Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка інформаційної системи для автоматизованого аналізу рівня продаж інтернет-магазину побутової техніки.

Виконав: студент IV курсу, групи СН-43
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

(підпис) Шпота Т.О. (прізвище та ініціали)

Керівник (підпис) Палка О.В. (прізвище та ініціали)

Нормоконтроль (підпис) Липак Г.І (прізвище та ініціали)

Завідувач кафедри (підпис) Боднарчук І.О. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Шпоті Тарасу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка інформаційної системи для автоматизованого аналізу рівня продаж інтернет-магазину побутової техніки.

Керівник роботи Палка Олег Вікторович, доктор філософії, асистент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*07*» *травня* 2025 року № *4/7-444*

2. Термін подання студентом завершеної роботи 23 червня 2025 р.

3. Вихідні дані до роботи Літературні та інтернет джерела

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1.Аналіз предметної області. 1.1 Ключові аспекти аналізу даних. 1.2 Основні метрики та KPI онлайн-продажів..1.3 Архітектура зберігання та аналітики даних. 1.4 Архітектури зберігання та аналітики даних. 1.5 Висновки до 1-го розділу. 2 Проектування програми для автоматизованого аналізу рівня продаж. 2.1 Вимоги до проекту. 2.2 Вибір інструментів розробки. 2.3 Визначення користувачів та способів використання. 2.4 Створення та налаштування бази даних. 2.4.1 Проектування БД. 2.4.2 Варіанти створення таблиць. 2.4.3 Створення та налаштування бази даних. 2.5 Авторизація та безпека. 2.6 Висновок до 2-го розділу. 3. Розробка програми для автоматизованого аналізу рівня продаж. 3.1 Послідовність дій та логіка реалізації. 3.2 Ключові кодові рішення. 3.2.1 Моделі EF Core (C#). 3.2.2 Контекст бази даних. 3.2.3 Репозиторій та unit of Work. 3.2.4 Сервіс аналітики продажів. 3.2.5 REST API (ASP.NET Core). 3.2.6 Docker-оркестрація. 3.2.7 Приклад юніт-тесту (xUnit). 3.3 Інтерфейс користувача. 4 Безпека життєдіяльності, основи охорони праці. 4.1 Менеджмент безпеки в умовах воєнного стану. 4.2 Заходи, для покращення умови праці розробника. 4.3 Висновок до четвертого розділу. Висновок. Перелік джерел. Додатки

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Сенчишин В.С., кандидат технічних наук, доцент кафедри МТ		

7. Дата видачі завдання 27 січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	30.01.2025	*Виконано*
2.	Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи	01.02.2025-15.02.2025	*Виконано*
3.	Виконання дослідження щодо аналізу рівня продаж	16.02.2025-30.02.2025	*Виконано*
	Розроблення системи автоматизованого аналізу рівня продаж інтернет магазину побутової техніки.		
4.	Оформлення розділу «Назва першого розділу	01.03.2025-15.03.2025	*Виконано*
	»		
5.	Оформлення розділу «Назва другого розділу	16.03.2025-30.03.2025	*Виконано*
	»		
6.	Оформлення розділу «Назва третього розділу	01.04.2025-15.04.2025	*Виконано*
	»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	16.04.2025-30.04.2025	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони	01.05.2025-15.05.2025	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	15.05.2025-30.05.2025	*Виконано*
10.	Нормоконтроль		*Виконано*
11.	Перевірка на плагіат		*Виконано*
12.	Попередній захист кваліфікаційної роботи		*Виконано*
13.	Захист кваліфікаційної роботи		

Студент _____ Шпота Т.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Палка О.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка інформаційної системи для автоматизованого аналізу рівня продаж інтернет-магазину побутової техніки// Кваліфікаційна робота освітнього рівня «Бакалавр» // Шпота Тарас Олександрович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-43 // Тернопіль, 2025 // С. 60, рис. – 24, табл. – 5.

Ключові слова: аналіз даних, система, інтернет-магазин, C#, SQL, таблиця, автоматизація.

Кваліфікаційна робота присвячена дослідженню розробки інформаційної системи для автоматизованого аналізу рівня продаж інтернет-магазину побутової техніки.

В першому розділі кваліфікаційної роботи описано ключові етапи створення система аналізу даних, та процесів її автоматизації. Також приділена увага інструментам, які використовують в процесі розробки система автоматизованого аналізу даних.

В другому розділі кваліфікаційної роботи подано процес підготовки до написання коду, а саме: постановка вимог, створення та налаштування бази даних, визначення основних користувачів, їх можливостей, та аналіз процесів, що відбуватимуться в системі.

В третьому розділі кваліфікаційної роботи описано процес розробки проекту, створення об'єктів, контролерів та сервісів, що будуть відповідати за коректну роботу програми.

ANNOTATION

Development of an Information System for Automated Sales Analysis of a Household Appliance Online Store// Qualification work for educational level “Bachelor” // Shpota Taras Oleksandrovych // Ternopil National Technical University named after Ivan Pului, Faculty of Computer and Information Systems and Software Engineering, Department of Computer Science, group SN-43 // Ternopil, 2025 // P. 60, Fig. - 24, Tables - 5.

Keywords: data analysis, system, online store, C#, SQL, table, automation.

The qualification work is devoted to the study of the development of an information system for automated analysis of the level of sales of an online store of household appliances.

The first chapter of the qualification work describes the key stages of creating a data analysis system and its automation processes. Attention is also paid to the tools that uses in the process of developing an automated data analysis system.

The second section of the qualification work describes the process of preparation for writing code, namely: setting requirements, creating and setting up a database, identifying the main users, their capabilities, and analysis of the processes that will take place in the system.

The third section of the qualification work describes the process of project development, creation of objects, controllers and services that will be responsible for the correct operation of the program.

ЗМІСТ

ВСТУП

Актуальність автоматизованого аналізу. Внаслідок глобалізації постійно збільшується потік даних, який люди опрацьовують кожного дня, починаючи від статей в інтернеті, закінчуючи списком товарів в інтернет магазині. Відповідно виникає потреба в аналізі цих даних, для виділення найбільш важливих, та оптимізації часто використовуваних процесів.

Сучасний ринок електронної комерції вимагає використання передових аналітичних інструментів для ефективного прийняття рішень та оптимізації стратегій продажів [1].

Що стосується сфери продажів, аналіз рівня продажу дозволить зробити сам процес продажу більш зрозумілим як для клієнта інтернет-магазину, так і для його працівника. Клієнту будуть відображатись найбільш популярні товари в певних категоріях, що дозволить швидше зробити вибір стосовно необхідної категорії. Що стосується продавця або магазину то виникає розуміння на яку категорію товарів слід зробити акцент, або який товар порадити клієнту, спираючись на досвід попередніх клієнтів. Тому аналіз рівня продажів є актуальним напрямком сучасних досліджень в галузі продажів.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Бакалавр» є опрацювати основні принципи автоматизації аналізу даних, використовуючи такі технології як C#, Entity Framework та SQL. Для досягнення поставленої мети потрібно виконати ряд завдань, зокрема:

- Дослідити процеси аналізу даних.
- Проаналізувати способи автоматизації.
- Ознайомитись з інструментами автоматизації.
- Розробити список технічних та не технічних вимог.
- Розробити ТЗ, включаючи можливості кожного користувача.
- Створити проект, який буде виконувати поставлені задачі.
- Протестувати готовий проект.

Практичне значення одержаних результатів. Як результат ми отримаємо готовий застосунок, який дозволить проаналізувати систему продажів, виявити слабкі та сильні стороно як окремого продавця так і магазину в цілому, та оптимізувати внутрішні процеси.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Ключові аспекти аналізу даних

В наш час абсолютна більшість процесів базуються на обробці та аналізі інформації. Відповідно до збільшення кількості таких процесів і запитів збільшується кількість інформації яку необхідно збирати та обробляти.

Як правило інформація поступає в сирому вигляді, тобто в ній є певна кількість аномалій і шумів, які необхідно відфільтрувати. Після чого проходить низка процесів аналізу, структурування та візуалізації інформації. В результаті ми отримаємо чітку картину яка нам показує стан системи, яку ми аналізуємо.

Весь цей процес і є аналізом інформації. Його можна чітко структурувати, і автоматизувати, на чому і базуються сучасні системи обробки і аналізу інформації.

Системний підхід широко використовує у всіх сферах науки і практики [2]. Загальна етапи роботи системи аналізу даних виглядає наступним чином:

- збір даних;
- очистка від шумів та аномалій;
- виявлення закономірностей;
- створення моделей прогнозування;
- візуалізація даних;
- автоматизація постійних процесів;
- збільшення ефективності автоматизованих процесів.

Кожен з етапів є важливим, і без нього система буде не ефективною, або взагалі не зможе існувати чи працювати.

Важливим також є інтеграція системи автоматизації збору та аналізу даних. Цей крок дозволить значною частиною прискорити та здешевити подальшу роботу системи, а правильно налаштована система дозволить

запобігти великій кількості проблем, які можуть виникнути в наслідок людського фактору.

Першим і ключовим етапом є збір даних. Його суть полягає в автоматичному зборі даних з різноманітних джерел і завантаження їх в єдину базу даних. Залежно від даних, методи і джерела можуть різнитись: датчики, системи відслідковування активності, веб сайти, стороні бази даних. Метод збору даних обирається відповідно до мети дослідження та його придатності для конкретного типу дослідження, яке планується провести[3]. Не виключено що в процесі збору дані буде потрібно видозмінити до певної сталої форми, для цього використовують ETL (англ. Extract, Transform, Load) процеси. Відповідно до назви суть даних процесів полягає в отриманні даних з різноманітних джерел, трансформація до необхідної форми, та завантаження в центральну базу даних. Еволюція конвеєрів ETL в епоху озер даних вимагає автоматичної перевірки якості на кожному кроці перетворення [4].

Другим етапом ми очищаємо дані. Суттю очищення даних є їх аналіз для виявлення помилок і невідповідностей, які виникли в базі даних [5]. Завдяки цьому ми збільшуємо подальшу ефективність роботи з даними, зменшуючи їх кількість, шляхом видалення дублікатів, очищення від аномалій і шумів. Таким чином видаляється інформація яка не несе цінності, або не є актуальною. Як результат в подальшому скорочується час на обробку інформації, зменшується об’єм інформації, яку дані займають і в загальному покращується якість даних

Третій етап це аналіз даних. За допомогою таких методів як машинне навчання, статистичний аналіз, кластеризація та багатьох інших ми шукаємо приховані закономірності. Це допомагає отримати додаткову інформацію по отриманих раніше даних, яка поверхнево може показати що відбувається в бізнес процесах, але потребує подальшого аналізу щоб отримати більш точну картину.

Після цього ми можемо продовжити аналізувати дані і на їх основі прогнозувати подальші події, такі як збільшення або зменшення рівня продажів, прогноз погоди, поведінку певних груп та інші події. Для цього

використовуються такі методи як: регресивний аналіз, часові ряди, нейронні мережі та інші методи та моделі. На цьому етапі з'являється можливість сформувати декілька варіантів розвитку подій і відповідно дій при даних сценаріях.

П'ятим етапом є візуалізація даних. На цьому етапі оброблені дані та інформація отримана при їх обробці візуалізується за допомогою таблиць та діаграм. Це дозволяє зробити її легшою для сприйняття, і відповідно полегшити її подачу в звітах. Відповідно для цього також існує велика кількість інструментів, в тому числі інтегрованих в редактори текстів, як яскравий приклад «Діаграми» в програмі Microsoft Word.

Решта етапів спрямовані в основному на підвищення ефективності автоматизованих процесів, та зменшення кількості помилок. Тобто самі процеси мають проходити швидше, при цьому споживаючи менше ресурсів, та видаючи меншу кількість помилок. Особливо зручно якщо така система може працювати в умовах реального часу, тобто оновлювати інформацію в процесі отримання початкових даних. Це дозволяє в будь який момент отримати актуальну інформацію про поточний стан процесів.

Також важливою є можливість масштабувати систему, таким чином, щоб вона могла працювати як з досить малими об'ємами даних, так і з великою їх кількістю. З подальшою можливістю додавати новий інструментарій, який буде ефективно працювати не залежно від розмірів системи.

Також логічним є факт того що при існуванні такої кількості важливої інформації виникає потреба в їх захисті. У всьому світі виникла низка інцидентів, пов’язаних із безпекою даних, зокрема пряма крадіжка даних для перепродажу, використання даних для створення точних шахрайських дій і навіть шифрування даних користувачів і вимагання викупу[6]. Щоб уникнути таких інцидентів системи зберігання даних намагаються збільшити рівень свого захисту від зловмисників.

Найбільш популярними зараз є декілька способів зберігання даних, які мають ряд переваг та недоліків.

Локальні ‘сховища знаходяться локально, тобто на самому місці використання даних. Це дозволяє мати до них прямий доступ, що забезпечує хорошу швидкість і безпеку, проте в такому випадку не уникнути високих витрат на створення локального сховища та його обслуговування. Також недоліком є важкість дублювання і актуалізації інформації на віддалені сховища.

Віддалені сховища від сторонніх компаній які надають такі послуги дозволяють багатократно скоротити витрати на апаратну частину, проте виникає проблема в постійному доступі до інтернету або іншому джерелу передачі, що не гарантує настільки високий рівень безпеки. Також нікуди не зникає проблема дублювання інформації.

Хмарні сховища вирішують проблему дублювання інформації. Вони дозволяють зберігати, керувати та отримувати доступ до файлів за допомогою відповідного ПЗ та інтернет-з’єднання [7]. Також за рахунок розділення інформації на різні кластери хмари безпека в цілому покращується.

1.2 Ключові метрики та KPI онлайн-продажів

Виділяють п’ять стратегічних показників ефективності: GMV, AOV, CR, CAC і CLV.

- GMV (Gross Merchandise Value) – валовий обсяг продажів за певний період.
- AOV (Average Order Value) – середній чек
- CR (Conversion Rate) – частка відвідувачів, що здійснили покупку.
- CAC (Customer Acquisition Cost) – середні витрати на залучення нового клієнта.
- CLV (Customer Lifetime Value) – інтегральна цінність клієнта за весь життєвий цикл.

Також є додаткові метрики, які додатково дозволяють проаналізувати ефективність.

- RPR – відсоток клієнтів що повернулись.
- Return Rate – відсоток продажів що повернули.
- Attach Rate – відсоток апсейлу в кожній продажі.

Вони є ядром аналітики, однак для високої точності планування та оперативного управління необхідно деталізувати їхню структуру й зв'язки (див. таблиця 1.1). Нижче наведено розгорнуті підпункти, які описують алгоритм розрахунку, частоту оновлення, типові галузеві бенчмарки та практичні дії, що допомагають оптимізувати кожен індикатор.

Таблиця 1.1 – Метрики онлайн-продажів

Метрика	**Формула**	**Частота оновлення**	**Галузевий бенчмарк**	**Типова реакція**
1	**2**	**3**	**4**	**5**
GMV	Σ(quantity × price)	Щодня / Щомісяця	2–4 млн ₴ / міс (UA, дрібна техніка)	Сповіщення, коли GMV ↓ > 10 % m/m
AOV	GMV / #замовлень	Щодня	120–180 USD	Бандлінг аксесуарів, безкоштовна доставка
CR	(замовлення / сеанси) × 100 %	Щодня	1,6–2,3 % (UA)	A/B-тест checkout, оптимізація швидкості
CAC	Marketing Spend / #нових клієнтів	Щотижня	CAC/AOV ≤ 0,3	Перерозподіл бюджету, ROAS-звіт

Продовження таблиці 1.1

1	2	3	4	5
CLV	(AOV × PF × AL × маржа) – CAC	Квартал	≈ 4,5 × AOV	Програма лояльності, утримання клієнтів
RPR	repeat customers / all customers	Щомісяця	≥ 25 %	E-mail-кампанії для повторних покупок
Return Rate	returns / orders	Щомісяця	≤ 8 %	Перевірка описів, фотоконтенту
Attach Rate	accessory sales / core SKU	Щодня	≥ 15 %	Upsell-віджети на сторінках товару

1.3 Архітектури зберігання та аналітики даних

Комплексна аналітична платформа для e-commerce-ритейлу зазвичай поєднує кілька шарів і концепцій: класичний Data Warehouse (DW) для «щільної» звітності, Data Lake/Lakehouse (це рівень зберігання з відкритим вихідним кодом, який забезпечує надійність, узгодженість і масштабованість озер даних [8]) – для напівструктурованих логів і ML-завдань, а також in-memory кеш (уповільнення < 15 с) для дашбордів у реальному часі. Нижче деталізовано ключові підходи (див. таблиця 1.2).

Таблиця 1.2 – Ключові підходи архітектури зберігання даних

Критерій	**Star Schema**	**Snowflake Schema**	**Коли обрати**
Нормалізація	Денормалізована	Нормалізована	Обрати Star, якщо пріоритет — швидкі агрегати; Snowflake — при великій кількості пов'язаних довідників
JOIN-и	Мінімальні (fact → dimensions)	Багаторівневі	Star для BI-дашбордів; Snowflake — економія диску, коли S3-витрати критичні
Швидкість OLAP	Вища (менше JOIN)	Нижча, але краще компресія	Star, якщо SLA < 2 с; Snowflake, коли місце > швидкість
Гнучкість змін	Складніше змінювати виміри	Просте додавання вимірів	Snowflake при швидкій еволюції бізнес-категорій

Lakehouse (Databricks Delta, Apache Iceberg) об'єднує переваги озера (гнучке зберігання файлів Parquet/ORC) з транзакційними гарантіями табличного шару DW. Сценарії:

- зберігання сирих click-stream логів (≈ 5 GB/день);
- тренування моделей рекомендацій (ALS/LightFM) прямо на Delta-таблицях;
- блакитно-зелений деплой аналітичної схеми через ACID-комміти.

