КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

інформаційних технологій

(кафедра)

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «БАКАЛАВР»

на тему: «Розробка інформаційної підсистеми управління роботою підприємства»

ВОЛЕНКО ТЕТЯНА ОЛЕКСАНДРІВНА

(прізвище, ім’я та по батькові студента повністю)

Київ 2023 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

_____

_____

(факультет)

інформаційних технологій

_____

_____

(кафедра)

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТ

д.т.н., професор Цюцюра С.В.

_____

«___» _____20___ року

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «БАКАЛАВР»

на тему: «Розробка інформаційної підсистеми управління роботою підприємства»

Виконала:студентка 4-го курсу, групи КН-42с

Спеціальності: 122 «Комп`ютерні науки»

Спеціалізація: _«Інформаційні управляючі системи та технології»

(шифр і назва напряму підготовки, спеціальності)

Воленко Т.О.

(прізвище та ініціали)

Керівник д.т.н., доц. Цюцюра М.І.

(прізвище та ініціали)

Рецензент к.т.н., доц. Баліна О.І.

(прізвище та ініціали)

Київ, 2023 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: __автоматизації і інформаційних технологій

Кафедра:___інформаційних технологій

Освітній рівень: «бакалавр» за ОПП

Спеціальність:_122 «Комп`ютерні науки»

Спеціалізація:__Інформаційні управляючі системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТ
д.т.н., професор Цюцюра С.В.
_____
„___” _____2023 року

З А В Д А Н Н Я
ДО ВИКОНАННЯ АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «БАКАЛАВР»

Воленко Тетяна Олександрівна

(прізвище, ім’я, по батькові)

1. Тема роботи: Розробка інформаційної підсистеми управління роботою підприємства
керівник роботи: Цюцюра Микола Ігорович, д.т.н,
затверджені наказом ректора КНУБА № 1811/2 від « 17» листопада 2022 р.
2. Термін подачі студентом роботи до захисту: 01 червня 2023 .
3. Вихідні дані до роботи _____

4. Зміст пояснювальної записки: Вступ 1. Аналіз та дослідження проблеми. 2. Проєктування інформаційного забезпечення. 3. Практична реалізація. 4. Техніко-економічне обґрунтування розробки підсистеми (Бізнес-план). 5. Охорона праці.

5. Перелік презентаційно-інформаційних слайдів: 1. Інформаційна підсистема управління роботою підприємства 2. Актуальність теми. 3. Об'єкт дослідження . 4. Мета. 5. Завдання для досягнення мети. 6. Вивчення предметної області. 7. Існуючі розробки. 8. Існуючі розробки. 9. Постановка задачі. 10. Постановка задачі. 11. Вимоги до ПК користувача. 12. Інструменти реалізації. 13. Стек «View», класи інтерфейсу користувача. 14. Інтерфейс додатку. 15. Схема даних. 16. Лістингу коду, що відповідає за оновлення даних. 17. Інтерфейс запущеного додатку. 18. Економічна доцільність. 19. Що забеспечує розроблена інформаційна система.

6. Консультанти розділів атестаційної випускної роботи

Розділ	Прізвище, ініціали та посада консультанта, представника комісії	дата	підпис
Техніко-економічне обґрунтування розробки підсистеми (Бізнес-план)	д.т.н. проф. Цюцюра С.В.		
Прийом програмного продукту	к.т.н., доц. Єрукаєв А.В.		

7. Дата видачі завдання: 15 лютого 2023 р.

КАЛЕНДАРНИЙ ПЛАН

Види робіт та їх зміст	Дата виконання
Р. 1. Аналіз та дослідження проблеми	15.04.2023-23.04.2023
Р. 2. Проєктування інформаційного забезпечення	24.04.2023-03.05.2023
Р. 3. Практична реалізація	05.05.2023-15.05.2023
Р. 4. Бізнес план	08.04.2023-14.04.2023
Р. 5. Охорона праці	16.05.2023–22.05.2023
Остаточне оформлення роботи	23.05.2023–24.05.2023
Направлення роботи на рецензування	24.05.2023-28.05.2023
Попередній захист роботи на кафедрі	13.06.2023–14.06.2023

Бакалавр		Воленко Т.О.
	(підпис)	(прізвище та ініціали)
Керівник		Цюцюра М.І.
	(підпис)	(прізвище та ініціали)

АНОТАЦІЯ

Розроблення додатку для забезпечення роботи складу реалізації товарів.

