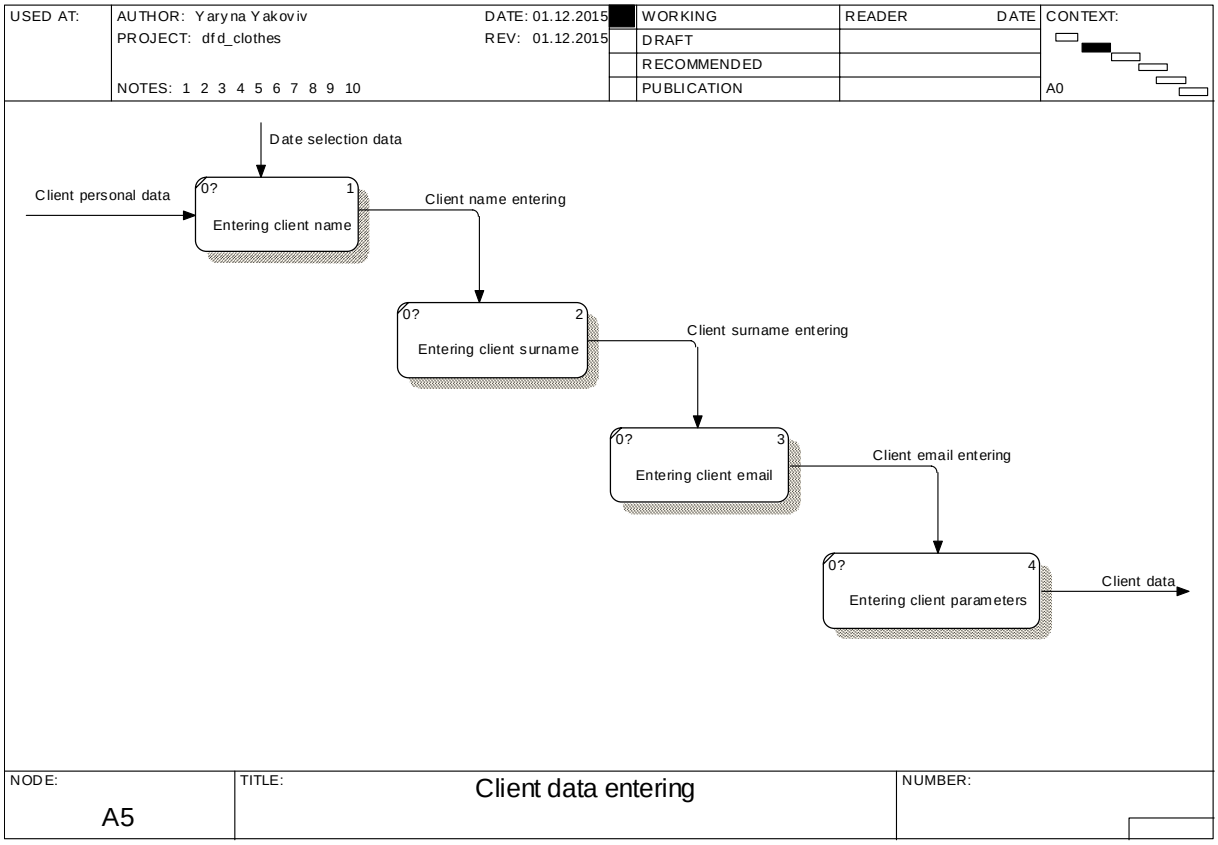

Рисунок 1.2 – Діаграма декомпозиції для введення даних про клієнта

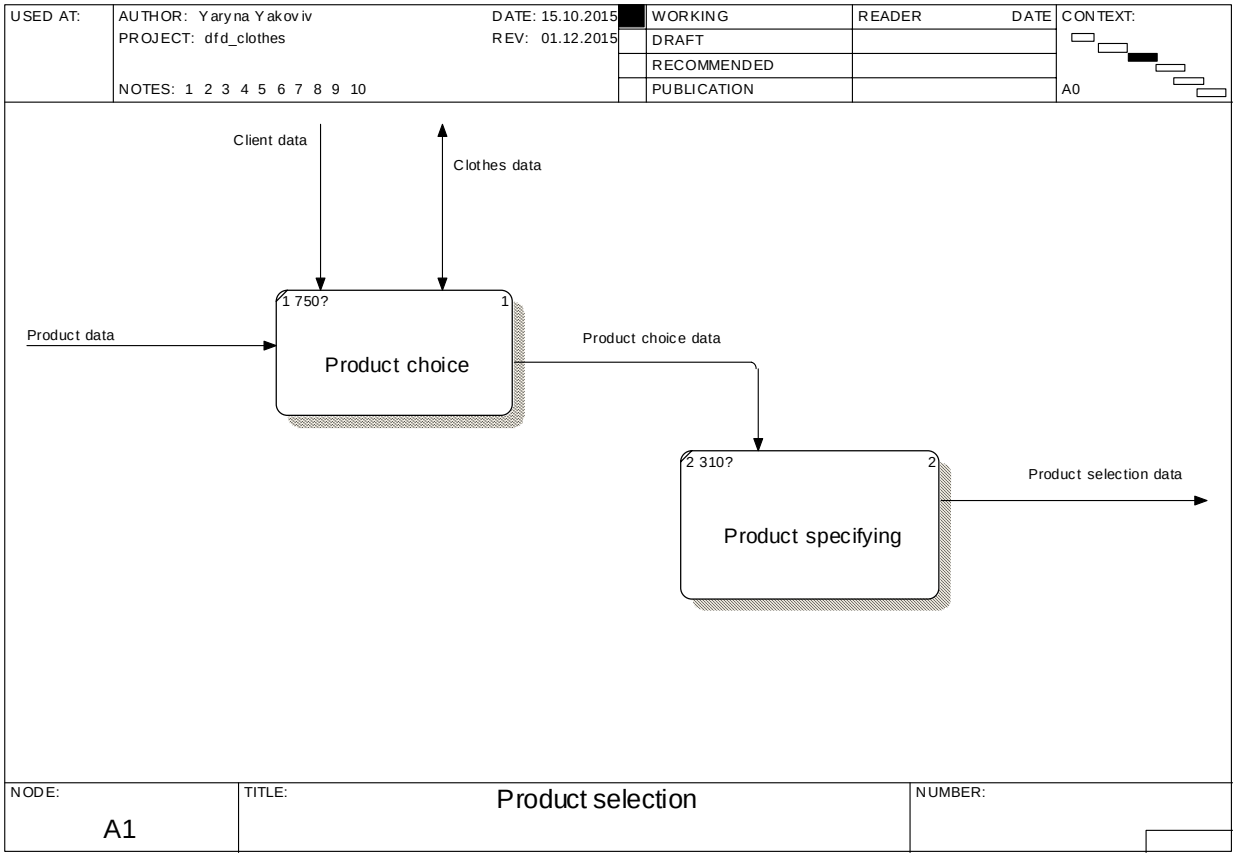

Рисунок 1.3 – Діаграма декомпозиції для вибору продукту

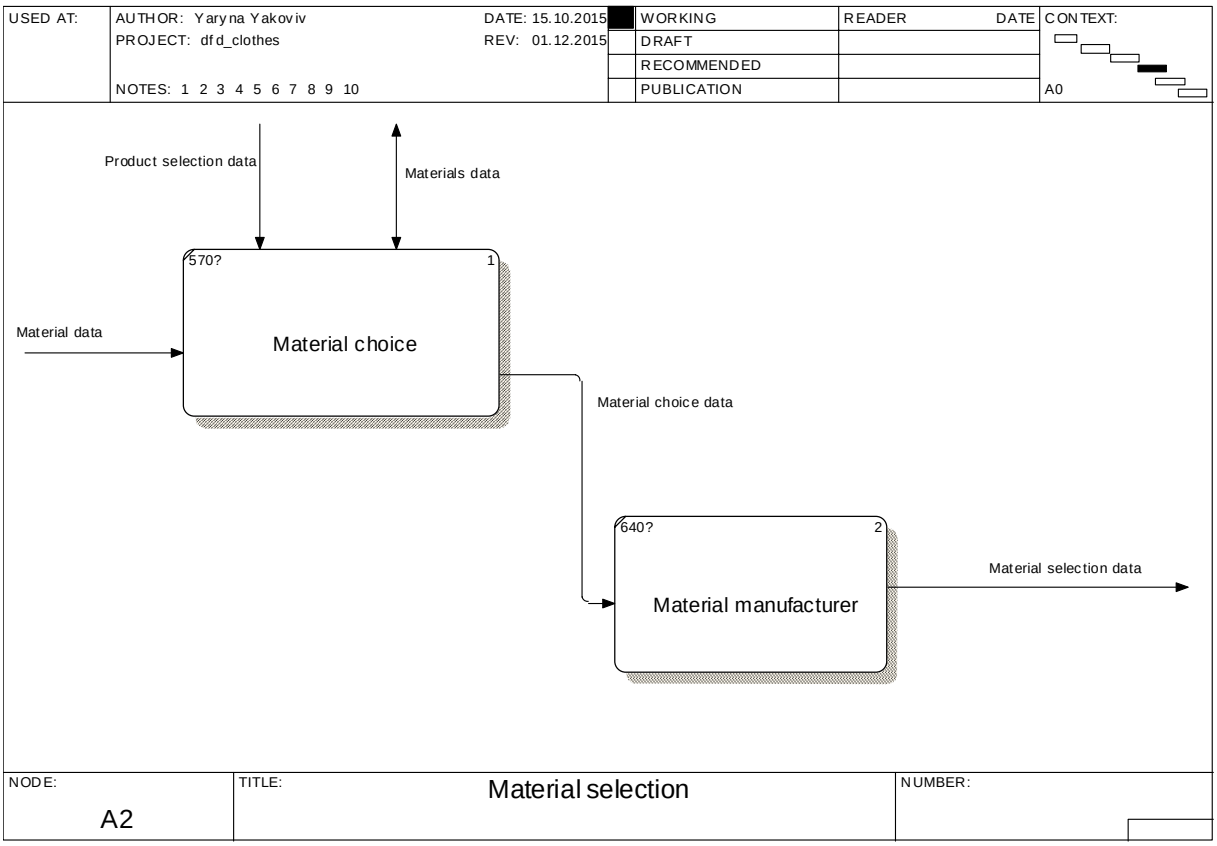

Рисунок 1.4 – Діаграма декомпозиції для вибору матеріалу

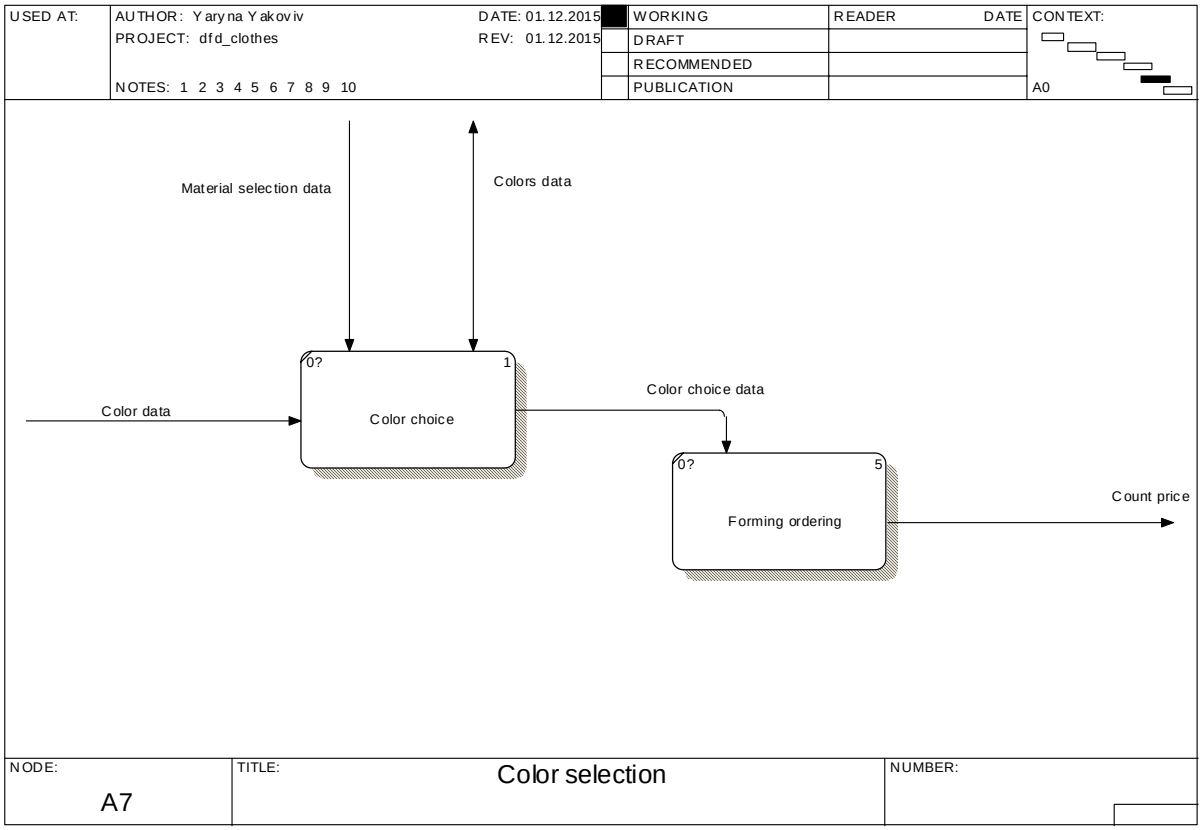

Рисунок 1.5 – Діаграма декомпозиції для вибору кольору

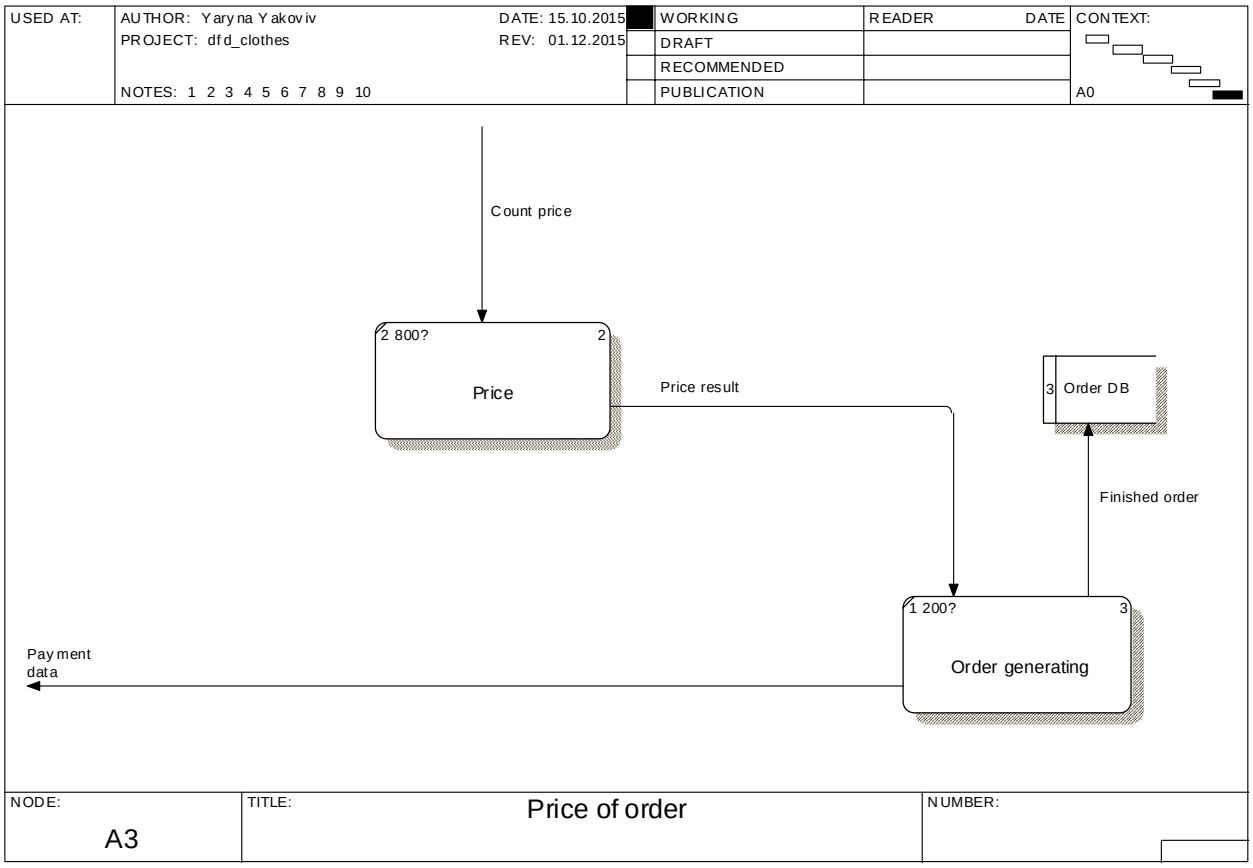

Рисунок 1.6 – Діаграма декомпозиції для вибору ціни

ДОДАТОК В

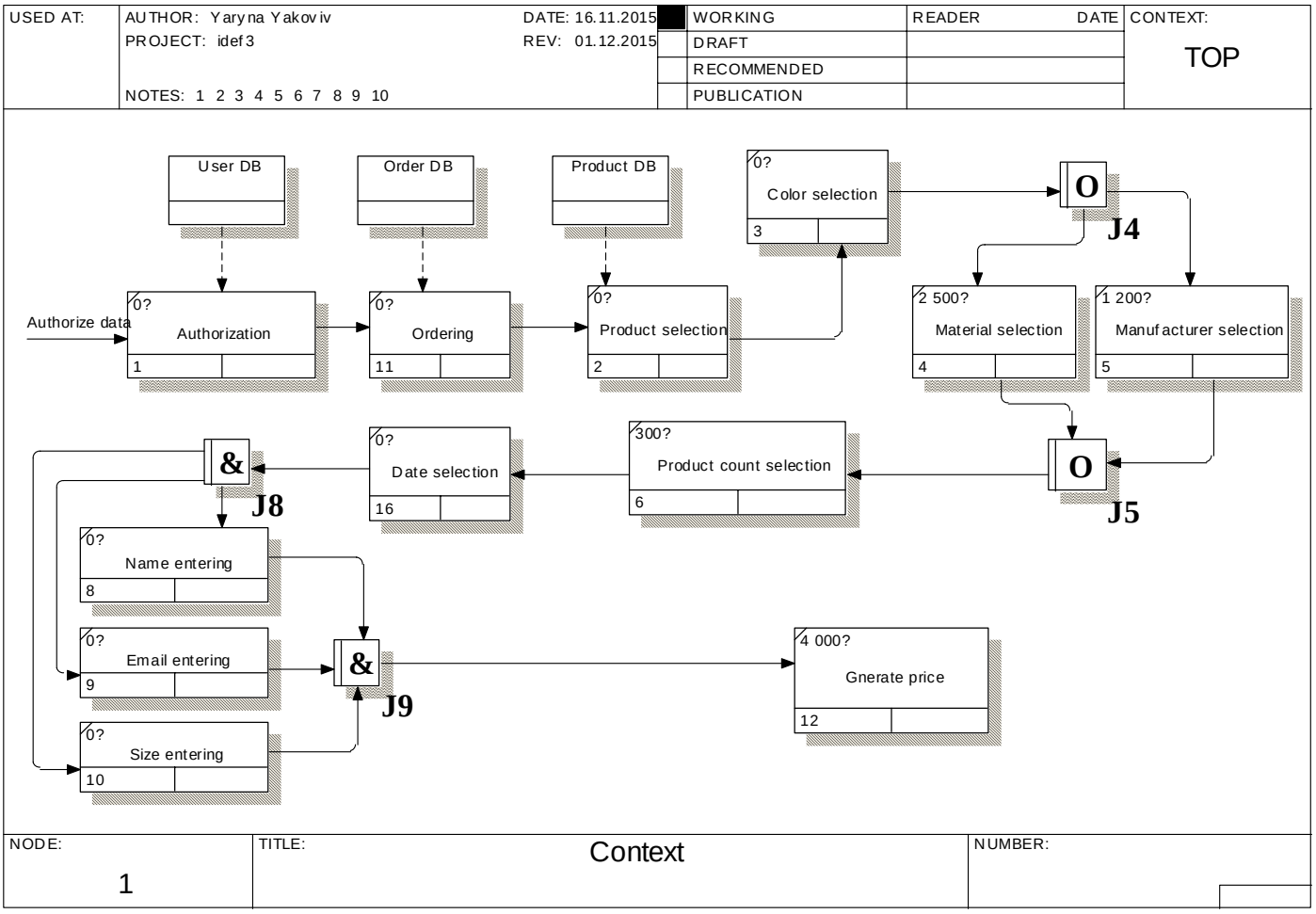

Рисунок 1.7 – Діаграма IDEF3

ДОДАТОК Г

Клас Клієнт

```
public class Client
{
    #region Properties

    private Guid id;

    public Guid Id
    {
        get { return id; }
        set { id = value; }
    }

    private String firstName;

    public String FirstName
    {
        get { return firstName; }
        set { firstName = value; }
    }

    private String lastName;

    public String LastName
    {
        get { return lastName; }
        set { lastName = value; }
    }

    private String email;

    public String Email
    {
        get { return email; }
        set { email = value; }
    }

    private Size size;

    public Size Size
    {
        get { return size; }
        set { size = value; }
    }

    #endregion

    #region Methods

    public Client()
    {

    }

    public Client(Guid id, String firstName, String lastName, String email, Size size)
    {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.size = size;
    }
```

```
    public override string ToString()
    {
        return this.firstName+" "+this.LastName;
    }
    #endregion

}
```