Таблиця 1.3 -порівняння Lamda і Kappa методів

Шар	**Lambda**	**Kappa**
Batch-DW	Так (щодобове ETL)	Ні
Streaming ETL	Так (Kafka → Flink)	Так (єдиний шлях)
Спрощення стека	Складніша підтримка	Менше компонентів
Затримка	5-15 хв	1-5 с

Для KPI реального часу (ON-SITE stock-alert) достатньо Kappa-моделі (Kafka + ksqldb) і оновлення матеріалізованих представлень PostgreSQL (хвостова репліка).

- ClickHouse / BigQuery / Redshift Spectrum — сканують терабайти за секунди; партиціювання + сортування за (date, category) дає ×8 прискорення.
- Materialized Views у PostgreSQL 15: денні агрегати продажів заощаджують до 70 % CPU при дашборді.
- Z-ORDER у Delta/Iceberg — кластеризація по SKU + date для зондування «гарячих» SKU.

Порівняння цих платформ наведено нижче в таблиці 1.4.

Таблиця 1.4 – Порівняння різних платформ

Платформа	**Плюси**	**Мінуси**
BigQuery	Serverless, automatic tuning, SQL-ML	Data-egress $0.12/GB, менша гнучкість індексів
AWS Redshift	Spectrum + S3 lake, AQUA-акселератор	Потрібно керувати cluster-scale
Snowflake	Розділення compute-/storage-кластерів, zero-copy clone	Ліцензія «crédit» > $2/TB scan
On-prem ClickHouse	Повний контроль, дешевий HDD RAID10	DevOps-навантаження, SLA здебільшого на команді

- Партиція по date_trunc('month') + sub-partition category; гибридне TTL → авто-архів на S3 Cold.
- BRIN-індекси для широких лог-таблиць (> 10 мил рядків/день).
- Shard-кеї в Citus (PostgreSQL MPP) — hash(order_id) дає лінійний приріст throughput.

Таблиця 1.5 – Реалізації аспектів

Аспект	**Реалізація**
RBAC	Row-Level Security (PostgreSQL 16) — ограничує менеджерів лише своїми магазинами
Шифрування at-rest	Transparent Data Encryption (Postgres pgcrypto, AWS KMS)
GDPR/PCI Compliance	Pseudonymization (`customer_hash`), лог аудиту (immutable append-only)
Backups	Point-in-time recovery (WALG → S3, retention 30 днів)

1.4 Інструменти для розробки проекту

Аналіз даних часто передбачає безліч різнорідних кроків, від застосування різних інструментів командного рядка до використання скриптових мов, таких як R або Python, для створення графіків і таблиць[10] Правильний вибір цих інструментів дозволяє спростити процес розробки, та зробити роботу програми більш ефективною та швидкою.

База даних – це поіменована, структурована сукупність взаємопов'язаних даних, що належать до певної предметної області і зберігається на комп’ютерних носіях[11].

БД активно використовуються при розробці сайтів, в яких потрібно виводити інформацію в режимі актуального часу. Як приклад онлайн магазин, в якому потрібно показати доступні товари з актуальними цінами. Також в них зберігається інформація з різних операцій, будь то фінансові операції, або записи дій з певними об’єктами.

Всі дані в БД повинні бути зв’язаними. Найчастіше це набір інформації про певний об’єкт або групу об’єктів.

Також не мало важливою є можливість групувати та зв’язувати між собою дані. Групування відбувається в межах однієї таблиці, де зберігаються однотипні дані, наприклад дані клієнтів, або список і кількість товарів. Між таблицями також існують зв’язки, які дозволяють робити структуру більш гнучкою та легкою для розуміння, і легко вносити зміни, коли оновлення однієї частини даних, автоматично оновляє зв’язану з нею частину.

Однією з ключових переваг є можливість швидко вносити дані та мати до них моментальний доступ, а також швидкий пошук, як в межах однієї таблиці, так і по декількох одразу. Як приклад ми можемо вивести список всіх покупок за певний період, і відсортувати їх по ціні. Такий запит буде звертатись до декількох таблиць, та групувати дані, що відносяться до одного замовлення.

Також в базі даних існує певний взаємозв'язок інформації: зміна в одному рядку може спричинити зміни в інших рядках – це допомагає працювати з інформацією простіше і швидше.

Бази даних дають змогу зберігати інформацію, що виглядає як зв'язані між собою таблиці. Саме в БД зберігаються вся необхідна та корисна інформація для функціонування сайту (клієнтські дані, прайс-лист, список товарів).

Щоб створити запит до бази даних часто використовують. SQL (Structured Query Language), що дає змогу додавати, редагувати та видаляти інформацію, що міститься у таблицях.

Під час розробки програмних продуктів, таких як сайти, програми, або системи аналізу даних використовують різноманітні системи керування базами даних (СКБД) – це програмна система, яка використовується для створення, використання та підтримки база даних. До основних СКБД, відносять:

- Oracle Database.
- PostgreSQL.
- Microsoft SQL Server.
- MySQL.

Такі системи управління відрізняються централізованою обробкою запитів, забезпечують надійність, доступність та безпеку БД.

Однією з найбільш популярних систем керування є MySQL, вона дає зручний доступ для управління БД та підтримує велику кількість таблиць різних типів.

Також варто розуміти що різні СКБД мають різні моделі типізації, наприклад:

- Реліційні – тип СКБД в яких інформація зберігається у вигляді таблиць, де кожен стовпець має свій тип даних і призначається для конкретного типу інформації по типу тексту, чисел чи булевих значень. Вони всі базуються на SQL (Structured Query Language). До такого типу відносяться такі СКБД як:

PostgreSQL, MySQL, Microsoft SQL Server, Oracle Database, та інші, менш популярні.

- Документо-орієнтовані (No-SQL) – група СКБД які зберігають документи у вигляді документів, в форматі .json або .bson. Вони мають більш гнучку структуру, від чого менш передбачувані ніж реляційні, проте в певних напрямках розробки це і є їх перевагою. До них можна віднести MongoDB, CouchDB, Firebase Firestore.
- Ключ-значення – призначені для компактного зберігання даних “парами”. Найбільш популярні вони в таких напрямках як кешування та управління чергами повідомлень. Її представниками є Redis, Amazon DynamoDB, Riak.

Відповідно кожен тип СКБД має свій ряд задач для якого найкраще підходить, і оскільки в наш час найбільш затребуваною є розробка програмного забезпечення і сайтів, то найчастіше використовують реляційні БД. Це явище добре помітно в переліку найбільш популярних СКБД (див. рисунок 1.1).

Rank			DBMS	Database Model	Score		
Mar 2025	Feb 2025	Mar 2024			Mar 2025	Feb 2025	Mar 2024
1.	1.	1.	Oracle	Relational, Multi-model	1253.08	-1.74	+32.02
2.	2.	2.	MySQL	Relational, Multi-model	988.13	-11.86	-113.37
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	788.14	+1.27	-57.67
4.	4.	4.	PostgreSQL	Relational, Multi-model	663.42	+3.81	+28.52
5.	5.	5.	MongoDB	Document, Multi-model	396.42	-0.21	-28.11
6.	↑ 7.	↑ 9.	Snowflake	Relational	161.78	+6.20	+36.40
7.	↓ 6.	↓ 6.	Redis	Key-value, Multi-model	155.36	-2.55	-1.64
8.	8.	↓ 7.	Elasticsearch	Multi-model	131.38	-3.25	-3.41
9.	9.	↓ 8.	IBM Db2	Relational, Multi-model	126.57	+1.14	-1.18
10.	10.	10.	SQLite	Relational	113.08	-0.74	-5.08

Рисунок 1.1 – Найбільш популярні СКБД

Наступним інструментом є мова програмування. Це основний інструмент, що дозволяє повністю створити та налаштувати програму, її реакцію на різні дії користувача, та зв'язок між рештою складових проекту.

Кожна мова програмування має свій ряд завдань, які вони виконують і відповідно під які пишуться більшість бібліотек під них. Багато сучасних мов програмування підтримують парадигму об'єктноорієнтованого програмування (ООП) або принаймні мають можливості для роботи з об'єктами[11], що дозволяє використовувати їх в конкретних напрямках розробки. Інші ж навпаки слабко прив’язані до цієї парадигми і спрямовані на інші завдання. Тому виникла умовна система типізації, за якою для кожного завдання можна виділити декілька мов які дозволяють ефективно виконати поставлення завдання. Найбільша популярними мовами програмування є наступні:

- Python.
- JavaScript.
- Java.
- C#.
- C++.
- PHP.

C++ та Java найчастіше використовуються для розробки додатків, ігор та драйверів, де потрібен контроль даних на низькому рівні.

Python, C# та PHP навпаки частіше використовуються для розробки бекенду, де не потрібно так детально контролювати пам'ять, або аналізу даних, де важлива точність і швидкість виконання.

IDE (integrated development environment) – це застосунки, що використовуються для написання та редагування коду, а також його оформлення та структуризації.

Як привило кожна IDE підтримує одну або кілька мов програмування, і велику кількість інструментів для цих мов.

Найбільш популярними на даний момент є декілька варіантів, таких як:

- Visual Studio.
- Visual Studio Code.
- Sublime Code.

- Notepad++.

Visual Studio (див. рисунок 1.2) і Visual Studio Code – дві IDE від компанії Microsoft. Є досить популярними за рахунок великої кількості документації, яку можна знайти на офіційному сайті. В даний момент активно розвиваються і отримують оновлення. Немало важливим є факт того що ці середовища мають обширний інструментарій, починаючи від підсвічування синтаксису і автозаповнення, закінчуючи великою кількістю додаткових плагінів і бібліотек доступних для встановлення на проект.

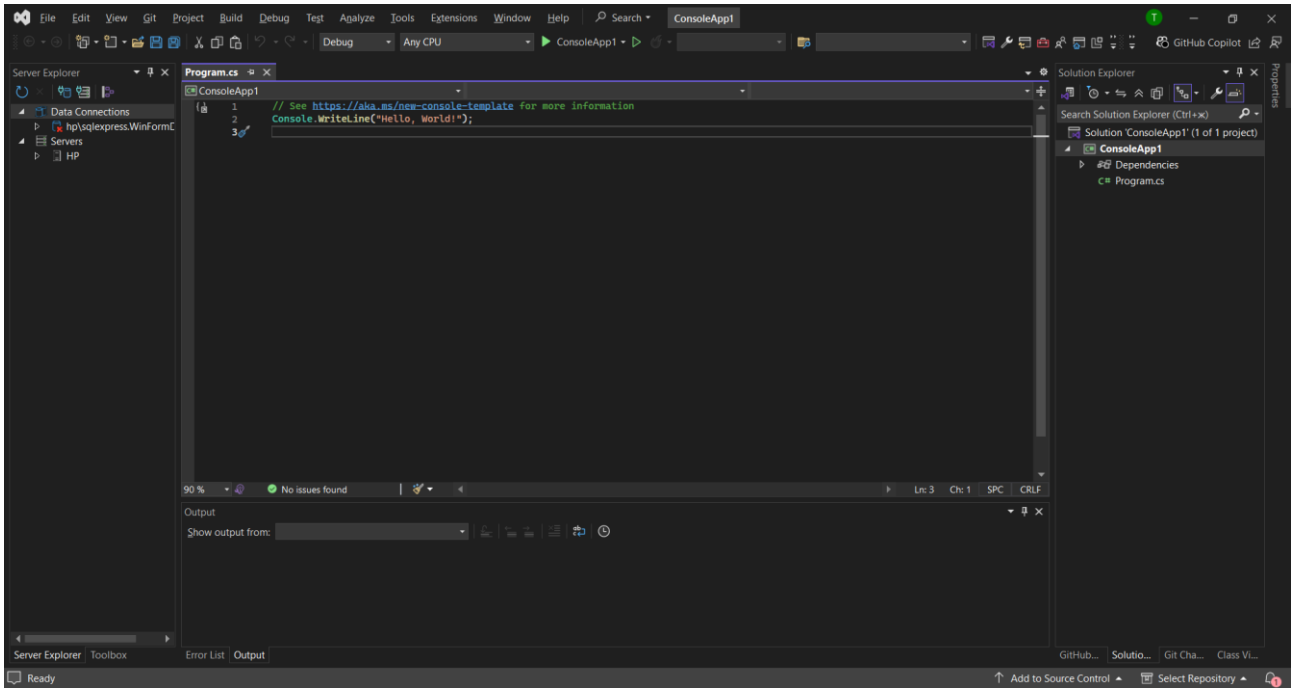

Рисунок 1.2 – Інтерфейс Visual Studio

Також ці IDE дозволяють писати код на великій кількості мов, таких як JavaScript, TypeScript, Python, C++, Java, PHP, Go, Rust та багато інших.

Ключова різниця між Visual Studio і Visual Studio Code полягає в інструментарії і його призначенні. Visual Studio - це повноцінне IDE з більш обширним функціоналом і призначена спеціально для повноціної розробки масштабних проектів. Основними мовами для цієї IDE є C# та F#, також

розроблені компанією Microsoft. Проте з недоліків для швидкої роботи ця програма потребує потужного апаратного забезпечення.

Visual Studio Code (див. рисунок 1.3) навпаки ж більше схожа на редактор тексту ніж на повноцінну IDE, хоча підтримує більшу кількість мов програмування. Вона має зручніший інтерфейс і використовується здебільшого для невеликих проектів, в тому числі для розробки FrontEnd частини.

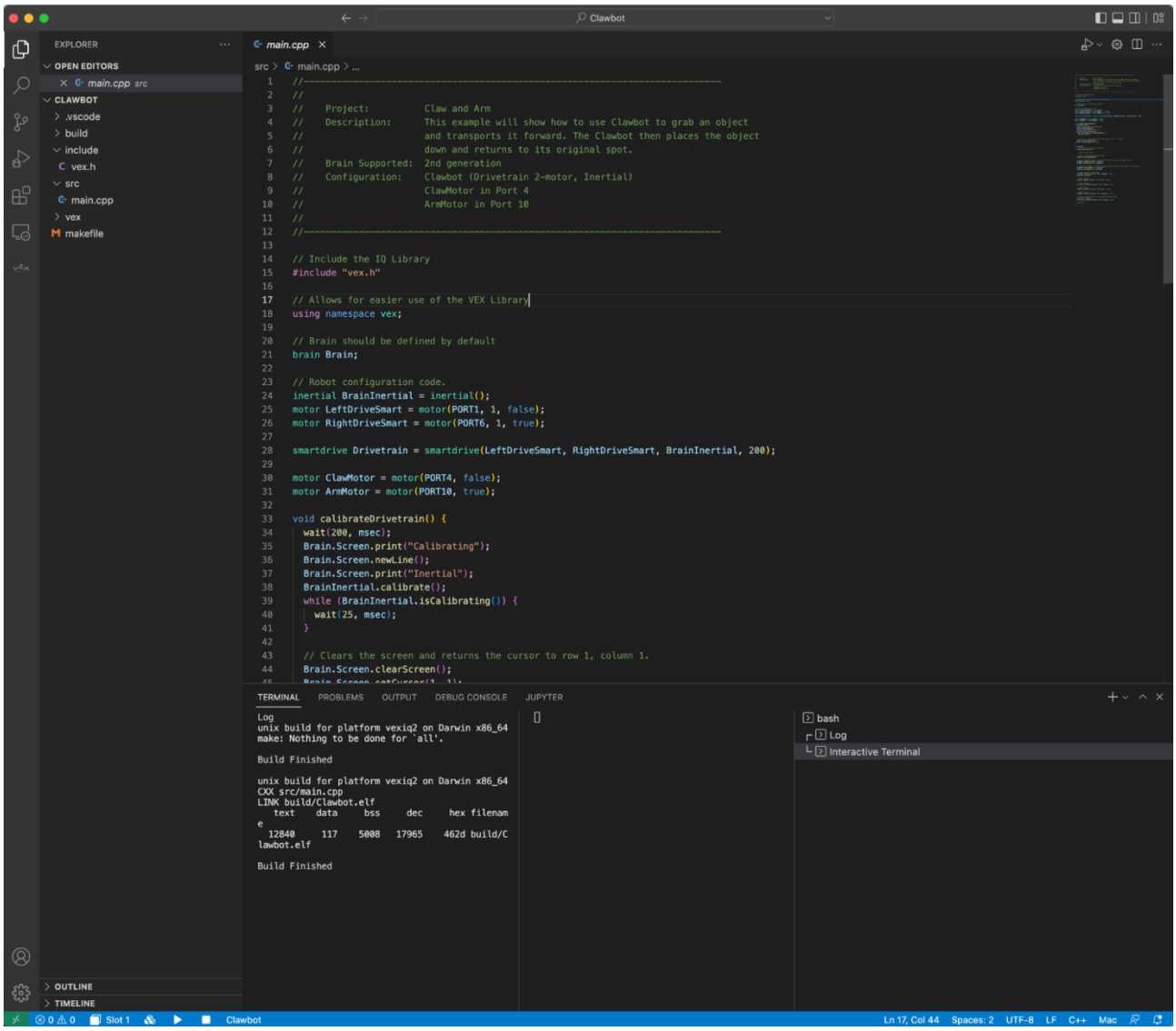

Рисунок 1.3 – Інтерфейс Visual Studio Code

Sublime Text (див. рисунок 1.4) – це високошвидкісний текстовий редактор з відкритим кодом. В основному його використовують для розробки веб

застосунків. Вагомими його перевагами є швидкість, зручний інтерфейс та велика кількість інструментів, що дозволяють пришвидшити роботу з кодом.