Кваліфікаційна робота містить: 68 сторінок, 5 розділів, 27 рисунків, 7 таблиць та 27 літературних джерел.

Спеціальність: Комп’ютерні науки.

Метою кваліфікаційної роботи є покращення управління підприємством за допомогою впровадження додатку автоматизації робочих процесів складу.

У кваліфікаційній роботі: проведено аналіз області автоматизації, наведено приклади реалізації подібних систем, розроблено вимоги до додатку та його функцій, обґрунтовано бізнес доцільність розробки, розроблено інтерфейс, базу даних та програмну частину додатку, досліджено охорону праці під час роботи програміста.

Результатом виконання кваліфікаційної роботи є повноцінний додаток для управління роботою підприємства.

КЛЮЧОВІ СЛОВА: ДОДАТОК, ІНТЕРФЕЙС КОРИСТУВАЧА, СКЛАД, КЛАС, ФУНКЦІЯ, МЕТОД.

ABSTRACT

Development of an application to ensure the operation of the goods sales warehouse.

The qualifying work contains: 68 pages, 5 sections, 27 figures, 7 tables and 27 literary sources.

Specialty: Computer science

The purpose of the qualification work is to improve enterprise management by introducing an application for automating warehouse workflows.

In the qualification work: an analysis of the field of automation is carried out, examples of the implementation of such systems are given, requirements for the application and its functions are developed, the business feasibility of development is substantiated, the interface, database and the software part of the application are developed, labor protection is investigated during the work of the programmer.

The result of the qualification work is a full-fledged application for managing the work of the enterprise.

KEYWORDS: APPLICATION, USER INTERFACE, COMPOSITION, CLASS, FUNCTION, METHOD.

ЗМІСТ

ВСТУП

Питання управління роботою підприємства є актуальним у всі часи. Оцінка ефективності складу передбачає аналіз якості та кількості складських операцій. Добре структурована робота - запорука успішного розвитку компанії.

Актуальність теми полягає в тому, що ведення складського обліку дозволяє зробити роботу складу «прозорою» і мінімізувати тимчасові витрати на проведення різних складських операцій.

Об'єктом який досліджується в даній роботі є робота малого підприємства яке займається реалізаціє продукції та обліком замовлень і клієнтів.

Метою цієї роботи є створення системи для працівників підприємства. Система буде підтримувати можливості ведення звітності, та збереження даних в хмарі, дані які зберігаються: товари, замовлення, клієнти, категорії товарів.

Для досягнення мети в підсумковій кваліфікаційній роботі слід вирішити наступні завдання:

- Вивчення предметної області;
- Розглянути ряд існуючих розробок для вирішення мети;
- Обрати спосіб здійснення розробки;
- Розробка інформаційної системи;
- Аналіз бізнес доцільності розробки;
- Аналіз охорони праці та вимог безпеки під час розробки;

Щоб досягти поставленої мети буде використовуватися середа розробки Visual Studio 2022 та СУБД Microsoft SQL Server, мова реалізації додатку C#.

Дана кваліфікаційна робота складається зі вступу, п’ятьох розділів та висновка;

В 1 розділі наведено загальну інформацію про важливість подібних систем та наведено короткі характеристики лідируючих систем в цій області.

У 2 розділі було складене технічне завдання на розробку, визначення призначення додатку, загальні вимоги до додатку, вимоги до шаблону додатку, перелік функцій та вимоги до них.

У 3 розділі описано створення логічних та фізичних моделей БД, описано створення інтерфейсу користувача та розроблення функцій.