Клас Колір

```
public class Color
{
    #region Properties

    private Guid id;

    public Guid Id
    {
        get { return id; }
        set { id = value; }
    }

    private String colorName;

    public String ColorName
    {
        get { return colorName; }
        set { colorName = value; }
    }

    #endregion

    #region Methods

    public Color()
    {

    }

    public Color(Guid id, string colorName)
    {
        this.id = id;
        this.colorName = colorName;
    }

    #endregion    }
```

Клас Матеріал

```
public class Material
{
    public static List<Material> listOfMats = new List<Material>();

    public Guid id;

    private String materialName;

    public String MaterialName
    {
        get { return materialName; }
        set { materialName = value; }
    }

    private String countryDepartment;

    public String CountryDepartment
```

```
    {
        get { return countryDepartment; }
        set { countryDepartment = value; }
    }

    private float price;

    public float Price
    {
        get { return price; }
        set { price = value; }
    }

    public Material()
    {
    }

    public Material(String materialName, String countryDep, float price)
    {
        this.materialName = materialName;
        this.countryDepartment = countryDep;
        this.price = price;
    }

    public override string ToString()
    {
        return this.materialName;
    }

}
```

Клас Замовлення

```
public class Order
{
    #region Properties

    public Guid id;

    private DateTime startdate;

    public DateTime StartDate
    {
        get { return startdate; }
        set { startdate = value; }
    }

    private DateTime enddate;

    public DateTime EndDate
    {
        get { return enddate; }
        set { enddate = value; }
    }

    private Client client;

    public Client Client
    {
        get { return client; }
        set { client = value; }
    }

    private Product product;

    public Product Product
```

```
        {
            get { return product; }
            set { product = value; }
        }

        private Material material;


        public String color;



        private float price;

        public float Price
        {
            get { return price; }
            set { price = value; }
        }

        private int count;
        public int Count
        {
            get { return count; }
            set { count = value; }
        }

        #endregion

        #region Methods

        public Order()
        {

        }

        public Order(Guid id, DateTime startdate, DateTime enddate, Client client, Product
product, Material material, String color, float price, int count)
        {
            this.id = id;
            this.startdate = startdate;
            this.enddate = enddate;
            this.client = client;
            this.product = product;
            this.material = material;
            this.color = color;
            this.price = price;
            this.count = count;
        }

        #endregion
    }
```

Клас Продукт

```
    public class Product
    {
        #region Properties

        public Guid Id;

        public static List<String> categories = new List<string>
        {
            "Evening","Casual","Outwear","Sport's"
        };
```

```
        private String name;

        public String Name
        {
            get { return name; }
            set { name = value; }
        }

        private String type;

        public String Type
        {
            get { return type; }
            set { type = value; }
        }

        private Material material;

        public Material Material
        {
            get { return material; }
            set { material = value; }
        }

        public String Image;

        private float price;

        public float Price
        {
            get { return price; }
            set { price = value; }
        }

        #endregion

        #region Methods

        public Product()
        {

        }

        public Product(Guid id, String name, String type, Material material, String image,
float price)
        {
            this.Id = id;
            this.name = name;
            this.type = type;
            this.material = material;
            this.Image = image;
            this.price = price;
        }

        public override string ToString()
        {
            return this.Name;
        }

        #endregion

    }
```

Клас Розмір

```
    public class Size
```

```
    {

        #region Properties

        private Guid id;

        public Guid Id
        {
            get { return id; }
            set { id = value; }
        }

        private float breastvalue;

        public float BreastValue
        {
            get { return breastvalue; }
            set { breastvalue = value; }
        }

        private float waistvalue;

        public float WaistValue
        {
            get { return waistvalue; }
            set { waistvalue = value; }
        }

        private float loinsvalue;

        public float LoinsValue
        {
            get { return loinsvalue; }
            set { loinsvalue = value; }
        }

        #endregion

        #region Methods

        public Size()
        {

        }

        public Size(Guid id, float breastvalue, float waistvalue, float loinsvalue)
        {
            this.id = id;
            this.breastvalue = breastvalue;
            this.waistvalue = waistvalue;
            this.loinsvalue = loinsvalue;
        }

        #endregion
}
```

Stored Procedure

- Додавання клієнта

```
USE [FashionHouse]
GO
/****** Object:  StoredProcedure [dbo].[AddNewClient]    Script Date: 29.05.2016 23:59:50
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[AddNewClient]
(
	@ID uniqueidentifier,
	@FirstName varchar(50),
	@LastName varchar(50),
	@Email varchar(50),
	@SizeId uniqueidentifier

)
AS
BEGIN
	INSERT INTO Client(ID,FirstName,LastName,Email,SizeId)
	VALUES (@ID,@FirstName, @LastName, @Email, @SizeId)
	END
```

- Додавання замовлення

```
USE [FashionHouse]
GO
/****** Object:  StoredProcedure [dbo].[AddNewOrder]    Script Date: 30.05.2016 0:00:45
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[AddNewOrder]
(
	@StartDate date,
	@EndDate date,
	@ClientId uniqueidentifier,
	@ProductId uniqueidentifier,
	@MaterialId uniqueidentifier,
	@Color varchar(50),
	@Price float,
	@kount int

)
AS
BEGIN
	INSERT INTO
Ordering(id,StartDate,EndDate,ClientId,ProductId,MaterialId,Color,Price,kount)
	VALUES (NEWID(),@StartDate, @EndDate, @ClientId,
@ProductId,@MaterialId,@Color,@Price,@kount)
	END
```

- Додавання продукту

```
USE [FashionHouse]
GO
/****** Object:  StoredProcedure [dbo].[AddNewProduct]    Script Date: 30.05.2016 0:01:11
******/
```

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[AddNewProduct]
(
	@ProductName nchar(50),
	@ProductType nchar(50),
	@MaterialId uniqueidentifier,
	@Image nchar(50),
	@Price float
)
AS
BEGIN
	INSERT INTO Product(id,ProductName,ProductType,MaterialId,Image,Price)
	VALUES (NEWID(),@ProductName, @ProductType, @MaterialId, @Image,@Price)
	END
```

- Додавання розміру