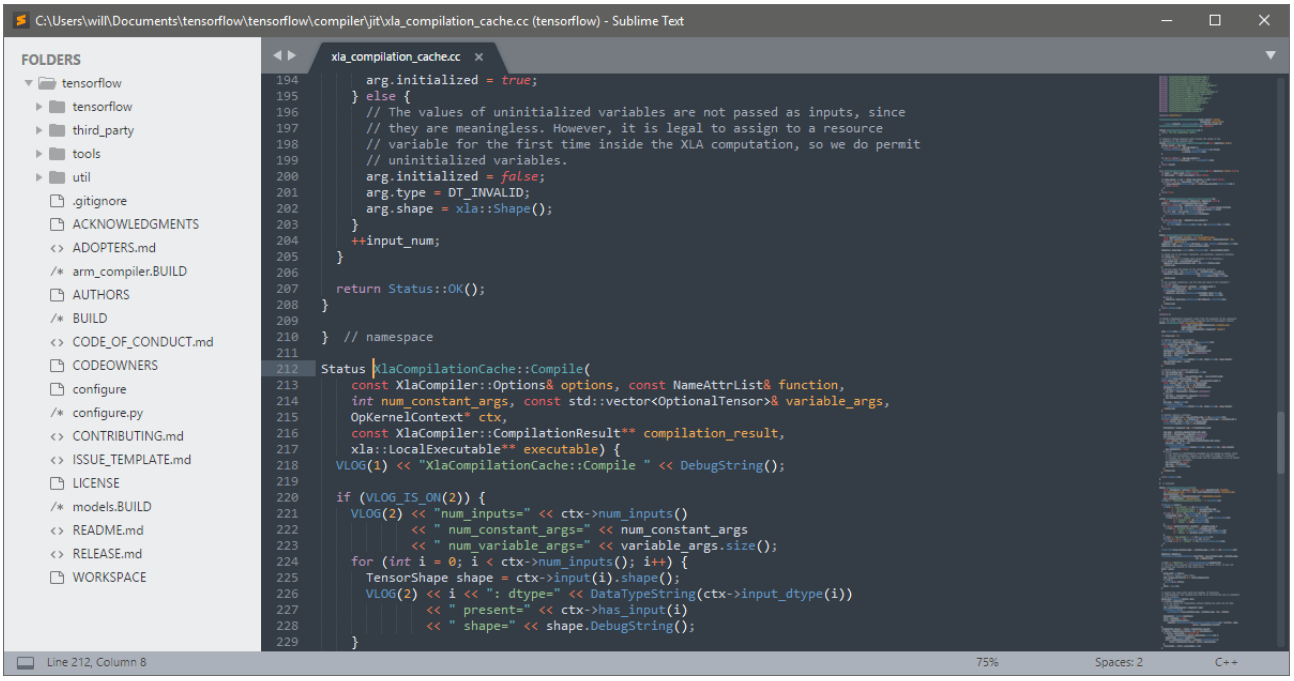

Рисунок 1.4 – Інтерфейс Sublime Text

Notepad++ (див. рисунок 1.5) більш простий варіант, створений для непотужних систем. Свою популярність набув саме завдяки цьому, що дозволяло встановлювати його на багатьох пристроях, особливо в сфері навчання. Не дивлячись на невимогливість до апаратного забезпечення має достатній функціонал для розробки невеликих проектів.

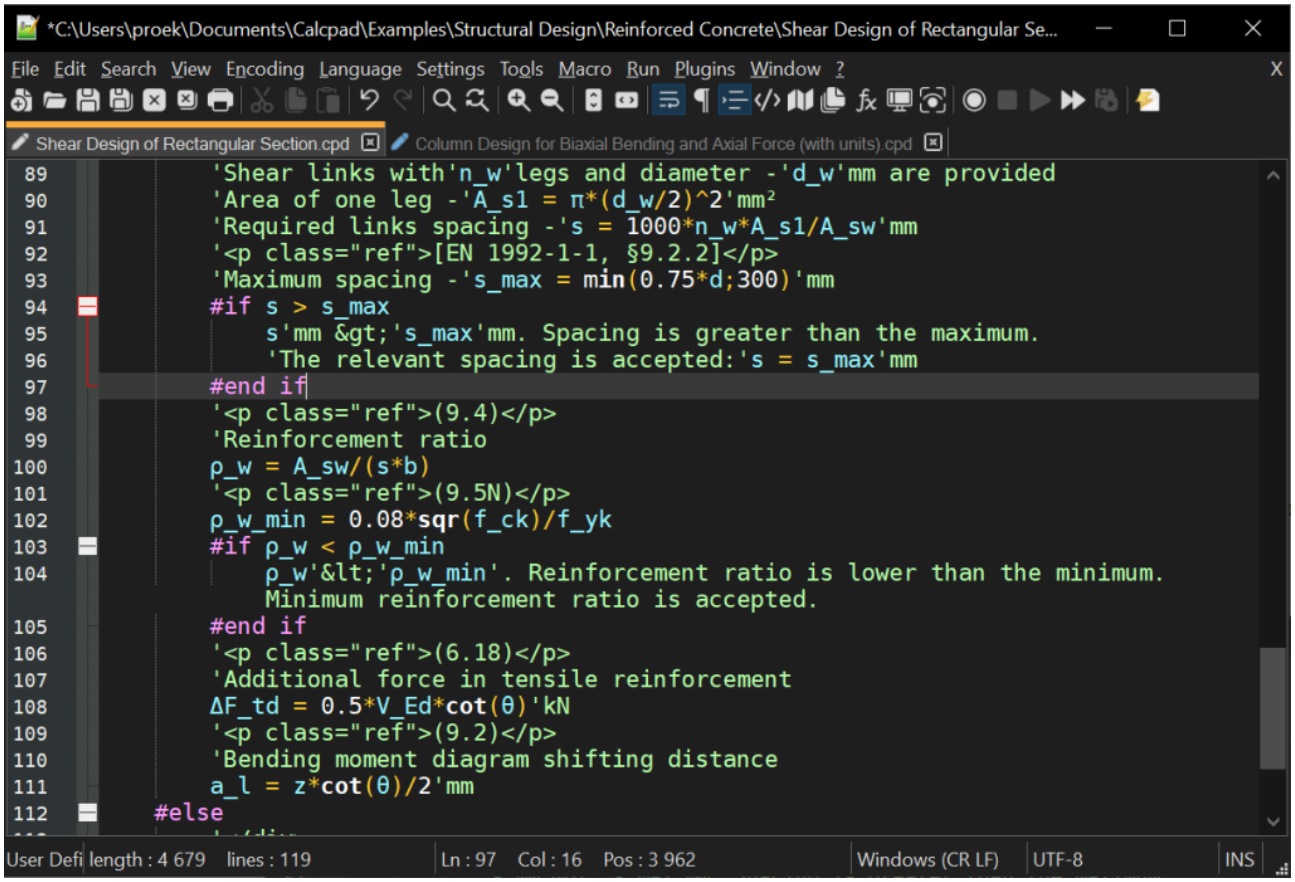

Рисунок 1.5 – Інтерфейс Notepad++

Відповідно при процесі вибору IDE потрібно відштовхуватись від обраної мови програмування та кінцевої цілі проекту, для вибору IDE, які дозволить найбільш ефективно виконати всі поставлені завдання

1.5 Висновки до 1-го розділу

У першому розділі кваліфікаційної роботи проведено аналіз предметної області – системи аналізу даних. Розглянуто процеси та етапи в цій сфері, та їх важливість. Проведено аналіз кожного етапу, його ціль та розглянуто способи виконання. Також проведено аналіз процесу автоматизації даного процесу та інструментів, за допомогою яких можливо це виконати. Окремо виділено питання безпеки даних.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ПРОГРАМИ ДЛЯ АВТОМАТИЗОВАНОГО АНАЛІЗУ РІВНЯ ПРОДАЖ

2.1 Вимоги до проекту

Значна кількість помилок вноситься у програмне забезпечення на етапі формування та формулювання вимог [12]. Тому правильна постановка вимог є одним з ключових етапів розробки проекту, що дозволить полегшити подальше проектування і розробку, та зменшити кількість помилок. Таким чином ми структуруємо що і як повинен робити проект, яким чином він буде працювати і як виглядатиме.

Крім того при розробці інформаційної системи для аналізу продажів ключовим є детальне вивчення потреб кінцевих користувачів [13], оскільки програма розробляється конкретно для них, і розуміння їх потреб дозволить розробити максимально ефективний інструмент роботи.

Вимоги можна поділити на два типи: технічні та нетехнічні. До технічних можна віднести такі вимоги як функціонал, швидкість роботи то що.

В даному проекті виставлено наступні технічні вимоги:

- Легкість освоєння та розширення коду.
- Швидка робота.
- Функціонал для створення продажів.
- Функціонал для перегляду статистики по вже здійснених продажах.
- Функціонал для перегляду прогнозів на певний період майбутнього.
- Можливість запуску на різних платформах.

Іншим видом вимог є нетехнічні. Вони більше стосуються не самого процесу створення, а досвіду використання.

В даному проекті виставлено наступні нетехнічні вимоги:

- Зручний інтерфейс.
- Низький поріг входу для користування.

- Можливість запуску на різних платформах.

Після постановки всіх цих вимог процес проектування коду стає більш комфортним, оскільки тепер в нас є чіткі критерії за якими ми можемо оцінити чи досягнуті поставлені нами задачі. Також цей етап дозволить уникнути великої кількості помилок, а в комерційних проектах пошук і виправлення дефектів може становити 50-75% від загальної вартості розробки, якщо проблеми виникають у вимогах. [14]

2.2 Вибір інструментів розробки

Як було описано в попередньому розділі існує велика кількість мов програмування які спеціалізуються на обробці та аналізі даних.

Для розробки даного проекту було обрано мову програмування C#. Основними критеріями для вибору були наступні:

- Сучасна мова програмування, яка підтримується багатьма платформами та має активну спільноту.
- Доступність бібліотек, які дозволять реалізувати процеси аналізу та візуалізації даних.

Оскільки C# задовольняє ці критерії, було вирішено обрати саме її.

Також потрібно обрати середовище розробки, в якому буде розроблятись код проекту. Visual Studio 2022 надає графічне інтерактивне середовище розробки (IDE). IDE крім іншого надає інструменти перевірки синтаксису, файлового менеджменту та інструменти організації проектів[15]. Оскільки це ПЗ було розроблено компанією Microsoft, яка так само розробила C#, то в них хороша сумісність, і розробка коду буде комфортною в цьому середовищі. Крім того Visual Studio це дуже гнучкий інструмент, який дозволяє реалізувати всі процеси в одній програмі, а саме:

- Написання коду проекту, який відповідає за всю логіку.
- Створення візуального інтерфейсу.

- Контроль БД.

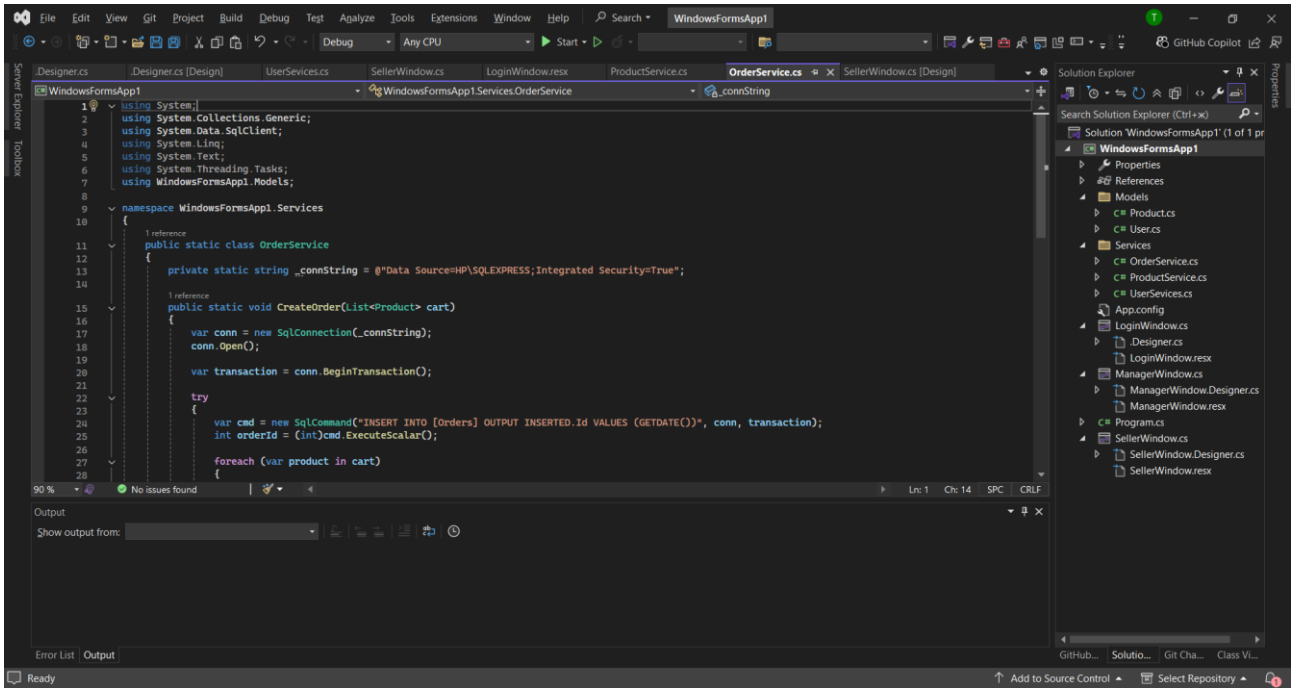

Рисунок 2.1 – Інтерфейс Visual Studio 2022

Для розробки і контролю БД було обрано Microsoft SQL Server. Це також програмний продукт компанії Microsoft, що чудово працює з іншими їх програмами. Ця програма використовує синтаксис Transact-SQL, що дозволить використовувати аутентифікацію Microsoft яка в свою чергу дозволяє легко підключити базу даних до проекту, та гарантувати високий рівень безпеки даних.

Також в розробці використовується SSMS (SQL Server Management Studio), інтерфейс якого зображено на рисунку 2.2. Це програмне забезпечення яке дозволяє керувати серверами Microsoft SQL Server, та використовувати SQL-запити, для створення, редагування та звернення до баз даних. Воно інкапсулює багато можливостей, раніше доступних в інших інструментах, а також пропонує багато нових можливостей. Більшість цих нових можливостей можна згрупувати в чотири основні категорії: зміни середовища, інтегроване керування, розширені можливості створення запитів та покращене керування проектами [16].

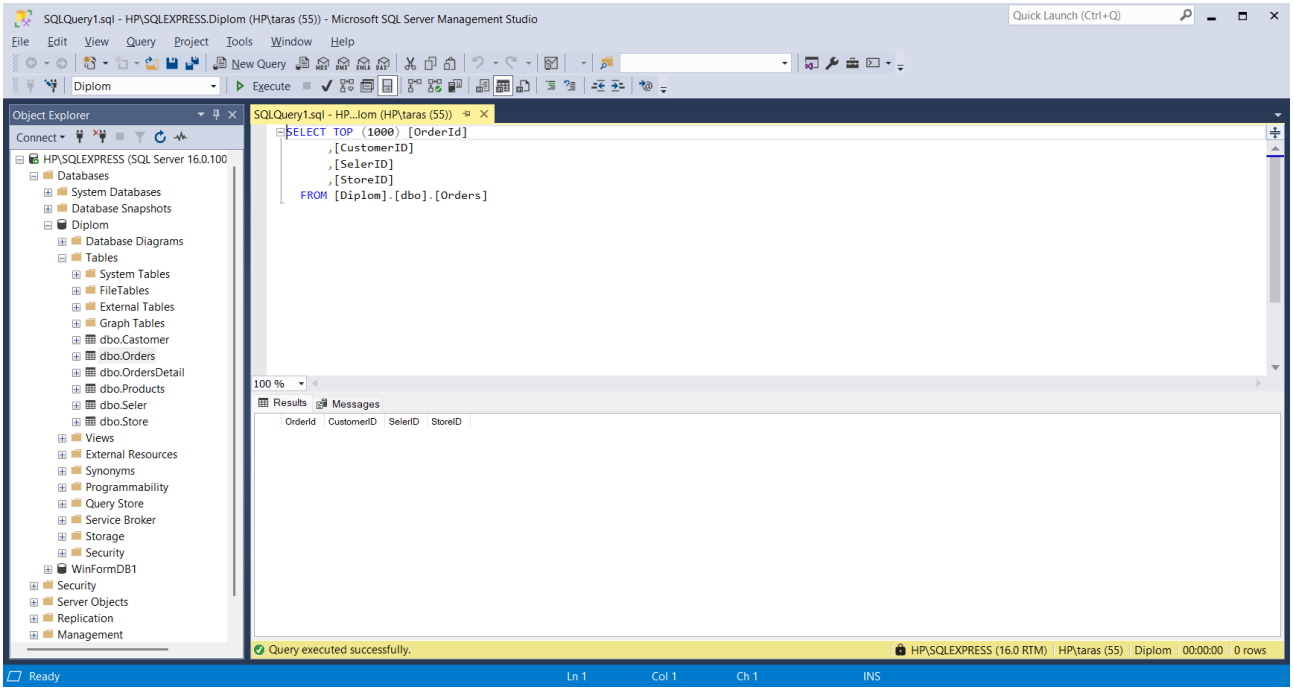

Рисунок 2.2 – Інтерфейс SQL Server Management Studio

В якості системи контролю версій було обрано GitHub. Це найбільш популярний вебсервіс для спільної розробки програмного забезпечення, що базується на системі керування версій Git. Цей сервіс в безкоштовному тарифному плані дозволяє використовувати функції, які допомагають контролювати процес розробки коду, та за необхідності повертатись до попередніх версій.

2.3 Визначення користувачів та способів використання

Процес проектування програмного забезпечення передбачає створення специфікацій на основі переліку вимог, які формуються перед початком проектування [17].

В першу чергу нам потрібно знати які саме користувачі передбачені в нашій програмі. Щоб такий процес зробити простішим і зрозумілішим використовують діаграми використання. Це один з видів діаграм, які чітко візуалізують які є користувачі по ролях, що саме вони можуть робити в програмі

і що не можуть робити. Без допомоги персонажів дизайнери схильні надмірно покладатися на свій особистий досвід або припущення. Персони допомагають прив'язати дизайн продукту до реальних потреб користувачів, зменшуючи ризик погано поінформованого вибору[18]

Це зручний і швидкий спосіб візуалізувати дану частину проекту, та полегшити процес занурення в процес нових спеціалістів.

Першочерговим користувачем є клієнт. Його функціонал в певному роді обмежений і доступні йому тільки ті функції, які не сильно впливають на систему і стосуються створення та часткового редагування замовлення. До цього списку входять:

- Авторизація.
- Створення замовлень.
- Видалення замовлень.
- Оплата замовлення.
- Перегляду своєї корзини.
- Перегляд своєї корзини замовлень.

Цей функціонал продемонстровано в діаграмі на рисунку 2.3.

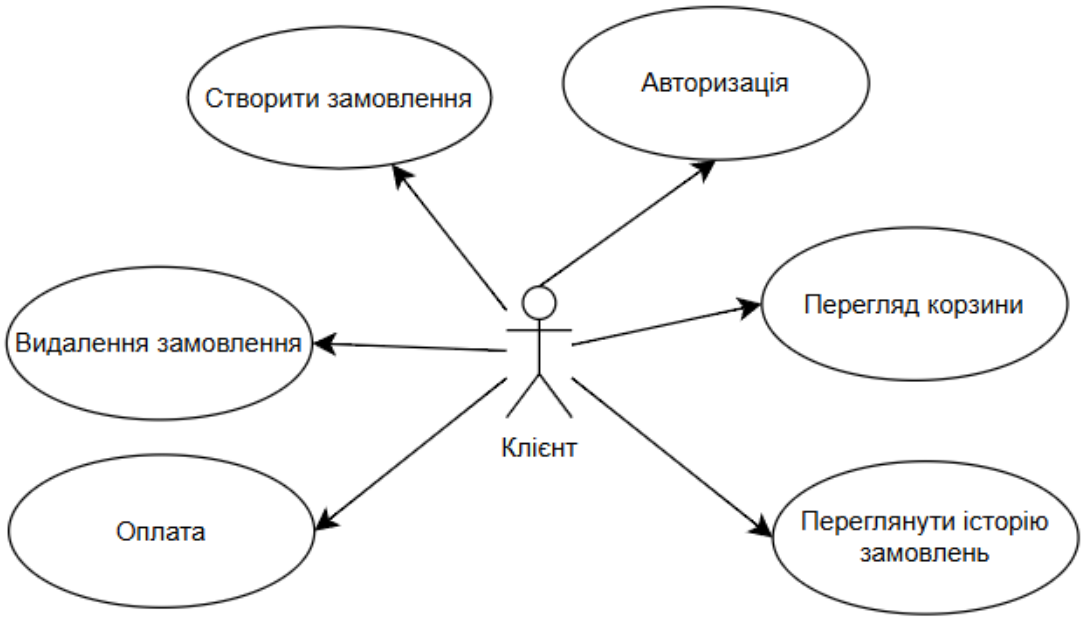

Рисунок 2.3 – Діаграма використання для актора ”Клієнт"

Наступним є продавець. В нього вже ширший функціонал оскільки він є працівником магазину і для роботи йому потрібно більше функцій. Його список варіантів використання виглядає наступним чином:

- Авторизуватись.
- Зареєструвати клієнта.
- Створити замовлення.
- Закрити замовлення.
- Відредагувати замовлення.
- Переглянути історію замовлень.
- Розрахувати клієнта.
- Переглянути особисту статистику.

Отже в продавця вже є ширший функціонал, який дозволяє йому виконувати певний список робіт і більше взаємодіяти з системою. Це продемонстровано на рисунку 2.4.

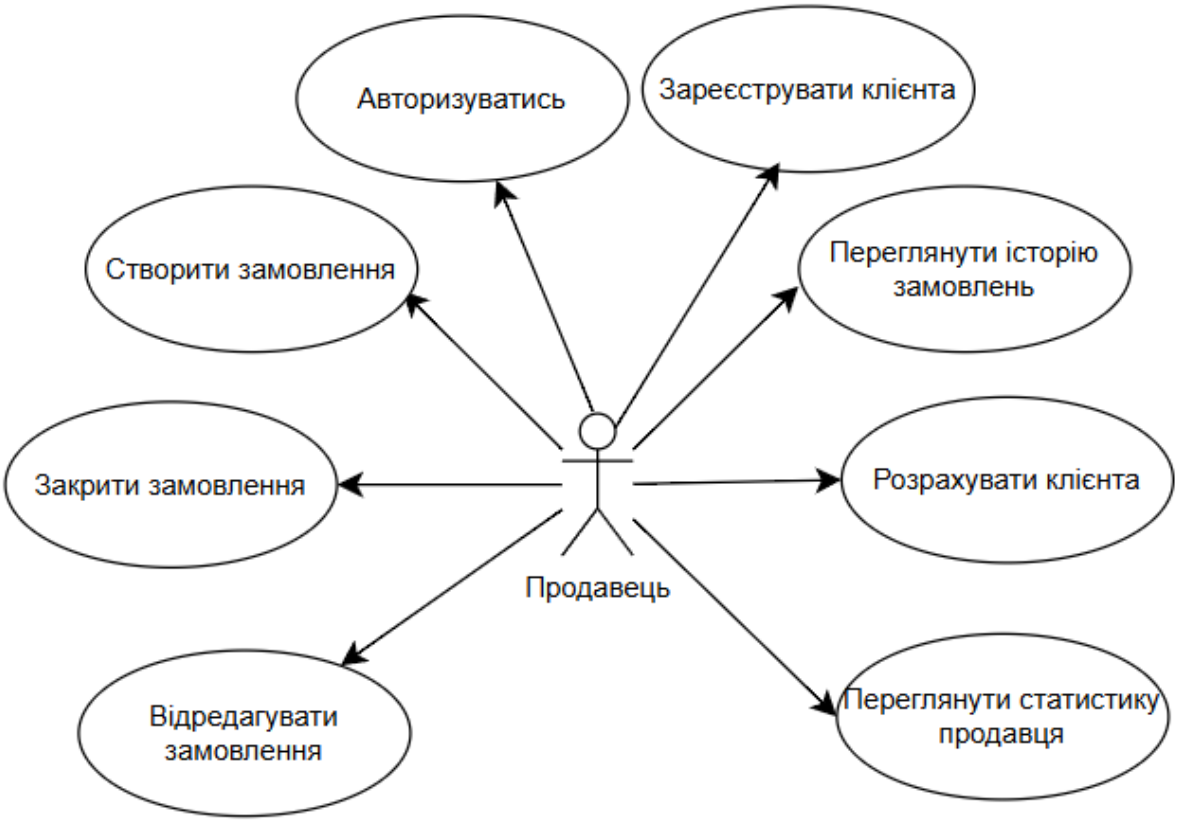

Рисунок 2.4 – Діаграма використання для актора ”Продавець”

Більшим списком повноважень володіє керівник магазинку, оскільки йому потрібно приймати більш важливі рішення та бачити ширшу картину що до статистики магазину. Його список можливостей виглядає наступним чином:

- Авторизуватись.
- Зареєструвати клієнта.
- Створити замовлення.
- Закрити замовлення.
- Відредагувати замовлення.
- Переглянути історію замовлень.
- Розрахувати клієнта.
- Переглянути особисту статистику.

- Переглянути статистику магазину.
- Редагувати ціну в замовленні (знижка).
- Змінити деталі замовлення.

Список цих можливостей візуалізовано в діаграмі використання на рисунку 2.5.

Рисунок 2.5 – Діаграма використання для актора ”Керівник”

Найбільш вповноваженим користувачем є адміністратор. До його списку можливостей входять такі функції як:

- Авторизуватись.
- Зареєструвати клієнта.
- Видалення акаунта клієнта.
- Створити замовлення.

- Закрити замовлення.
- Відредагувати замовлення.
- Переглянути історію замовлень.
- Розрахувати клієнта.
- Переглянути особисту статистику продавців.
- Переглянути статистику магазинів.
- Редагувати ціну в замовленні (знижка).
- Нарахування бонусів.
- Змінити деталі замовлення.

Весь цей список візуалізовано на рисунку 2.6.

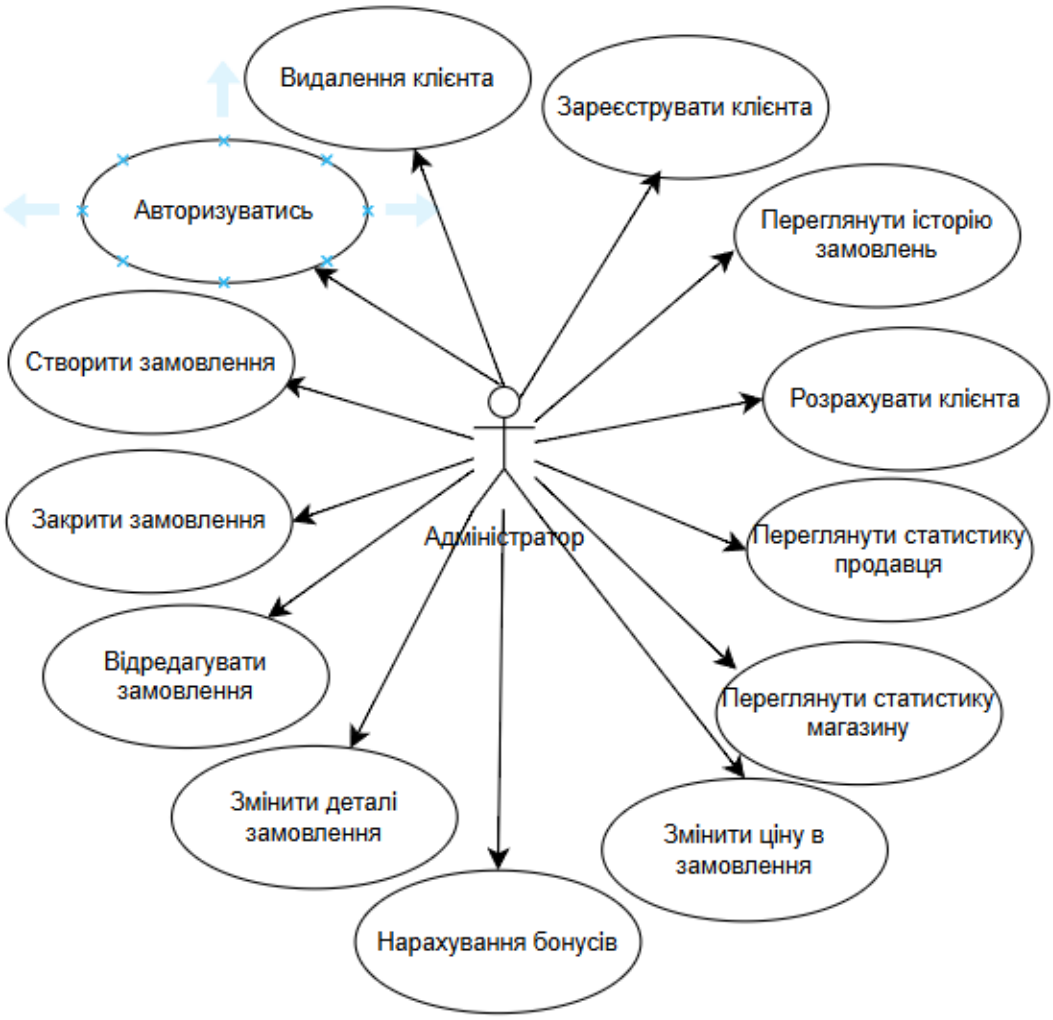

Рисунок 2.6 – Діаграма використання для актора ”Адміністратор”

В загальному можна помітити що в більшості акторів є багато схожих можливостей, тобто система має певну ієрархію, де вища посада має доступ до більш високорівневих функцій. При цьому більшість можливостей в користувачів є спільними, тобто їх можна об’єднати в одну спільну діаграму, яка показуватиме можливості користувачів у вигляді цілісної системи

Таким чином було створено діаграму подану на рисунку 2.7, на якій продемонстровано повноцінну систему всіх акторів, та їх можливостей.

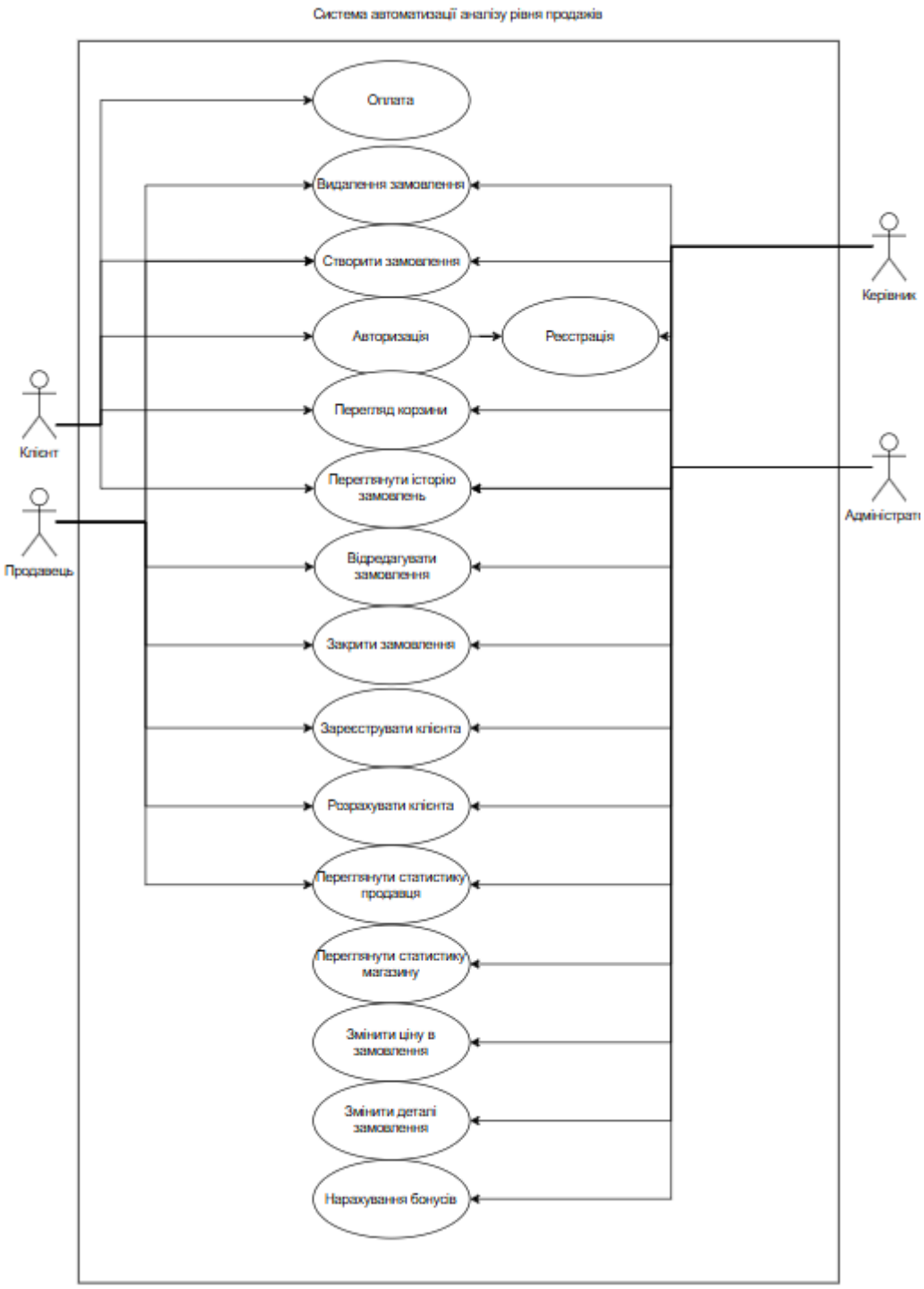

Рисунок 2.7 – Діаграма використання для всіх ролей в системі

Також перед процесом написання коду слід створити діаграми структури, та алгоритмів дій. Це дозволить зрозуміти швидко поділити код на різні модулі, та описати взаємодію між ними.

Основою є діаграма архітектури (див. рисунок 2.8). Вона дозволить розбити повноцінний проект на складові, та побачити які будуть основні модулі, та як вони будуть взаємодіяти один з одним.