```
USE [FashionHouse]
GO
/****** Object:  StoredProcedure [dbo].[AddNewSize]    Script Date: 30.05.2016 0:01:32
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[AddNewSize]
(
	@ID uniqueidentifier,
	@BreastValue float,
	@WaistValue float,
	@LoinsValue float

)
AS
BEGIN
	INSERT INTO Size(ID,BreastValue,WaistValue,LoinsValue)
	VALUES (@ID,@BreastValue, @WaistValue, @LoinsValue)
	END
```

- Видалення замовлення

```
USE [FashionHouse]
GO
/****** Object:  StoredProcedure [dbo].[DeleteOrder]    Script Date: 30.05.2016 0:01:57
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[DeleteOrder]
	@id uniqueidentifier
AS
BEGIN
	SET NOCOUNT ON;

	DELETE FROM Ordering
	WHERE ID=@id
END
```

- Видалення продукту

```
USE [FashionHouse]
GO
/****** Object:  StoredProcedure [dbo].[DeleteProduct]    Script Date: 30.05.2016 0:02:22
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[DeleteProduct]
	@id uniqueidentifier
AS
BEGIN
	SET NOCOUNT ON;

	DELETE FROM Product
	WHERE Id=@id
END
```

- Отримання замовлення

```
USE [FashionHouse]
GO
/****** Object:  StoredProcedure [dbo].[GetOrders]    Script Date: 30.05.2016 0:02:52 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- Batch submitted through debugger:
SQLQuery4.sql|7|0|C:\Users\1D4F~1\AppData\Local\Temp\~vs15B5.sql

CREATE PROCEDURE [dbo].[GetOrders]
AS
BEGIN
SELECT *
FROM Ordering
END
```

- Отримання продуктів

```
USE [FashionHouse]
GO
/****** Object:  StoredProcedure [dbo].[GetProducts]    Script Date: 30.05.2016 0:03:11
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- Batch submitted through debugger:
SQLQuery4.sql|7|0|C:\Users\1D4F~1\AppData\Local\Temp\~vs15B5.sql

CREATE PROCEDURE [dbo].[GetProducts]
AS
BEGIN
SELECT *
FROM Product
END
```

- Оновлення клієнта

```
USE [FashionHouse]
GO
/****** Object:  StoredProcedure [dbo].[UpdateClient]    Script Date: 30.05.2016 0:03:29
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
```

```
GO

CREATE PROCEDURE [dbo].[UpdateClient]
	@ID uniqueidentifier,
	@FirstName varchar(50),
	@LastName varchar(50),
	@Email varchar(50)

AS
BEGIN
	SET NOCOUNT ON;
	UPDATE Client
SET FirstName=@FirstName
WHERE ID=@ID

UPDATE Client
SET LastName=@LastName
WHERE ID=@ID

UPDATE Client
SET Email=@Email
WHERE ID=@ID

END
```

- Оновлення замовлення

```
USE [FashionHouse]
GO
/****** Object:  StoredProcedure [dbo].[UpdateOrder]    Script Date: 30.05.2016 0:03:42
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[UpdateOrder]
	@id uniqueidentifier,
	@StartDate date,
	@EndDate date,
	@ProductId uniqueidentifier,
	@MaterialId uniqueidentifier,
	@Color varchar(50),
	@Price float,
	@kount int

AS
BEGIN
	SET NOCOUNT ON;
	UPDATE Ordering
SET StartDate=@StartDate
WHERE ID=@id


UPDATE Ordering
SET EndDate=@EndDate
WHERE ID=@id

UPDATE Ordering
SET Color=@Color
WHERE ID=@id


UPDATE Ordering
SET Price=@Price
WHERE ID=@id
```

```
UPDATE Ordering
SET kount=@kount
WHERE ID=@id

UPDATE Ordering
SET ProductId=@ProductId
WHERE ID=@id

UPDATE Ordering
SET MaterialId=@MaterialId
WHERE ID=@id

END
```

- Оновлення продукту

```
USE [FashionHouse]
GO
/****** Object:  StoredProcedure [dbo].[UpdateProduct]    Script Date: 30.05.2016 0:03:54
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[UpdateProduct]
       @ProductName nchar(50),
       @ProductType nchar(50),
       @MaterialId uniqueidentifier,
       @Image nchar(50),
       @Price float

AS
BEGIN
       SET NOCOUNT ON;
       UPDATE Product
SET ProductName=@ProductName

UPDATE Product
SET ProductType=@ProductType

UPDATE Product
SET @MaterialId=@MaterialId

UPDATE Product
SET @Image=@Image

UPDATE Product
SET @Price=@Price

END
```

- Оновлення розміру

```
USE [FashionHouse]
GO
/****** Object:  StoredProcedure [dbo].[UpdateSize]    Script Date: 30.05.2016 0:04:22
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

```
CREATE PROCEDURE [dbo].[UpdateSize]
	@ID uniqueidentifier,
	@BreastValue float,
	@WaistValue float,
	@LoinsValue float

AS
BEGIN
	SET NOCOUNT ON;
	UPDATE Size
SET BreastValue=@BreastValue
WHERE ID=@ID

UPDATE Size
SET WaistValue=@WaistValue
WHERE ID=@ID

UPDATE Size
SET LoinsValue=@LoinsValue
WHERE ID=@ID

END
```

Код взаємодії модулів системи

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace FashionHouse
{
    public class DataBaseConnector
    {
        private static String connectionString
        {
            get
            {
                return "Data Source = HP; database=FashionHouse; Integrated Security=SSPI;
Trusted_Connection=Yes;";
            }
        }


        public static List<Product> GetProducts()
        {
            List<Product> result = new List<Product>();

            try
            {
                using (SqlConnection con = new SqlConnection(connectionString))
                {
                    con.Open();
                    SqlCommand cmd = new SqlCommand("GetProducts", con);
                    cmd.CommandType = CommandType.StoredProcedure;
                    SqlDataReader dataReader = cmd.ExecuteReader();

                    while (dataReader.Read())
                    {
                        Product curProduct = new Product();
                        Material m = new Material();
                        curProduct.Id = new Guid(Convert.ToString(dataReader["ID"]));
                        curProduct.Name = Convert.ToString(dataReader["ProductName"]);
                        curProduct.Type = Convert.ToString(dataReader["ProductType"]);
                        m.id = new Guid(Convert.ToString(dataReader["MaterialId"]));
                        foreach(Material mat in GetMaterials())
                        {
                            if(mat.id == m.id)
                            {
                                m = mat;
                            }
                        }

                        curProduct.Image = Convert.ToString(dataReader["Image"]);
                        curProduct.Price =
float.Parse(Convert.ToString(dataReader["Price"]));
                        curProduct.Material = m;
                        result.Add(curProduct);
                    }
                }