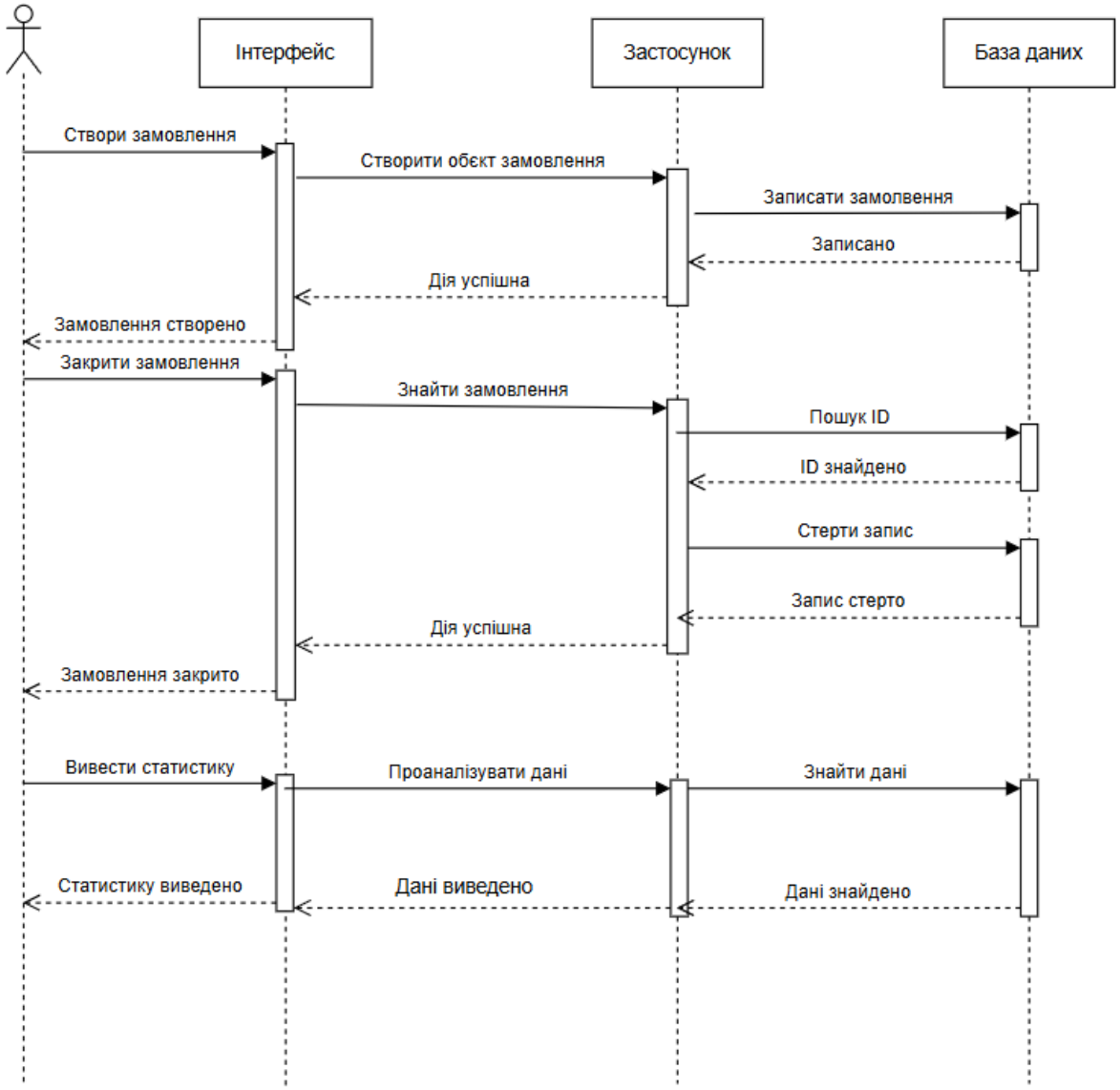

Рисунок 2.8 – Архітектура система аналізу

Також слід розробити алгоритми процесів для найчастіших дій в системі, а саме: створення замовлення, закриття замовлення, та виводу даних статистики (див. рисунок 2.9). Це також дозволить краще зрозуміти як саме буде функціонувати система, та як саме її компоненти будуть взаємодіяти між собою.

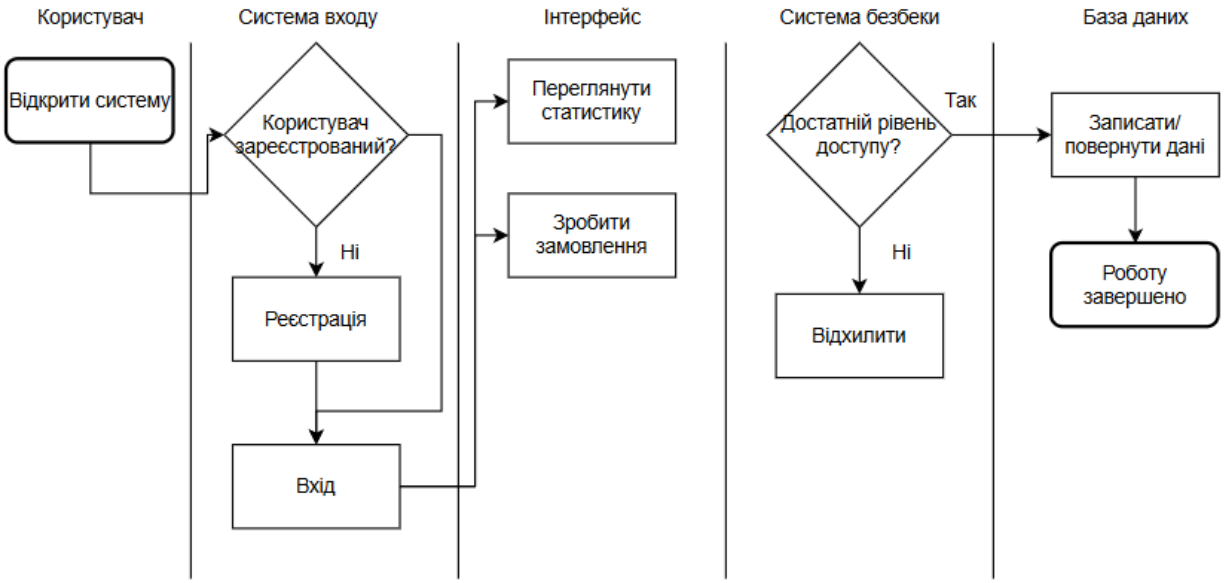

Рисунок 2.9 – Діаграма алгоритмів процесів системи

Всі ці діаграми дозволили зрозуміти та спроектувати систему, і як наслідок спростити подальшу розробку. Сам процес створення документації для системи і проекту в цілому дозволяє в значній мірі спростити і пришвидшити розробку, а також процес залучення нових розробників, адже діаграми дозволяють простіше ознайомитись з проектом, ніж перечитування всього масиву коду і перегляду всіх елементів.

2.4 Створення та налаштування бази даних

Процес проектування БД являє собою послідовність переходів від неформального мовного опису інформаційної структури предметної області до формалізованого опису об'єктів предметної області в термінах деякої моделі. [19].

База даних є одним з ключових елементів готового програмного продукту. Від її вигляду буде залежати подальше проектування і написання коду, тому розробити структуру БД слід ще на етапі проектування.

При створені БД слід враховувати наступні вимоги:

- Використання нормальних форм.
- Цілісність даних.
- Запобігання помилок та втрати даних.
- Можливість масштабування та модифікації.

Дотримуючись цих вимог ми створюємо ефективну БД, яка дозволить зручно використовувати програму, та скоротить час на звернення та обробку інформації.

2.5 Проектування БД

Першим етапом створення будь якої БД є її проектування. Важливо розуміти які саме таблиці будуть створені, і які типи зв’язку в них будуть. Для цього зручно використовувати UML –діаграми, на яких чітко видно які ми створюємо таблиці, їх поля та зв’язки, а також як ці таблиці між собою пов’язані, і як вони формують повноцінну базу даних.

Таку діаграму для створеної нами бази даних наведено нижче на рисунку 2.10.

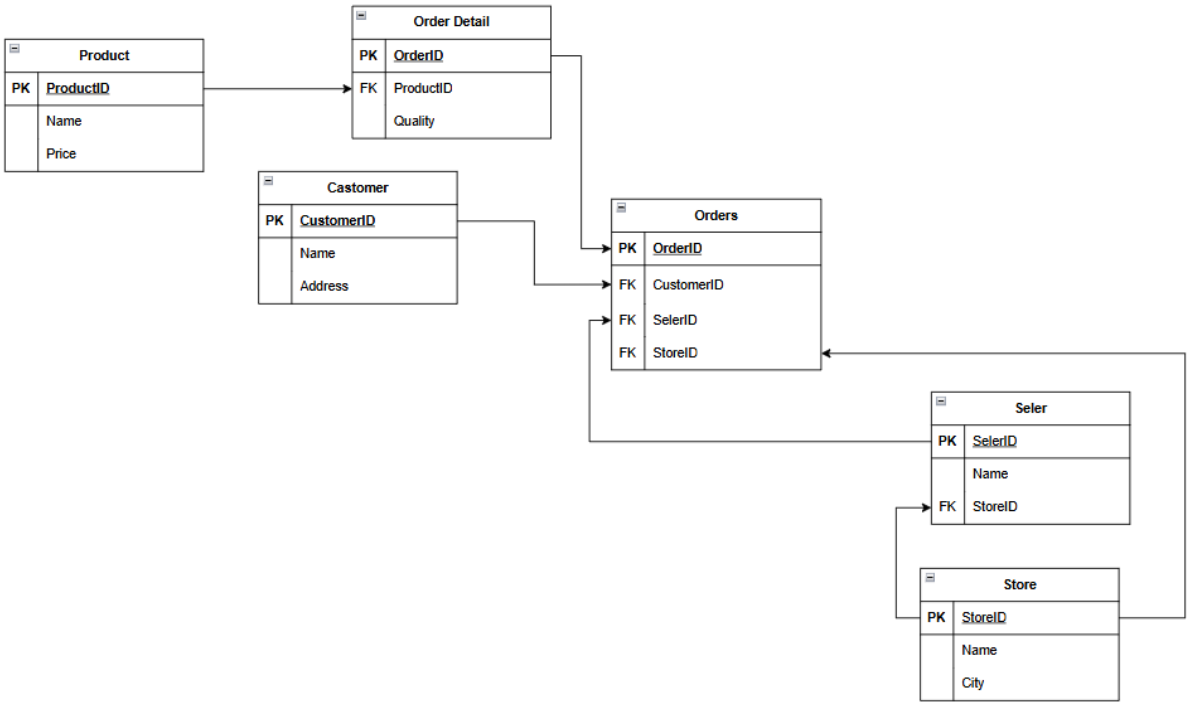

Рисунок 2.10 – UML-діаграма бази даних

Дана діаграма зручно демонструє які в нас є таблиці, та як вони між собою пов’язані.

2.5.1 Варіанти створення таблиць

В SSMS є декілька способів створення таблиць: через інтерфейс та за допомогою SQL –запитів.

Візуальний спосіб більш інтуїтивний. Він не вимагає знання мови SQL, проте налаштування в ньому займають більше часу. Цей спосіб більше підходить початківцям, які ще не встигли ознайомитись з мовою SQL, і починають вивчати структуру баз даних. Приклад створення таблиці через візуальний інтерфейс наведено нижче на рисунку 2.11.

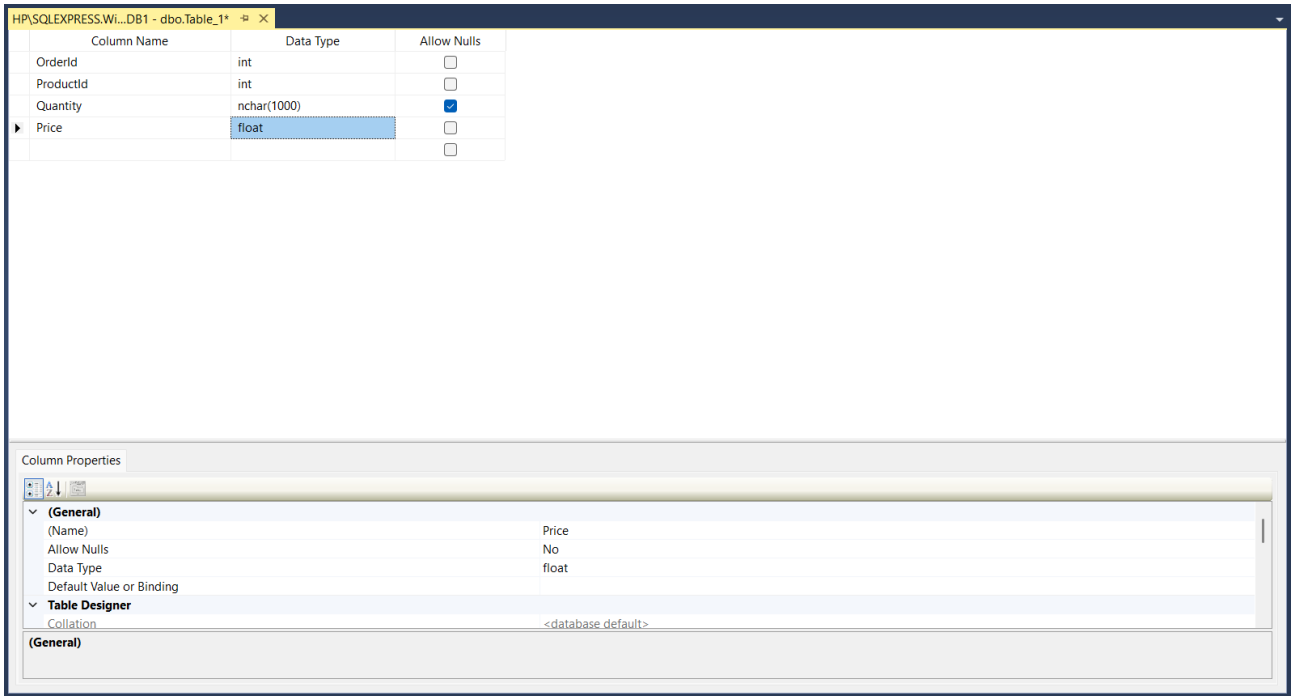

Рисунок 2.11 – Вікно створення таблиці

Іншим, більш популярним, способом є створення через SQL –запити. Для більшості користувачів цей спосіб більш зручний, оскільки через запит ми можемо одразу прописати всі потрібні нам дані, та налаштувати таблицю саме так як нам це потрібно, при цьому не витрачаючи час на пошук необхідного поля, яке відповідає за конкретне налаштування.

Приклад створення таблиці через SQL-запит наведено нижче на рисунку 2.12.

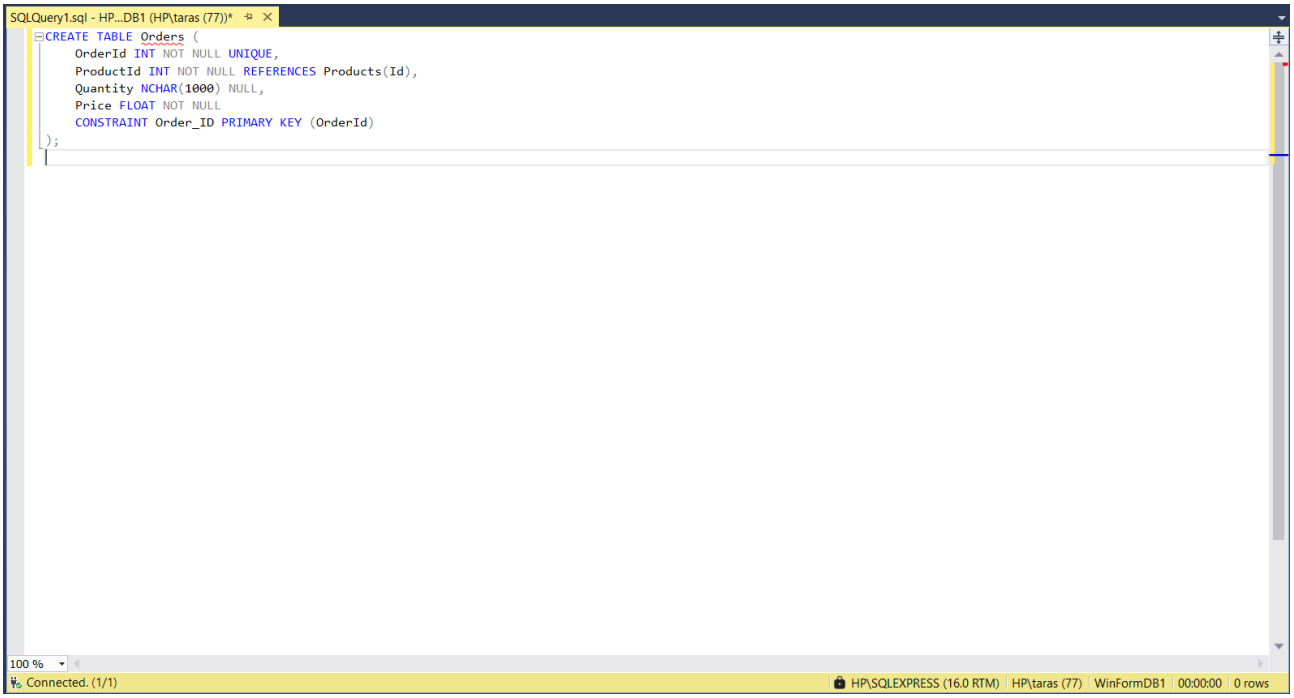

Рисунок 2.12 – Створення таблиці через SQL-запит

Як можна побачити з даного рисунку цей спосіб є зручнішим, оскільки в запиті одразу прописуються всі потрібні атрибути для полів. При достатньому знанні мови SQL і розумінні структури БД написання такого запиту займає набагато менше часу ніж використання візуального інтерфейсу.

2.5.2 Створення та налаштування бази даних

Першою буде таблиця для товарів під назвою “Products”. Вона містить такі поля як ID, назва, категорія товару, ціна та опис. Лістинг для даної таблиці наведений нижче в лістингу 2.1.

Лістинг 2.1 – SQL–запит для створення таблиці Products

```
CREATE TABLE Products(
      ProductId INT NOT NULL UNIQUE,
      Name NCHAR(100) NOT NULL,
      Price DECIMAL NOT NULL,
      Category NCHAR(10),
      Description NCHAR(1000) NULL,
```

```
     CONSTRAINT Product_Id PRIMARY KEY (ProductId)
)
```

Наступна таблиця Store. Вона описує магазини і містить такі поля як ID, назва та місто де магазин розташований. В лістингу 2.2 продемонстровано SQL–запит для даної таблиці.

Лістинг 2.2 – SQL-запит для створення таблиці Store

```
CREATE TABLE Store(
     StoreId INT NOT NULL UNIQUE,
     Name NCHAR(100) NOT NULL,
     City NCHAR(100) NOT NULL,
     CONSTRAINT Store_Id PRIMARY KEY (StoreId)
)
```

В таблиці Selers будуть зберігатись дані про продавців мережі. Ця таблиця містить такі поля як ID продавця, його ім’я і ID магазину до якого він прикріплений. SQL–запит для створення цієї таблиці міститься в лістингу 2.3.

Лістинг 2.3 – SQL–запит для створення таблиці Selers

```
CREATE TABLE Selers(
     SelerId INT NOT NULL UNIQUE,
     Name NCHAR(100) NOT NULL,
     StoreId INT NOT NULL REFERENCES Store(StoreId),
     CONSTRAINT Seler_Id PRIMARY KEY (SelerId)
)
```

Наступний наведений лістинг 2.4 в якому продемонстровано SQL-запит створення таблиці Castomer. Ця таблиця містить такі поля як ID клієнта, його ім’я та адреса для доставки.

Лістинг 2.4 – SQL–запит для створення таблиці Products

```
CREATE TABLE Castomer(
     CastomerId INT NOT NULL UNIQUE,
     Name NCHAR(100) NOT NULL,
     Adress NCHAR(100) NOT NULL,
     CONSTRAINT Castomer_Id PRIMARY KEY (CastomerId)
```

```
)
```

Наступною є таблиця Orders, лістинг 2.5, якої, наведено нижче. Вона містить такі поля як ID замовлення, ID клієнта який це замовив, ID продавця який замовив товар і ID магазину в якому було створено замовлення. Поля CustomerID, SelerID і StoreID мають властивість NULL, яка дозволяює залишати поля без значень, на випадок якщо клієнт замовляв онлайн, або не хоче залишати свої контактні дані.

Лістинг 2.5 – SQL–запит для створення таблиці Orders

```
CREATE TABLE Orders(
     OrderId INT NOT NULL UNIQUE,
     CustomerID INT NULL REFERENCES Castomer(CastomerId),
     SelerID INT NULL REFERENCES Seler(SelerId),
     StoreID INT NULL REFERENCES Store(StoreID),
     CONSTRAINT Order_ID PRIMARY KEY (OrderId)
)
```

Остання таблиця OrdersDetail, яка використовується для уникнення звя’зку багато до багатьох між таблицями замовлень та товарів. SQL–запит для створення цієї таблиці міститься в лістингу 2.6. Таблиця містить такі поля як ID замовлення та ID продукту який замовили, та їх кількість.

Лістинг 2.6 – SQL–запит для створення таблиці OrdersDetail

```
CREATE TABLE OrdersDetail(
     OrderId INT NOT NULL REFERENCES Orders(OrderId),
     ProductId INT NOT NULL REFERENCES Products(ProductId),
     Quality INT NOT NULL
)
```

Таким чином ми створили БД, яка гармонічно зберігає дані, та має нормальну форму, що дозволяє уникнути зв’язку багато до багатьох, що дозволить уникнути помилок та надмірної кількості звернень до бази даних в подальшій розробці проекту.

2.6 Авторизація та безпека

Немало важливим є питання безпеки. В наш час дані це досить ціна валюта, тому для їх збереження використовують різні заходи.

Одним з таких є аутентифікація при вході та різні рівні доступу для різних типів користувачів.

В даному проекту для входу в програму користувач повинен ввести свій логін і пароль, для їх збереження використовується окрема таблиця в базі даних, діаграма якої продемонстрована нижче на рисунку 2.13.

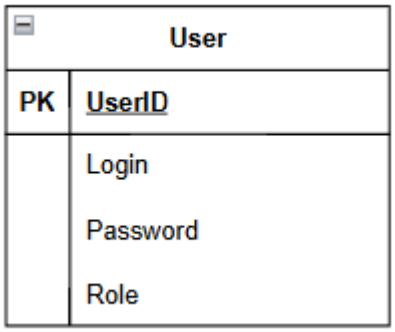

Рисунок 2.13 – Діаграма таблиці User

Відповідно після введення даних програма визначає яка роль в користувача, відкриває обране для даної ролі вікно, та дає доступ до необхідних функцій, при цьому забороняючи доступ до функцій з вищим рівнем доступу. SQL-запит для створення даної таблиці наведено в лістингу 2.7.

Лістинг 2.7 – SQL-запит для створення таблиці User

```
CREATE TABLE User(
     UserID INT NOT NULL UNIQUE,
     Login NCHAR(100) NOT NULL,
     Password INT NOT NULL,
     Role NCHAR(20),
     CONSTRAINT User_Id PRIMARY KEY (UserID)
)
```

Впровадження прозорого шифрування даних може призвести до відчутних накладних витрат, але значно підвищує безпеку бази даних [20].

Таким чином таблиця зберігає всі необхідні дані для входу і дозволяє уникнути можливості неправомірного використання даних, і відповідно їх виходу за межі системи.

2.7 Висновки до 2-го розділу

В даному розділі було проведено підготовку до написання коду програми. Розглянуто які саме будуть ролі в проекті, та який функціонал буде в кожного користувача. Крім того було спроектовано та реалізовано базу даних, в якій проведено нормалізацію таблиць та прописано засоби безпеки, які дозволять уникнути дублювання даних, або їх втрати чи неповноцінності. Окремий розділ виділено під безпеку, що дозволить уникнути втручання в програму небажаних користувачів, які не мають доступу до системи.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ ДЛЯ АВТОМАТИЗОВАНОГО АНАЛІЗУ РІВНЯ ПРОДАЖ

3.1 Послідовність дій та логіка реалізації

У цьому підрозділі описано кроки, які виконує розробник, щоби ввести в дію аналітичний модуль.

Налаштування середовища. Створюємо docker-compose.yml, де піднімаються api (ASP.NET Core) та db (PostgreSQL 16-alpine). Локально розробник запускає docker compose up -d, отримуючи бірюзове середовище, і вже тут застосовує початкові міграції dotnet ef migrations add Init.

Первинне завантаження даних. Через REST-ендпоінт /seed або SQL-скрипт імпортуємо CSV-файли продажів; це заповнює таблиці Products, Receipts, ReceiptItems.

Конфігурація Entity Framework Core. Використовуємо lazy-loading-пакет Microsoft.EntityFrameworkCore.Proxies, однак у сервісах виключаємо проксі-класи й читаємо AsNoTracking() із projection-select, щоби мінімізувати споживання RAM.

Імплементація Repository + Unit of Work. Усі запити до БД проходять через EfRepository<T>, а транзакції — через UnitOfWork у сервісному шарі. Це спрощує тестування (InMemoryDbContext) і забезпечує атомарність SaveAsync().

Бізнес-логіка аналітики. У SalesAnalyticsService методи GetRevenueAsync та GetAovAsync агрегують дані з заданого діапазону. У складніших сценаріях (CLV, RPR) застосовуватимуться LINQ-групування та когортні проєкції.

API-контролери. AnalyticsController публікує конвенційно версіонований шлях api/v1/analytics/*. Використовуємо Swagger та патерн Skinny Controller → Fat Service.

Захист і кешування. Авторизація — JWT Bearer, кеш — Redis (IDistributedCache). Найгарячіші запити (GMV/AOV/CR сьогодні) кешуються на 15 с.

3.2 Ключові кодові рішення

Примітка. Повні лістинги збережено у репозиторії *SalesAnalytics* (GitHub). Тут наведено скорочені, але працездатні фрагменти.

3.2.1 Моделі EF Core (C#)

В проекті використовуються 3 основних клас.

Перший з них Product (див. лістинг 3.1), який зберігає дані про товар, такі як назва, ціна і категорія.

Лістинг 3.1 – Клас Product

```
public class Product
{
    public Guid Id          { get; set; }
    public string Name       { get; set; } = string.Empty;
    public decimal Price     { get; set; }
    public string Category   { get; set; } = string.Empty;
    public ICollection<ReceiptItem> ReceiptItems { get; set; } =
new List<ReceiptItem>();
}
```

Dластивість ReceiptItems (див. лістинг 3.2) визначає зв’язок один-до-багатьох з елементами чеків. ReceiptItem є проміжною сутністю, що об’єднує чек і товар та містить кількість.

Лістинг 3.2 – Клас ReceiptItem

```
public class ReceiptItem
{
    public Guid Id        { get; set; }
    public Guid ReceiptId { get; set; }
```

```
    public Guid ProductId { get; set; }
    public int  Quantity  { get; set; }

    public Product Product { get; set; }
    public Receipt Receipt { get; set; }
}
```

Receipt (див. лістинг 3.3) представляє сам продаж і агрегує колекцію позицій Items. Використання ідентифікаторів Guid спрощує масштабування та уникнення колізій у розподіленому середовищі.

Лістинг 3.3 – Клас Receipt

```
public class Receipt
{
    public Guid Id        { get; set; }
    public Guid SellerId  { get; set; }
    public Guid StoreId   { get; set; }
    public DateTime Date  { get; set; }
    public ICollection<ReceiptItem> Items { get; set; } = new
List<ReceiptItem>();
}
```

3.2.2 Контекст бази даних

ShopDbContext (див. лістинг 3.4) успадковує DbContext і реєструє три колекції DbSet. У методі OnModelCreating налаштовується схема: задаємо первинний ключ для Product та зовнішній ключ від ReceiptItem до Product. Інші зв’язки конфігуруються за конвенціями EF Core, що скорочує обсяг коду налаштувань.

Лістинг 3.4 – ShopDbContext

```
public class ShopDbContext : DbContext
{
    public ShopDbContext(DbContextOptions<ShopDbContext> opts) :
base(opts) {}

    public DbSet<Product> Products   => Set<Product>();
    public DbSet<Receipt> Receipts   => Set<Receipt>();
```

```
    public DbSet<ReceiptItem> Items  => Set<ReceiptItem>();

    protected override void OnModelCreating(ModelBuilder b)
    {
        b.Entity<Product>().HasKey(p => p.Id);
        b.Entity<ReceiptItem>()
            .HasOne(i => i.Product)
            .WithMany(p => p.ReceiptItems)
            .HasForeignKey(i => i.ProductId);
    }
}
```

3.2.3 Репозиторій та Unit of Work

Шаблон Repository (див. лістинг 3.5) інкапсулює доступ до даних, надаючи абстракцію поверх EF Core. Завдяки AsNoTracking() читання сутностей не тримає їх у змінному стані, що скорочує споживання пам’яті при сценаріях тільки читання.

Лістинг 3.5 – Інтерфейс IRepository<T> і реалізація EfRepository<T>

```
public interface IRepository<T> where T : class
{
    Task<IReadOnlyList<T>> GetAllAsync();
    Task<T?> GetAsync(Guid id);
    Task AddAsync(T entity);
    void Remove(T entity);
}

public sealed class EfRepository<T> : IRepository<T> where T :
class
{
    private readonly ShopDbContext _ctx;
    public EfRepository(ShopDbContext ctx) => _ctx = ctx;

    public async Task<IReadOnlyList<T>> GetAllAsync() =>
        await _ctx.Set<T>().AsNoTracking().ToListAsync();

    public async Task<T?> GetAsync(Guid id) =>
        await _ctx.Set<T>().FindAsync(id);

    public async Task AddAsync(T entity) => await
_ctx.AddAsync(entity);
    public void Remove(T entity) => _ctx.Remove(entity);
}
```

Unit of Work (див. лістинг 3.6) об’єднує кілька репозиторіїв і гарантує атомарне збереження змін через метод SaveAsync(). Такий підхід спрощує тестування: у юніт-тестах можна підставити InMemoryDbContext.

Лістинг 3.6 — IUnitOfWork та UnitOfWork

```
public interface IUnitOfWork
{
    IRepository<Product> Products { get; }
    IRepository<Receipt> Receipts { get; }
    Task<int> SaveAsync();
}

public sealed class UnitOfWork : IUnitOfWork
{
    private readonly ShopDbContext _ctx;
    public UnitOfWork(ShopDbContext ctx)
    {
        _ctx = ctx;
        Products = new EfRepository<Product>(ctx);
        Receipts  = new EfRepository<Receipt>(ctx);
    }
    public IRepository<Product> Products { get; }
    public IRepository<Receipt> Receipts  { get; }
    public Task<int> SaveAsync() => _ctx.SaveChangesAsync();
}
```

3.2.4 Сервіс аналітики продажів

Сервіс реалізує два показники: Revenue (див. лістинг 3.7) та Average Order Value. Дані агрегуються без проміжного підвантаження в пам’ять — використано LINQ-вирази, що транслюються в SQL-запит на стороні БД. Така реалізація масштабується на великі обсяги даних і швидко повертає результат.

Лістинг 3.7 — Revenue

```
public interface ISalesAnalytics
{
    Task<decimal> GetRevenueAsync(DateTime from, DateTime to);
    Task<double>  GetAovAsync(DateTime from, DateTime to);
}
```

```
public sealed class SalesAnalyticsService : ISalesAnalytics
{
    private readonly ShopDbContext _ctx;
    public SalesAnalyticsService(ShopDbContext ctx) => _ctx = ctx;

    public async Task<decimal> GetRevenueAsync(DateTime from,
DateTime to)
    {
        return await _ctx.Receipts
            .Where(r => r.Date >= from && r.Date <= to)
            .SumAsync(r => r.Items.Sum(i => i.Quantity *
i.Product.Price));
    }

    public async Task<double> GetAovAsync(DateTime from, DateTime
to)
    {
        var orders = await _ctx.Receipts
            .Where(r => r.Date >= from && r.Date <= to)
            .Select(r => new { Total = r.Items.Sum(i => i.Quantity
* i.Product.Price) })
            .ToListAsync();
        return orders.Count == 0 ? 0 : (double)orders.Average(o =>
o.Total);
    }
}
```

3.2.5 REST API (ASP.NET Core)

Контролер AnalyticsController (див. лістинг 3.8) є «тонким»: він лише приймає вхідні параметри, делегує обчислення сервісу й повертає результат у форматі JSON. Маршрутизація версійована (v1) і легко розширюється.

Лістинг 3.8 — AnalyticsController

```
[ApiController, Route("api/v1/analytics")]
public class AnalyticsController : ControllerBase
{
    private readonly ISalesAnalytics _svc;
    public AnalyticsController(ISalesAnalytics svc) => _svc = svc;

    [HttpGet("revenue")]
    public async Task<IActionResult> Revenue([FromQuery] DateTime
from, [FromQuery] DateTime to)
        => Ok(await _svc.GetRevenueAsync(from, to));
}
```

3.2.6 Docker-оркестрація

Файл docker-compose.yml (див. лістинг 3.9) піднімає два контейнери: api (ASP.NET Core) та db (PostgreSQL 16-alpine). Ефективне планування контейнерів вимагає балансування показників якості обслуговування між розподіленими ресурсами [21]. Зовнішні порти прокинуті для локальної налагоджувальної сесії. Стрічка підключення передається через змінну середовища, що спрощує деплой у різних оточеннях.

Лістинг 3.9 — docker-compose.yml

```
version: "3.9"
services:
  api:
    build: .
    ports: ["8080:8080"]
    depends_on: [db]
    environment:
      -
ConnectionStrings__Default=Host=db;Database=shop;Username=postgres
;Password=postgres
  db:
    image: postgres:16-alpine
    ports: ["5432:5432"]
    environment:
      - POSTGRES_PASSWORD=postgres
volumes:
  db-data:
```

3.2.7 Приклад юніт-тесту (xUnit)

Для забезпечення надійності та коректності функціонування розробленої інформаційної системи важливим є використання методів автоматизованого тестування, зокрема 'автоматизованої генерації тестових випадків' [22]

За допомогою фреймворку xUnit можна швидко перевірити правильність бізнес-логіки без запуску всього API та БД PostgreSQL. Юніт-тести гарантують

стабільність коду, дають можливість безпечно рефакторити та слугують «живою» документацією для команди.

- Tests.csproj – окремий .NET-проєкт із типом <IsPackable>false</IsPackable>; підключено NuGet-пакети xunit, xunit.runner.visualstudio, Microsoft.EntityFrameworkCore.InMemory та coverlet.collector (для збору покриття).
- TestDbContextFactory.cs – фабрика, що повертає налаштований in-memory ShopDbContext, із яким працюють юніт-тести.

Лістинг 3.10 — Фабрика in-memory контексту

```
public static class TestDbContextFactory
{
    public static ShopDbContext Create()
    {
        var options = new DbContextOptionsBuilder<ShopDbContext>()
            .UseInMemoryDatabase(Guid.NewGuid().ToString()) //
окрема БД для кожного тесту
            .Options;
        return new ShopDbContext(options);
    }
}
```

- UseInMemoryDatabase підключає провайдер Microsoft.EntityFrameworkCore.InMemory, тому дані зберігаються у RAM, а тест виконується швидко й ізольовано.
- Guid.NewGuid() гарантує унікальне ім’я БД, щоб паралельні тести не конфліктували.

Лістинг 3.11 — Приклад юніт-тесту методу GetRevenueAsync

```
public class SalesAnalyticsServiceTests
{
    [Fact]
    public async Task GetRevenueAsync_Returns_CorrectValue()
    {
        // Arrange — підготовка даних
        await using var ctx = TestDbContextFactory.Create();
```

```
        var tv = new Product { Id = Guid.NewGuid(), Name = "TV",
Price = 1000m };
        ctx.Products.Add(tv);
        ctx.Receipts.Add(new Receipt
        {
            Id   = Guid.NewGuid(),
            Date = new DateTime(2025, 01, 15),
            Items = { new ReceiptItem { Product = tv, Quantity = 2
} }
        });
        await ctx.SaveChangesAsync();

        var service = new SalesAnalyticsService(ctx);
        var from = new DateTime(2025, 01, 01);
        var to   = new DateTime(2025, 01, 31);

        // Act — викликаємо тестований метод
        var revenue = await service.GetRevenueAsync(from, to);

        // Assert — перевіряємо результат
        Assert.Equal(2000m, revenue);
    }
}
```

Arrange-Act-Assert: створюється ізольований контекст, заповнений мінімальним набором сутностей (1 продукт × 2 шт.); далі виконується метод *GetRevenueAsync* і перевіряється, що він повертає очікувану суму 2000m. Якщо бізнес-логіку змінять, тест стане «червоним», відразу попередивши розробника.