            }
            catch (Exception e)
            {
```

```
                Console.Write((e));
                return null;
            }

            return result;
        }

        public static List<Material> GetMaterials()
        {
            List<Material> result = new List<Material>();

            try
            {
                using (SqlConnection con = new SqlConnection(connectionString))
                {
                    con.Open();
                    SqlCommand cmd = new SqlCommand("GetMaterials", con);
                    cmd.CommandType = CommandType.StoredProcedure;
                    SqlDataReader dataReader = cmd.ExecuteReader();

                    while (dataReader.Read())
                    {
                        Material cur = new Material();
                        cur.id = new Guid(Convert.ToString(dataReader["ID"]));
                        cur.MaterialName = Convert.ToString(dataReader["MaterialName"]);
                        cur.CountryDepartment =
Convert.ToString(dataReader["CountryDepartment"]);
                        cur.Price = float.Parse(Convert.ToString(dataReader["Price"]));

                        result.Add(cur);
                    }
                }

            }
            catch (Exception e)
            {
                MessageBox.Show(e.Message);
                return null;
            }

            return result;
        }

        public static Size GetSize(Guid g)
        {
            Size c = new Size();
            try
            {
                using (SqlConnection con = new SqlConnection(connectionString))
                {
                    con.Open();
                    SqlCommand cmd1 = new SqlCommand("SELECT * FROM Size WHERE ID=@id", con);
                    cmd1.Parameters.Add(new SqlParameter("@id", g.ToString()));
                    SqlDataReader dataReader1 = cmd1.ExecuteReader();
                    while (dataReader1.Read())
                    {
                        c.Id = g;
                        c.BreastValue =
float.Parse(Convert.ToString(dataReader1["BreastValue"]));
                        c.WaistValue =
float.Parse(Convert.ToString(dataReader1["WaistValue"]));
                        c.LoinsValue =
float.Parse(Convert.ToString(dataReader1["LoinsValue"]));
                    }
                }
            }
```

```
            catch (Exception e)
            {
                MessageBox.Show(e.Message);
                return null;
            }
            return c;
        }
        public static Client GetClient(Guid g)
        {
            Client c = new Client();
            try
            {
                using (SqlConnection con = new SqlConnection(connectionString))
                {
                        con.Open();
                        SqlCommand cmd1 = new SqlCommand("SELECT * FROM Client WHERE ID=@id",
con);
                        cmd1.Parameters.Add(new SqlParameter("@id", g.ToString()));
                        SqlDataReader dataReader1 = cmd1.ExecuteReader();
                        while (dataReader1.Read())
                        {
                            c.Id = g;
                            c.FirstName = Convert.ToString(dataReader1["FirstName"]);
                            c.LastName = Convert.ToString(dataReader1["LastName"]);
                            c.Email = Convert.ToString(dataReader1["Email"]);
                            Guid g1 = new Guid(Convert.ToString(dataReader1["SizeId"]));
                            Size s = GetSize(g1);
                            c.Size = s;
                        }
                }
            }
            catch (Exception e)
            {
                MessageBox.Show(e.Message);
                return null;
            }

            return c;
        }



        public static List<Order> GetOrders()
        {
            List<Order> result = new List<Order>();

            try
            {
                using (SqlConnection con = new SqlConnection(connectionString))
                {
                    con.Open();
                    SqlCommand cmd = new SqlCommand("GetOrders", con);
                    cmd.CommandType = CommandType.StoredProcedure;
                    SqlDataReader dataReader = cmd.ExecuteReader();

                    while (dataReader.Read())
                    {
                        Order cur = new Order();
                        Product curProd = new Product();
                        Material curMat = new Material();
                        cur.id = new Guid(Convert.ToString(dataReader["ID"]));

                        Guid g = new Guid(Convert.ToString(dataReader["ClientId"]));
                        Client c = GetClient(g);

                        cur.StartDate = Convert.ToDateTime(dataReader["StartDate"]);
```

```
                    cur.EndDate = Convert.ToDateTime(dataReader["EndDate"]);
                    cur.Count = Convert.ToInt32(dataReader["Kount"]);
                    cur.Price = float.Parse(Convert.ToString(dataReader["Price"]));
                    List<Product> prods = GetProducts();
                    List<Material> maters = GetMaterials();
                    Guid prodId = new Guid(Convert.ToString(dataReader["ProductId"]));
                    Guid materId = new Guid(Convert.ToString(dataReader["MaterialId"]));
                    cur.color = Convert.ToString(dataReader["Color"]);
                    foreach(Product pr in prods)
                    {
                        if(pr.Id == prodId)
                        {
                            curProd = pr;
                            break;
                        }
                    }
                    foreach (Material m in maters)
                    {
                        if (m.id == materId)
                        {
                            curMat = m;
                            break;
                        }
                    }
                    cur.Product = curProd;
                    cur.Client = c;
                    cur.Product.Material = curMat;

                    result.Add(cur);
                }
            }
        }
        catch (Exception e)
        {
            MessageBox.Show(e.Message);
            return null;
        }
        return result;
    }



    public static void AddNewProduct(Guid id, String name, String type, Material
material, String image, float price)
    {
        Product product = new Product(id, name, type, material, image, price);
        try
        {
            using (SqlConnection con = new SqlConnection(connectionString))
            {
                con.Open();
                SqlCommand command = new SqlCommand("AddNewProduct", con);
                command.CommandType = CommandType.StoredProcedure;

                SqlParameter NameParam = new SqlParameter();
                NameParam.SqlDbType = SqlDbType.NChar;
                NameParam.Value = name;
                NameParam.ParameterName = "@ProductName";

                SqlParameter TypeParam = new SqlParameter();
                TypeParam.SqlDbType = SqlDbType.NChar;
                TypeParam.Value = type;
                TypeParam.ParameterName = "@ProductType";

                SqlParameter MaterialParam = new SqlParameter();
```

```
                    MaterialParam.SqlDbType = SqlDbType.UniqueIdentifier;
                    MaterialParam.Value = material.id;
                    MaterialParam.ParameterName = "@MaterialId";

                    SqlParameter ImageParam = new SqlParameter();
                    ImageParam.SqlDbType = SqlDbType.NChar;
                    ImageParam.Value = image;
                    ImageParam.ParameterName = "@Image";

                    SqlParameter PriceParam = new SqlParameter();
                    PriceParam.SqlDbType = SqlDbType.Float;
                    PriceParam.Value = price;
                    PriceParam.ParameterName = "@Price";

                    command.Parameters.Add(NameParam);
                    command.Parameters.Add(TypeParam);
                    command.Parameters.Add(MaterialParam);
                    command.Parameters.Add(ImageParam);
                    command.Parameters.Add(PriceParam);

                    command.ExecuteNonQuery();

                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        public static void DeleteOrder(Guid id)
        {
            try
            {
                using (SqlConnection con = new SqlConnection(connectionString))
                {
                    con.Open();
                    SqlCommand cmd = new SqlCommand("DELETE FROM Ordering WHERE ID=@id",
con);
                    //cmd.CommandType = CommandType.StoredProcedure;

                    cmd.Parameters.Add(new SqlParameter("@id", id));
                    SqlDataReader dataReader = cmd.ExecuteReader();
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }



        public static void DeleteProduct(Guid id)
        {
            try
            {
                using (SqlConnection con = new SqlConnection(connectionString))
                {
                    con.Open();
                    SqlCommand cmd = new SqlCommand("DELETE FROM Product WHERE ID=@id", con);
                    //cmd.CommandType = CommandType.StoredProcedure;

                    cmd.Parameters.Add(new SqlParameter("@id", id));
                    SqlDataReader dataReader = cmd.ExecuteReader();
                }
```