Запуск тестів та покриття коду

- Через CLI: dotnet test --collect:"XPlat Code Coverage".
- Через Visual Studio Test Explorer: ▶ Run All Tests..

Після запуску *coverlet.collector* формує файл coverage.cobertura.xml, який легко інтегрувати в CI (GitHub Actions, Azure Pipelines) та перетворити у HTML-звіт за допомогою ReportGenerator.

3.3 Інтерфейс користувача

Для забезпечення ефективного сприйняття інформації кінцевими користувачами, розроблена система повинна включати інструменти інтерактивної візуалізації даних [23].

Нижче наведено прототипи для всіх п’яти вікон програми. Кожен макет демонструє ключові елементи UI, що відповідають завданням конкретної секції.

Перше вікно (див. рисунок 3.1) показує загальну статистику магазину, а саме:

- Загальну вартість продажів.
- Онлайн продажі.
- Кількість замовлень.
- Кількість покупців.
- Список останіх покупок.
- Діаграми для візуалізації даних.

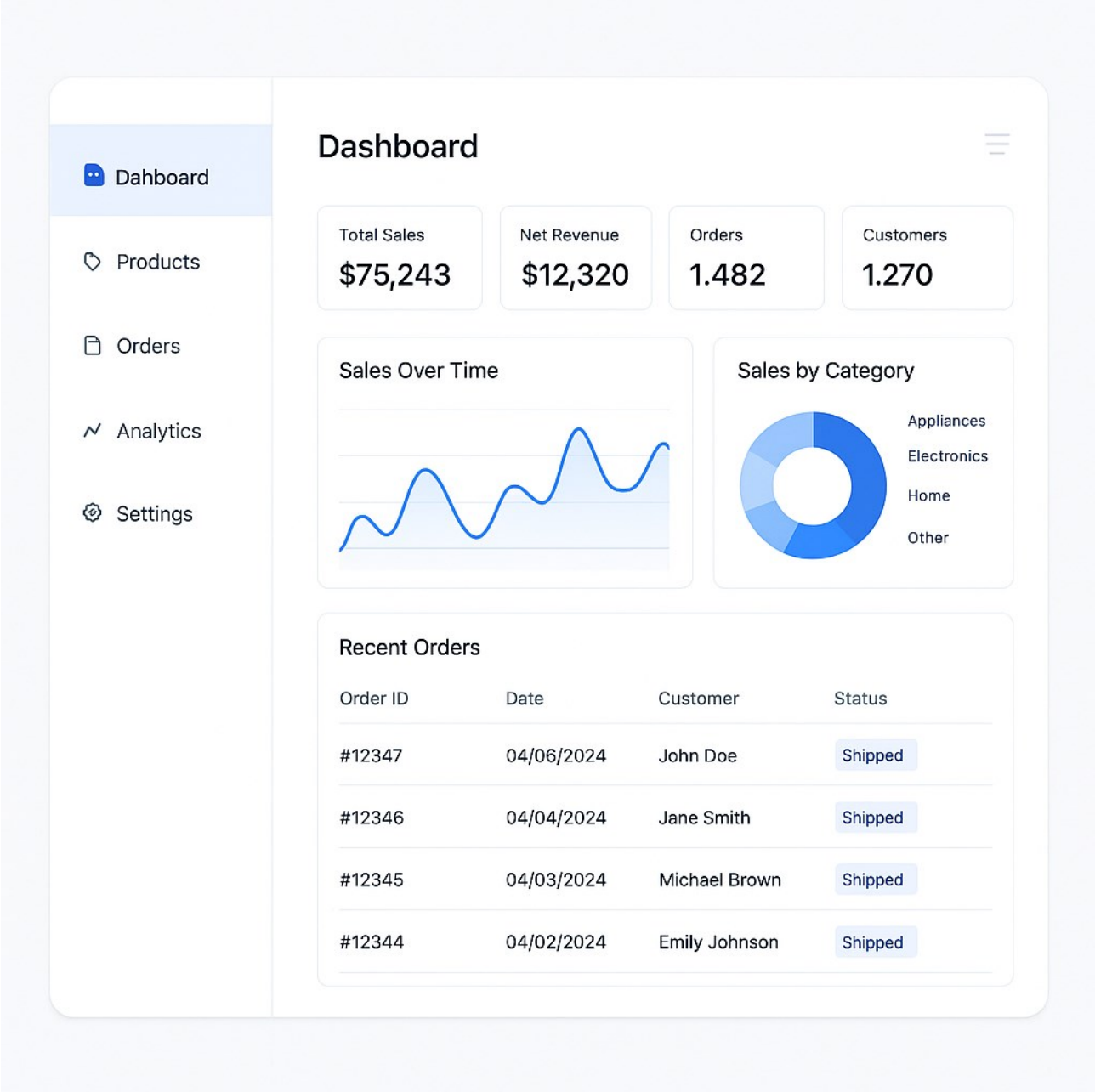

Рисунок 3.1 – Головне вікно програми

Наступне віно відображає всю можливу інформацю про список товарів в магазині (див. рисунок 3.2). Тут виведений список товарів які є на залишках, до якої категорії вони відносяться, їх ціна при продажі, залишок і статус оголошення про продаж. Також є можливість знайти продукт по назві, видалити, або додати новий.

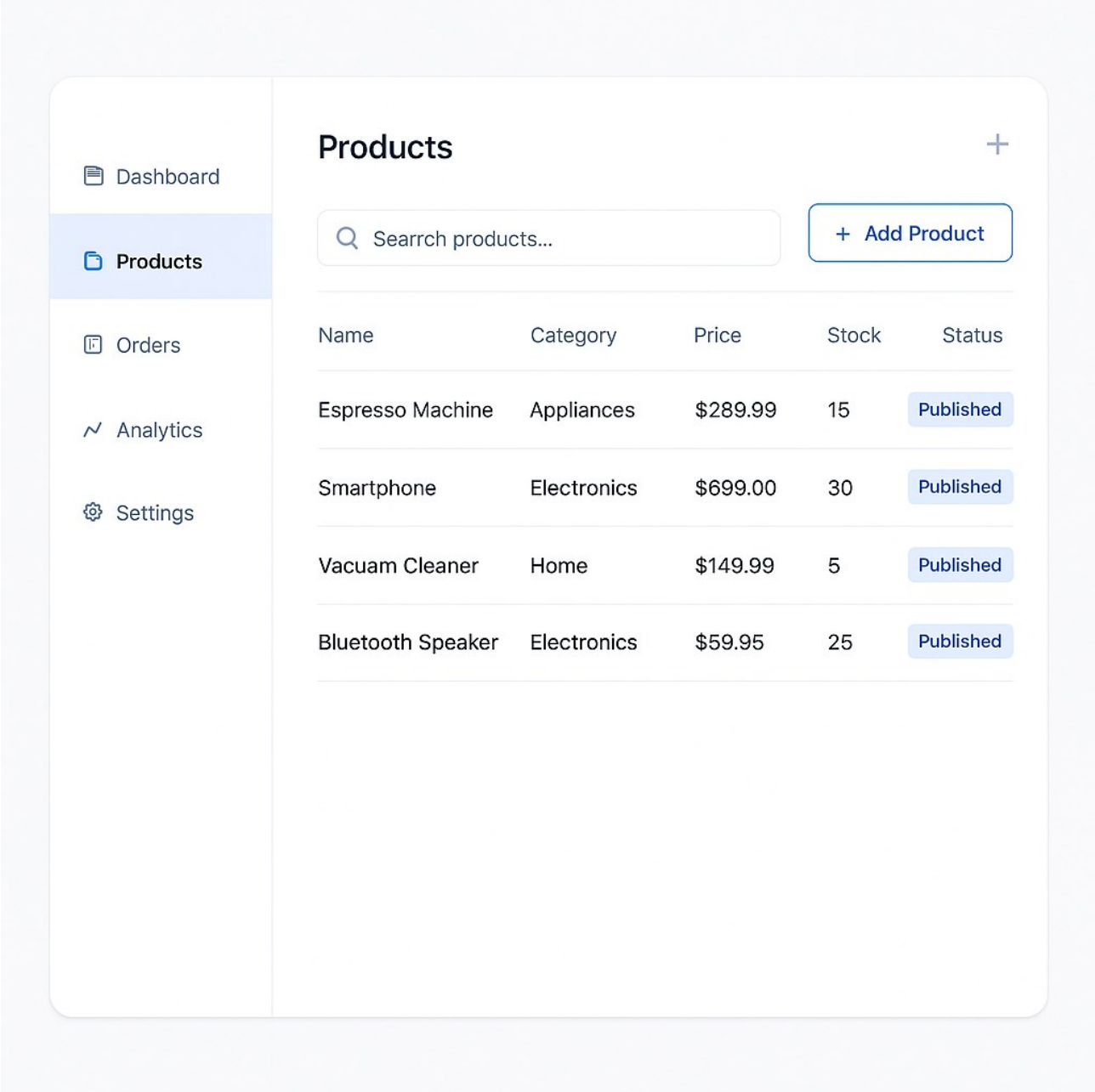

Рисунок 3.2 – Вікно товарів

В наступному вікні виведена інформація про продажі. Вікно містить такі поля як:

- ID замовлення.
- Дату здійснення покупки.
- Ім’я та прізвище клієнта.
- Статус покупки.

З функцій реалізований пошук по ID, а також можливість відсортувати за критерієм що нас цікавить.

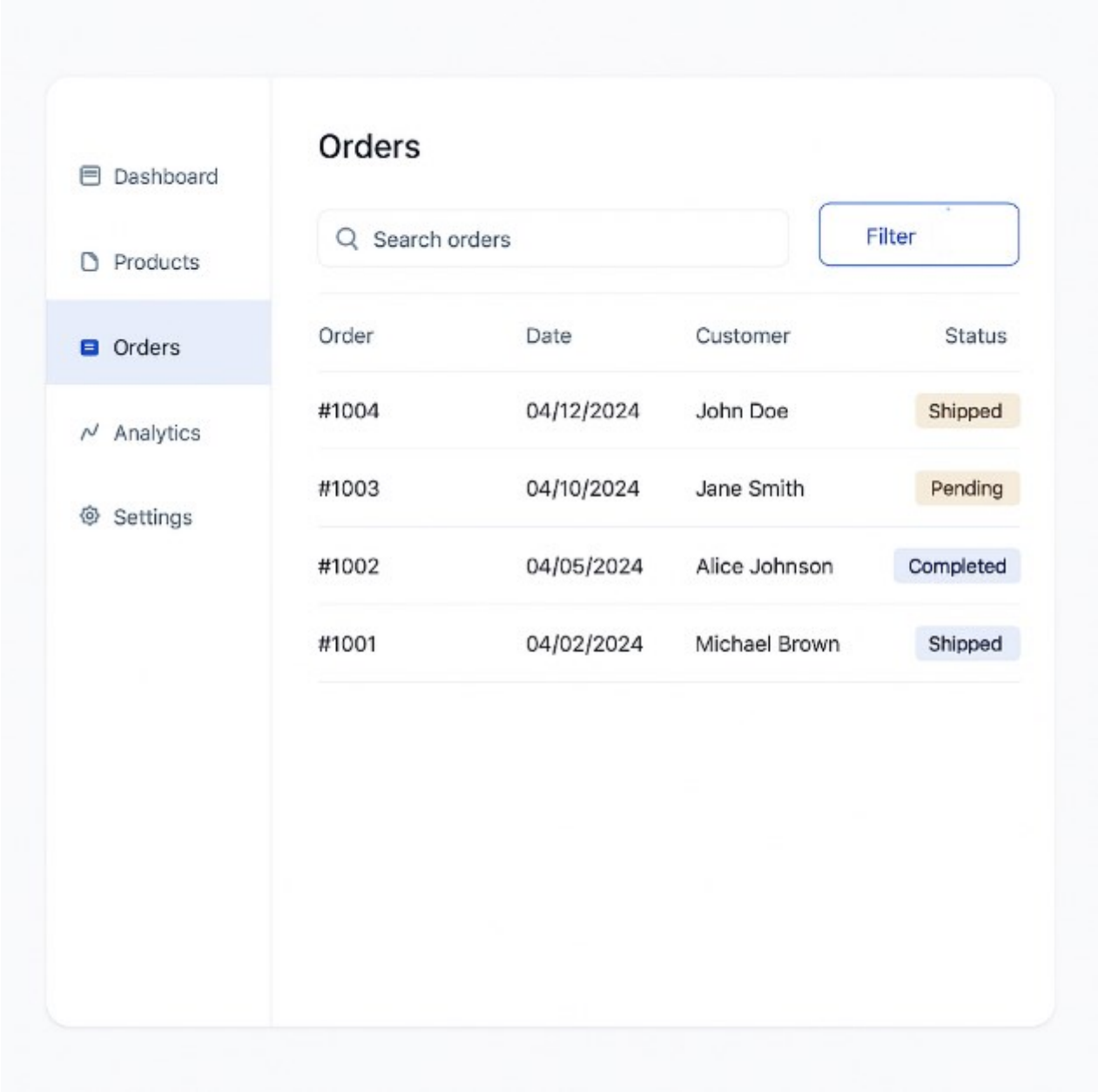

Рисунок 3.3 – Вікно замовлень

Наступне вікно з загальною аналітикою продажів (див. рисунок 3.4). В ньому виведено графік рівня продажів, за певні відрізки часу. Також виведено список найбільш продаваних категорій.

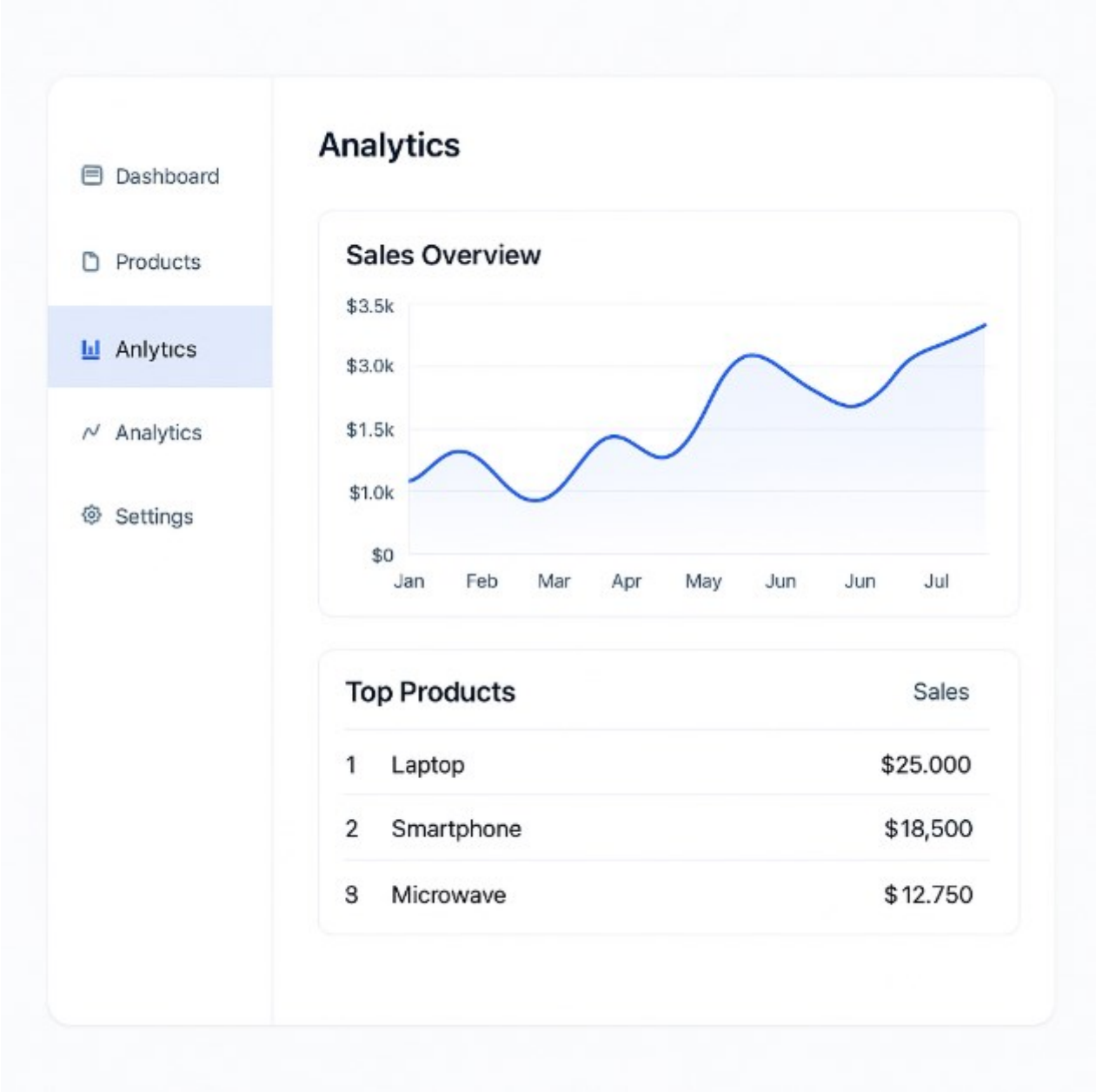

Рисунок 3.4 – Вікно загальної аналітики продаж

Останнім є вікно налаштувань кабінету користувача (див. рисунок 3.5). Тут надана можливість редагувати дані користувача, його ім’я та електронну пошту. Також змінити пароль та додати або прибрати опис.