```
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }


        public static void UpdateProduct(Guid id, String name, String type, Material
material, String image, float price)
        {
            Product order = new Product(id, name, type, material, image, price);

            try
            {
                using (SqlConnection con = new SqlConnection(connectionString))
                {
                    con.Open();
                    SqlCommand command = new SqlCommand("UpdateProduct", con);
                    command.CommandType = CommandType.StoredProcedure;

                    SqlParameter NameParam = new SqlParameter();
                    NameParam.SqlDbType = SqlDbType.NChar;
                    NameParam.Value = name;
                    NameParam.ParameterName = "@ProductName";

                    SqlParameter TypeParam = new SqlParameter();
                    TypeParam.SqlDbType = SqlDbType.NChar;
                    TypeParam.Value = type;
                    TypeParam.ParameterName = "@ProductType";

                    SqlParameter MaterialParam = new SqlParameter();
                    MaterialParam.SqlDbType = SqlDbType.UniqueIdentifier;
                    MaterialParam.Value = material.id;
                    MaterialParam.ParameterName = "@MaterialId";

                    SqlParameter ImageParam = new SqlParameter();
                    ImageParam.SqlDbType = SqlDbType.NChar;
                    ImageParam.Value = image;
                    ImageParam.ParameterName = "@Image";

                    SqlParameter PriceParam = new SqlParameter();
                    PriceParam.SqlDbType = SqlDbType.Float;
                    PriceParam.Value = price;
                    PriceParam.ParameterName = "@Price";

                    command.Parameters.Add(NameParam);
                    command.Parameters.Add(TypeParam);
                    command.Parameters.Add(MaterialParam);
                    command.Parameters.Add(ImageParam);
                    command.Parameters.Add(PriceParam);

                    command.ExecuteNonQuery();

                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        public static void AddNewOrder(Guid id, DateTime startdate, DateTime enddate, String
firstname, String lastname, String email, String BreastValue, String WaistValue, String
LoinsValue, Product product, Material material, String color, String price, String count)
        {
```

```
            Client c = new Client(Guid.NewGuid(), firstname, lastname, email, new
Size(Guid.NewGuid(),float.Parse(BreastValue),float.Parse(WaistValue),float.Parse(LoinsValue))
);
            Order order = new Order(id, startdate, enddate, c, product, material, color,
float.Parse(price), int.Parse(count));

            try
            {
                using (SqlConnection con = new SqlConnection(connectionString))
                {
                    con.Open();
                    SqlCommand command = new SqlCommand("AddNewSize", con);
                    command.CommandType = CommandType.StoredProcedure;

                    SqlParameter SizeParam = new SqlParameter();
                    SizeParam.SqlDbType = SqlDbType.UniqueIdentifier;
                    SizeParam.Value = c.Size.Id;
                    SizeParam.ParameterName = "@ID";

                    SqlParameter BreastValueParam = new SqlParameter();
                    BreastValueParam.SqlDbType = SqlDbType.VarChar;
                    BreastValueParam.Value = BreastValue;
                    BreastValueParam.ParameterName = "@BreastValue";

                    SqlParameter WaistValueParam = new SqlParameter();
                    WaistValueParam.SqlDbType = SqlDbType.VarChar;
                    WaistValueParam.Value = WaistValue;
                    WaistValueParam.ParameterName = "@WaistValue";

                    SqlParameter LoinsValueParam = new SqlParameter();
                    LoinsValueParam.SqlDbType = SqlDbType.VarChar;
                    LoinsValueParam.Value = LoinsValue;
                    LoinsValueParam.ParameterName = "@LoinsValue";


                    command.Parameters.Add(SizeParam);
                    command.Parameters.Add(BreastValueParam);
                    command.Parameters.Add(WaistValueParam);
                    command.Parameters.Add(LoinsValueParam);
                    command.ExecuteNonQuery();

                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }

            try
            {
                using (SqlConnection con = new SqlConnection(connectionString))
                {
                    con.Open();
                    SqlCommand command = new SqlCommand("AddNewClient", con);
                    command.CommandType = CommandType.StoredProcedure;

                    SqlParameter ClientParam = new SqlParameter();
                    ClientParam.SqlDbType = SqlDbType.UniqueIdentifier;
                    ClientParam.Value = c.Id;
                    ClientParam.ParameterName = "@ID";

                    SqlParameter FirstNameParam = new SqlParameter();
                    FirstNameParam.SqlDbType = SqlDbType.VarChar;
                    FirstNameParam.Value = firstname;
                    FirstNameParam.ParameterName = "@FirstName";
```

```
            SqlParameter LastNameParam = new SqlParameter();
            LastNameParam.SqlDbType = SqlDbType.VarChar;
            LastNameParam.Value = lastname;
            LastNameParam.ParameterName = "@LastName";

            SqlParameter EmailParam = new SqlParameter();
            EmailParam.SqlDbType = SqlDbType.VarChar;
            EmailParam.Value = email;
            EmailParam.ParameterName = "@Email";

            SqlParameter SizeParam = new SqlParameter();
            SizeParam.SqlDbType = SqlDbType.UniqueIdentifier;
            SizeParam.Value = c.Size.Id;
            SizeParam.ParameterName = "@SizeId";

            command.Parameters.Add(ClientParam);
            command.Parameters.Add(FirstNameParam);
            command.Parameters.Add(LastNameParam);
            command.Parameters.Add(EmailParam);
            command.Parameters.Add(SizeParam);

            command.ExecuteNonQuery();

        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }

    try
    {
        using (SqlConnection con = new SqlConnection(connectionString))
        {
            con.Open();
            SqlCommand command = new SqlCommand("AddNewOrder", con);
            command.CommandType = CommandType.StoredProcedure;

            SqlParameter StartDateParam = new SqlParameter();
            StartDateParam.SqlDbType = SqlDbType.Date;
            StartDateParam.Value = startdate;
            StartDateParam.ParameterName = "@StartDate";

            SqlParameter EndDateParam = new SqlParameter();
            EndDateParam.SqlDbType = SqlDbType.Date;
            EndDateParam.Value = enddate;
            EndDateParam.ParameterName = "@EndDate";

            SqlParameter ClientParam = new SqlParameter();
            ClientParam.SqlDbType = SqlDbType.UniqueIdentifier;
            ClientParam.Value = c.Id;
            ClientParam.ParameterName = "@ClientId";

            SqlParameter ProductParam = new SqlParameter();
            ProductParam.SqlDbType = SqlDbType.UniqueIdentifier;
            ProductParam.Value = product.Id;
            ProductParam.ParameterName = "@ProductId";

            SqlParameter MaterialParam = new SqlParameter();
            MaterialParam.SqlDbType = SqlDbType.UniqueIdentifier;
            MaterialParam.Value = material.id;
            MaterialParam.ParameterName = "@MaterialId";

            SqlParameter ColorParam = new SqlParameter();
            ColorParam.SqlDbType = SqlDbType.VarChar;
            ColorParam.Value = color;
```

```
                    ColorParam.ParameterName = "@Color";

                    SqlParameter PriceParam = new SqlParameter();
                    PriceParam.SqlDbType = SqlDbType.Float;
                    PriceParam.Value = price;
                    PriceParam.ParameterName = "@Price";

                    SqlParameter CountParam = new SqlParameter();
                    CountParam.SqlDbType = SqlDbType.Int;
                    CountParam.Value = count;
                    CountParam.ParameterName = "@kount";

                    command.Parameters.Add(StartDateParam);
                    command.Parameters.Add(EndDateParam);
                    command.Parameters.Add(ClientParam);
                    command.Parameters.Add(ProductParam);
                    command.Parameters.Add(MaterialParam);
                    command.Parameters.Add(ColorParam);
                    command.Parameters.Add(PriceParam);
                    command.Parameters.Add(CountParam);

                    command.ExecuteNonQuery();

                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        public static void UpdateOrder(Guid id, DateTime StartDate, DateTime EndDate, String
firstname, String lastname, String email, String BreastValue, String WaistValue, String
LoinsValue, Product product, Material material, String Color, String Price, String kount,
Guid clientId,Guid sizeId)
        {
            Client c = new Client(clientId, firstname, lastname, email, new Size(sizeId,
float.Parse(BreastValue), float.Parse(WaistValue), float.Parse(LoinsValue)));
            Order order = new Order(id, StartDate, EndDate, c, product, material, Color,
float.Parse(Price), int.Parse(kount));

            try
            {
                using (SqlConnection con = new SqlConnection(connectionString))
                {
                    con.Open();
                    SqlCommand command = new SqlCommand("UpdateSize", con);
                    command.CommandType = CommandType.StoredProcedure;

                    SqlParameter SizeParam = new SqlParameter();
                    SizeParam.SqlDbType = SqlDbType.UniqueIdentifier;
                    SizeParam.Value = c.Size.Id;
                    SizeParam.ParameterName = "@ID";

                    SqlParameter BreastValueParam = new SqlParameter();
                    BreastValueParam.SqlDbType = SqlDbType.Float;
                    BreastValueParam.Value = BreastValue;
                    BreastValueParam.ParameterName = "@BreastValue";

                    SqlParameter WaistValueParam = new SqlParameter();
                    WaistValueParam.SqlDbType = SqlDbType.Float;
                    WaistValueParam.Value = WaistValue;
                    WaistValueParam.ParameterName = "@WaistValue";

                    SqlParameter LoinsValueParam = new SqlParameter();
                    LoinsValueParam.SqlDbType = SqlDbType.Float;
```

```
            LoinsValueParam.Value = LoinsValue;
            LoinsValueParam.ParameterName = "@LoinsValue";


            command.Parameters.Add(SizeParam);
            command.Parameters.Add(BreastValueParam);
            command.Parameters.Add(WaistValueParam);
            command.Parameters.Add(LoinsValueParam);
            command.ExecuteNonQuery();

        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }

    try
    {
        using (SqlConnection con = new SqlConnection(connectionString))
        {
            con.Open();
            SqlCommand command = new SqlCommand("UpdateClient", con);
            command.CommandType = CommandType.StoredProcedure;

            SqlParameter ClientParam = new SqlParameter();
            ClientParam.SqlDbType = SqlDbType.UniqueIdentifier;
            ClientParam.Value = c.Id;
            ClientParam.ParameterName = "@ID";

            SqlParameter FirstNameParam = new SqlParameter();
            FirstNameParam.SqlDbType = SqlDbType.VarChar;
            FirstNameParam.Value = firstname;
            FirstNameParam.ParameterName = "@FirstName";

            SqlParameter LastNameParam = new SqlParameter();
            LastNameParam.SqlDbType = SqlDbType.VarChar;
            LastNameParam.Value = lastname;
            LastNameParam.ParameterName = "@LastName";

            SqlParameter EmailParam = new SqlParameter();
            EmailParam.SqlDbType = SqlDbType.VarChar;
            EmailParam.Value = email;
            EmailParam.ParameterName = "@Email";

            command.Parameters.Add(ClientParam);
            command.Parameters.Add(FirstNameParam);
            command.Parameters.Add(LastNameParam);
            command.Parameters.Add(EmailParam);

            command.ExecuteNonQuery();

        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }



    try
    {
        using (SqlConnection con = new SqlConnection(connectionString))
        {
            con.Open();
```

```
                    SqlCommand command = new SqlCommand("UpdateOrder", con);
                    command.CommandType = CommandType.StoredProcedure;

                    SqlParameter IdParam = new SqlParameter();
                    IdParam.SqlDbType = SqlDbType.UniqueIdentifier;
                    IdParam.Value = id;
                    IdParam.ParameterName = "@id";

                    SqlParameter StartDateParam = new SqlParameter();
                    StartDateParam.SqlDbType = SqlDbType.DateTime;
                    StartDateParam.Value = StartDate;
                    StartDateParam.ParameterName = "@StartDate";

                    SqlParameter EndDateParam = new SqlParameter();
                    EndDateParam.SqlDbType = SqlDbType.DateTime;
                    EndDateParam.Value = EndDate;
                    EndDateParam.ParameterName = "@EndDate";

                    SqlParameter ProductParam = new SqlParameter();
                    ProductParam.SqlDbType = SqlDbType.UniqueIdentifier;
                    ProductParam.Value = product.Id;
                    ProductParam.ParameterName = "@ProductId";

                    SqlParameter MaterialParam = new SqlParameter();
                    MaterialParam.SqlDbType = SqlDbType.UniqueIdentifier;
                    MaterialParam.Value = material.id;
                    MaterialParam.ParameterName = "@MaterialId";

                    SqlParameter ColorParam = new SqlParameter();
                    ColorParam.SqlDbType = SqlDbType.VarChar;
                    ColorParam.Value = Color;
                    ColorParam.ParameterName = "@Color";

                    SqlParameter PriceParam = new SqlParameter();
                    PriceParam.SqlDbType = SqlDbType.Float;
                    PriceParam.Value = Price;
                    PriceParam.ParameterName = "@Price";

                    SqlParameter kountParam = new SqlParameter();
                    kountParam.SqlDbType = SqlDbType.Int;
                    kountParam.Value = kount;
                    kountParam.ParameterName = "@kount";

                    command.Parameters.Add(IdParam);
                    command.Parameters.Add(StartDateParam);
                    command.Parameters.Add(EndDateParam);
                    command.Parameters.Add(ProductParam);
                    command.Parameters.Add(MaterialParam);
                    command.Parameters.Add(ColorParam);
                    command.Parameters.Add(PriceParam);
                    command.Parameters.Add(kountParam);

                    command.ExecuteNonQuery();

                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }


    }
}
```