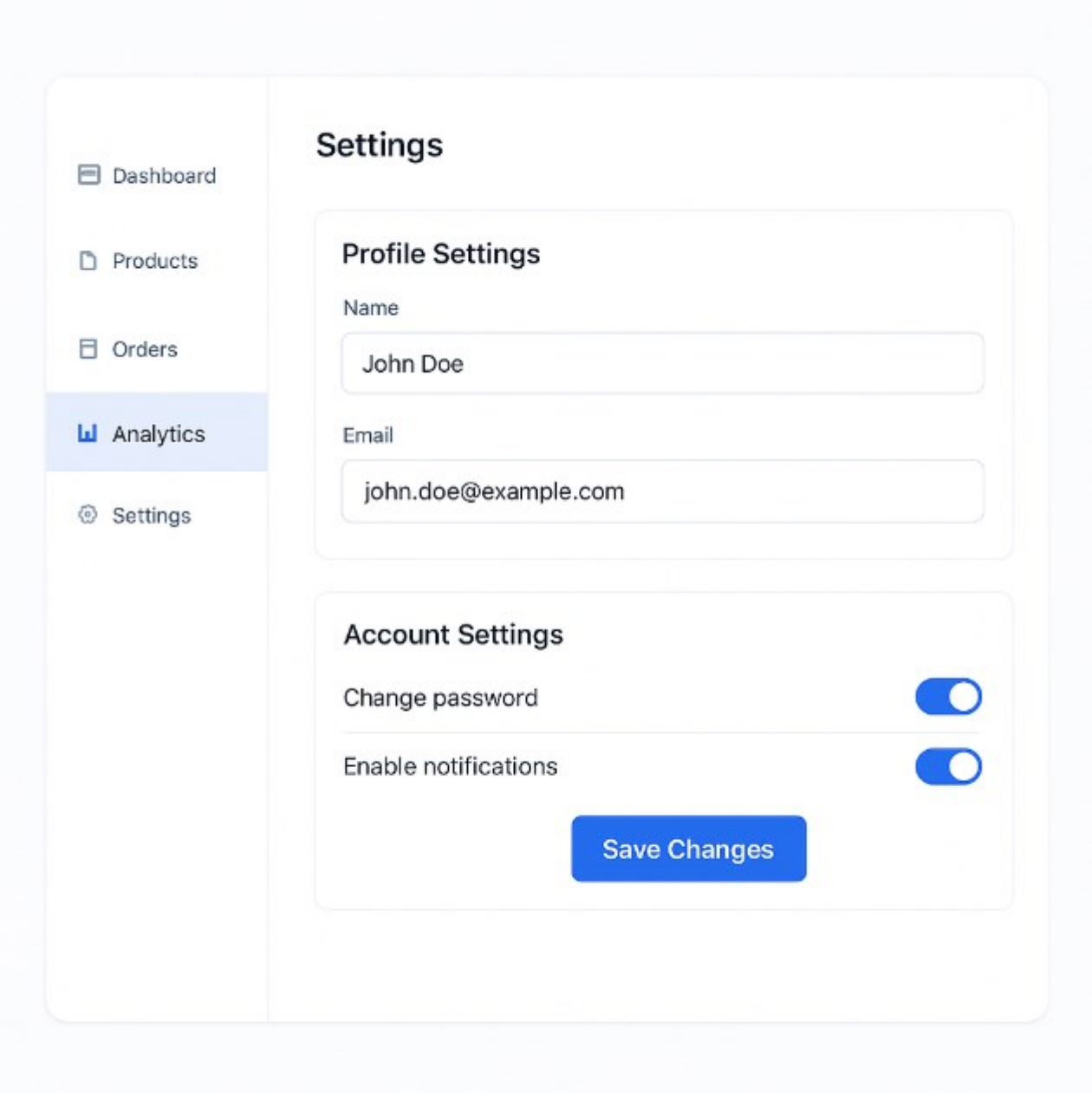

Рисунок 3.5 – Вікно налаштувань особистого кабінету.

Таким чином ми розробили комфортний інтерфейс, якому є можливість переглядати повний аналіз рівня продажів, після чого виконувати відповідні дії, для корекції росту продажів, та вибору асортименту магазину.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Менеджмент безпеки в умовах воєнного стану

Війна – це збройна боротьба між державами (їх коаліціями), або соціальними, етнічними та іншими спільнотами; у переносному розумінні слова – крайня ступінь політичної боротьби, ворожих відносин між певними політичними силами[27].

В сучасних реаліях воєнного стану питання безпеки постає дуже гостро, як для звичайних людей, так і для комерційних підприємств. Кількість ризиків для існування та функціонування кратно збільшилась і потребує постійного моніторингу і контролю, для запобігання або мінімізації втрат.

Основними видами небезпеки на даний момент є економічна та загрози вибухів. При цьому всі види небезпек які були актуальними в мирний час нікуди не зникли. Надзвичайна актуальність безпеки всіх видів зумовлює важливість проблематики її забезпечення[28]. Під час ворожих обстрілів руйнується значна частина інфраструктури, виробничі і офісні будівлі, жилі будинки, транспортні шляхи, відповідно такі процеси як виробництво, логістика і сфера надання послуг мають ризик заповільнитись, або тимчасово призупинити свою діяльність на певній території, або в цілому, якщо пошкоджено критично важливий вузол.

В таких умовах для мінімізації втрат гостро повстає питання менеджменту безпеки. На даний час сучасне цифрове суспільство і комплекс електронних послуг держави формують складне середовище, яке не може існувати без надійної системи інформаційної та кібернетичної безпеки [29]. Таке середовище дозволяє зробити процес менеджменту безпеки ефективнішим і простішим.

В основі процесу менеджменту безпеки стоїть надійна система оповіщення, яка завчасно повідомляє про ворожі атаки. В випадку України це система сирен, яка вмикає звукове сповіщення, при загрозі повітряного удару. Також є мобільний та веб-застосунок «Alarm», який в реальному часі транслює

карту України та показує ні якій території є ризики різного роду атак, та напрямки, я яких ці атаки здійснюються.

Одним з найважливіших аспектів менеджменту безпеки в умовах воєнного стану є ефективна організація укриттів, які дозволяють населенню безпечно перечекати повітряну тривогу та мінімізувати ризики від ракетних ударів чи обстрілів. Цей процес включає не лише визначення та облаштування існуючих захисних споруд, таких як підвали будинків, паркінги, або спеціально збудовані бомбосховища, але й оперативне створення тимчасових укриттів у місцях масового скупчення людей – на зупинках громадського транспорту, біля магазинів чи державних установ. Ключовими вимогами до таких об'єктів є:

- доступність;
- надійність конструкції;
- наявність базових засобів життєзабезпечення (води, аптечки, засобів зв'язку);
- чітка навігація, що дозволяє людям швидко знайти безпечне місце.

Окрім технічних аспектів, надзвичайно важливою є регулярна інформаційно-роз'яснювальна робота серед населення щодо розташування укриттів, правил поводження в них та необхідності неухильного дотримання сигналів повітряної тривоги. Це дозволяє сформувати культуру безпеки та забезпечити

4.2 Заходи, для покращують умови праці розробника

В наш час всі працівники стараються максимізувати працездатність своїх працівників. Під працездатністю людини розуміють можливість її виконувати ротобу з необхідною якістю та в установлений час[30]. Це забезпечує для працедавця кращий заробіток з працівника, і відсутність потреби наймати додаткових працівників. Що стосується самого працівника, то такий підхід покращує умови праці, та як правило позитивно впливає на заробітню плату.

Ефективність та комфорт програмістів на робочому місці залежать від низки взаємопов'язаних факторів, які включають:

- інформаційну адаптацію;
- ергономіку робочого простору;
- якість колективної взаємодії;
- рівень кваліфікації;
- емоційна стійкість.

В кожної людини індивідуальний поріг інформації яку вона може освоїти або опрацювати за певний період часу. Проте це можна натренувати шляхом постійної роботи з інформацією, в наслідок чого людина почне сприймати більше інформації, і опрацьовувати її за менший період часу. За рахунок того що програмісти постійно працюють з інформацією, і постійно вивчають нові технології, в них ця навичка на дуже високому рівні.

Щоб збільшити кількість інформації, з якою можуть працювати програмісти, компанії можуть вживати наступних заходів:

- Регулярні заходи з навчання.
- Проєкти з викликом, які змушують вивчати нові технології.

Правильне оформлення робочого місця це ще один важливий факторів ефективної роботи. За допомогою ергономічно комфортних елементів можна збільшити концетрацію, та мінімізувати шкідливий вплив на здоров’я.

Заходи для досягнення цієї мети:

- Використання якісного монітора.
- Покупка ергономічного крісла.
- Правильно підібрана перефирівя.
- Регулярні перерви та розминки.

Взаємодія в колективі, особливо якщо це колектив який працює на одній локації. В злагодженому колективі працівники ефективніше комунікують, і обмінюються досвідом. Також процес адаптації нових працівників за таких умов проходить легше і плавніше.

Заходи для досягнення цієї мети:

- Організація командоутворюючих заходів (Team Building).
- Вирішення конфліктів та комунікація з HR-відділом.
- Менторські програми.
- Внутрішні системи спілкування, як формального характеру, так і не дуже.

Високий рівень підготовки програміста безпосередньо впливає на його здатність виконувати складні завдання та скорочувати час на простіші. Емоційна стійкість, у свою чергу, є критично важливою в умовах жорстких дедлайнів та високих вимог сучасного ринку праці.

І одним з найважливіших параметрі є рівень компетентності. Добре навчений програміст може виконувати завдання більшої складності, і писати код, який буде ефективніше виконувати своє завдання, порівняно з кодом початківця. Також це сприятливо впливає на час виконання завдань, і підвищує цінність людини на ринку праці.

Заходи для досягнення цих цілей:

- Тренінги.
- Періодичні перевірки рівня кваліфікації
- Системи заохочення та мотивації для навчання.
- Психологічна підтримка та адаптація.

Впровадження цих заходів створює сприятливе середовище для розвитку та ефективної роботи програмістів, що, зрештою, позитивно впливає на загальну продуктивність та успіх компанії.

4.3 Висновок до четвертого розділу

У четвертому розділі кваліфікаційної роботи ґрунтовно розглянуто комплекс питань, що стосуються безпеки життєдіяльності та охорони праці, з особливим акцентом на специфіку роботи в умовах сучасної кризи. Проаналізовано ключові аспекти менеджменту безпеки в умовах воєнного стану, зокрема вплив фізичних загроз на функціонування підприємств. Детально висвітлено роль інформаційної безпеки, та роль інфраструктури для зменшення рівня загрози під час воєнуного стану.

Окремий підрозділ присвячений заходам, що сприяють покращенню умов праці розробників. Розглянуто комплексний вплив ергономіки робочого місця, психологічного клімату в колективі, рівня професійної підготовки та емоційної стійкості на працездатність IT-спеціалістів. Запропоновано практичні рекомендації щодо організації робочого простору, профілактики професійних захворювань, а також значення well-being програм та корпоративної культури для підтримки високої продуктивності та добробуту персоналу.

Всі розглянуті заходи є фундаментальними для створення безпечного, сприятливого та продуктивного середовища для розробки інформаційних систем, мінімізуючи ризики та забезпечуючи стабільність роботи в умовах сучасних викликів.

ВИСНОВОК

В ході виконання кваліфікаційної роботи було розроблено інформаційну систему контролю рівня продаж інтернет магазину. Програму було створено на основі мови програмування C# та SQL. Розроблено основний функціонал та розроблено інтерфейс, що дає змогу комфортно використовувати програму.

В першому розділі кваліфікаційної роботи було освітнього рівня «Бакалавр»:

- описано основні принципи аналізу даних;
- описано ключові метрики та KPI онлайн-продаж;
- розглянуто архітектури зберігання та аналітики даних;
- описано інструменти для розробки програми для аналізу.

В другому розділі кваліфікаційної роботи:

- описано вимоги до проекту;
- обрано інструменти і ПЗ для розробки;
- визначено користувачів та способи взаємодії з системою;
- створено та налаштовано базу даних.

В третьому розділі кваліфікаційної роботи:

- розроблено ключові кодові рішення;
- спроектовано моделі;
- підключено базу даних;
- розроблено сервіси аналітики продажів;
- розроблено Unit-тести.

У розділі «Безпека життєдіяльності, основи охорони праці» було проаналізовано питання менеджменту безпеки в умовах воєнного стану, та розглянуто заходи для покращення умов праці розробника.

ПЕРЕЛІК ДЖЕРЕЛ

1. Kumar, A., & Singh, R. K. (2021). Big data analytics for business intelligence in e-commerce: A review. *Journal of Retailing and Consumer Services*, *58*, 102272.

2. Пушкар, М. С., & Семанюк, В. З. (2009). Розробка систем обліку: Навчальний посібник.

3. Mazhar, S. A., Anjum, R., Anwar, A. I., & Khan, A. A. (2021). Methods of data collection: A fundamental tool of research. *Journal of Integrated Community Health*, *10*(1), 6-10.

4. Kotsiuba, O., & Yashchenko, R. (2024). Evolution of ETL processes in modern data-lake pipelines. *Medical Informatics & Engineering*, 2(1), 45-56.

5. Ridzuan, F., & Zainon, W. M. N. W. (2019). A review on data cleansing methods for big data. *Procedia Computer Science*, *161*, 731-738.

6. Марєнко, В. Ю. Безпека даних в епоху великих даних як стратегічний ресурс країни. *С83 Стратегічні пріоритети розвитку підприємництва, торгівлі та біржової діяльності*, 76.

7. Wang, J., Li, Y., & Chen, S. (2023). Interactive data visualization for business intelligence: A case study of sales analysis. *Journal of Big Data Analytics*, *7*(1), 1-15.

8. Чернега, В. М., & Зелінська, О. В. (2021). Хмарні сховища даних. *Прикладні аспекти сучасних міждисциплінарних досліджень*, 119-121.

9. Patchipala, S. (2025). *The benefits of Delta Lake and Lakehouse architecture*. Proceedings of the Data Innovation Summit.

10. АДАМИК, Оксана Василівна; АДАМИК, Катерина Богданівна. Реляційні бази даних як сучасний стандарт накопичення інформації в комп'ютерній системі бухгалтерського обліку. 2018.

11. Akhter, R., & Sofi, S. A. (2022). Precision agriculture using IoT data analytics and machine learning. *Journal of King Saud University-Computer and Information Sciences*, *34*(8), 5602-5618.

12. Кривонос, О. М., Галіцький, А. В., & Кривонос, М. О. (2023). Проблема вибору мови програмування в якості засобу вивчення ООП.

13. Koul, P., & Raju, V. (2023). Comparative performance of star and snowflake schemas in big-data warehouses. *International Journal of Recent Biotechnology and Applied Technology, 5*(2), 12-21.

14. Говорущенко, Т. О. (2021). Теоретичні та прикладні засади інформаційної технології оцінювання достатності інформації, щодо якості у специфікаціях вимог до програмного забезпечення. *Дисертація на здобуття наукового ступеня доктора технічних наук за спеціальністю*, *5*(06).

15. Karolita, D., McIntosh, J., Kanij, T., & Grundy, J. (2023). Use of personas in requirements engineering: A systematic mapping study. *Information and Software Technology*, *162*, 107264.

16. Pham, H. V., & Nguyen, T. T. (2024). *Defect prediction with content-based features*. *arXiv preprint* arXiv:2409.18365

17. Павлій, М. А. (2023). Розробка кросплатформного ігрового застосунку на платформі Unity.

18. Server, S. Q. L. SQL Server Management Studio.

19. Karolita, D., McIntosh, J., Kanij, T., & Grundy, J. (2023). *Use of personas in requirements engineering: A systematic mapping study*. *Information and Software Technology, 162*, 107264.

20. Зінов'єва, О. Г. (2023). Використання CASE-засобів для проектування інформаційних систем. *Українські студії в європейському контексті*, (7), 220-227.

21. Ahmed, M., & McIntosh, C. (2025). Automated unit test case generation: A systematic literature review. *arXiv*.

22. Sánchez, M. T., & Martínez, J. A. (2021). Integration of disparate data sources for business intelligence: Challenges and solutions. *Information Systems Management*, *38*(2), 125-140.
23. Putra, A., & Wardani, E. (2025). Performance testing on Transparent Data Encryption for SQL Server's reliability and efficiency. *Journal of Big Data, 12*(4), 1-16.
24. Ioannou, P., Georgiadis, K., & Tsiatsis, V. (2023). Performance evaluation of container orchestration tools in edge computing. *Sensors, 23*(8), 4008.
25. Ahmed, M., & McIntosh, C. (2025). Automated unit test case generation: A systematic literature review. *arXiv*.
26. Wang, J., Li, Y., & Chen, S. (2023). Interactive data visualization for business intelligence: A case study of sales analysis. *Journal of Big Data Analytics*, *7*(1), 1-15.
27. Желібо Є.П., Зацарний В.В. Безпека життєдіяльності. Підручник. – К.: Каравела, 2009.
28. Шевчук, Д. (2022). Менеджмент безпеки та його функції в умовах війни.
29. Ніколаєску, І. О., & Шинкарьова, В. С. (2022). Цифровізація освіти як сучасна вимога інформаційного суспільства. *Перспективи та інновації науки»(Серія «Педагогіка», Серія «Психологія», Серія «Медицина»), Вип.№ 2 (7)/редкол.: ІВ Жукова (голова)[та ін.]*, 914-924.
30. Бедрій І.Я., Нечай В.Я. Безпека життєдіяльності. Навчальний посібник. – Львів: Манголія 2006, 2007. 499 с.