У 4 розділі описано бізнес модель впровадження та ефективності подібного проектування та розробки.

У 5 розділі описано заходи із охорони праці техніка безпека при роботі в офісі, та загальні вимоги до програміста.

У заключній частині наведено висновки, література яка використовувалась та додатки.

РОЗДІЛ 1. ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Аналітичний огляд існуючих рішень

Незалежно від того, якого розміру чи масштабу має підприємство на ньому завжди відбуватиметься багато дій. Вирішувати занадто багато речей одночасно і не дозволяти роботі ставати хаотичною може бути проблемою. Ось чому підприємствам має сенс почати використовувати програмне забезпечення для керування бізнесом, щоб бути в курсі подій. Сьогодні все більше організацій і компаній інвестують у програмне забезпечення для управління бізнесом, щоб виконувати завдання, прогнозувати ризики та підвищувати загальну ефективність. Таким чином, не буде помилкою сказати, що інструменти управління бізнесом пройшли довгий шлях і постійно покращують ситуацію для будь-кого, хто займається певним бізнесом [6].

Ефективність роботи підприємства полягає у правильності роботи виробництва а й у точності роботи складу. Складування товарів необхідно практично завжди через те, що робота виробництва та транспорту, та й споживання, часто носить циклічний характер. При цьому чим більше підприємство, тим більше складських операцій воно виконує.

Підвищення точності та оперативності обліку товару, ефективне використання складських приміщень, доступ до них та раціональне планування складських операцій однозначно пов'язане з точністю роботи всього складу. Для автоматизації процесу часто вводять складський облік [6].

Для успішної та продуктивної роботи за працівниками складу продукції закріплено такі завдання:

- Керівництво роботами щодо прийому товарно-матеріальних цінностей на склад готової продукції.

- Організація розміщення товарно-матеріальних цінностей з урахуванням найбільш раціонального використання складських площ.
- Раціональне використання персоналу та обладнання при виконанні складських операцій, рівномірний розподіл обсягів робіт серед підлеглих.
- Забезпечення збереження товарно-матеріальних цінностей (відповідно до Договору про повну індивідуальну матеріальну відповідальність), дотримання режимів зберігання, ведення обліку складських операцій.
- Організація та проведення регулярної інвентаризації товарно-матеріальних цінностей з метою контролю залишків товарів на складі.
- Контроль за дотриманням правильності документообігу складської логістики, за своєчасним оформленням та здаванням прибутково-видаткових документів.
- Організація та керівництво відпусткою товарно-матеріальних цінностей, підготовкою вантажів до навантаження, перевіркою цілісності вантажу, проведенням вантажно-розвантажувальних робіт з дотриманням встановлених правил.

Забезпечення щоденної та періодичної звітності перед керівництвом про виконану роботу та залишки готової продукції на складі. Рід діяльності кожного працівника регламентований [7]. Відповідальність по роботі складу готової продукції несе завідувач складом готової продукції. До його безпосередніх обов'язків відноситься:

- Контроль за організацією прийому товарно-матеріальних цінностей на склад готової продукції.
- Організація та контроль за виконанням складських операцій з внутрішнього переміщення продукції між місцями зберігання та в рамках одного складського простору з метою підготовки місця розміщення прийнятого товару.

- Забезпечення розміщення товарно-матеріальних цінностей відповідно до планування приміщення та картограми розміщення продукції у складі.
- Забезпечення збереження товарно-матеріальних цінностей, дотримання режимів зберігання, ведення обліку складських операцій.
- Контроль за дотриманням правил оформлення та здачі прибутково-видаткових документів (внутрішніх накладних, накладних на здачу ТМЦ на ділянку підготовки сировини для дроблення та ін.)
- Організація складських операцій з відпуску товарів, збирання продукції та її вантажно-розвантажувальних робіт з дотриманням встановлених правил.
- Виконання розпоряджень керівника, пов'язаних із потребами підприємства.
- Введення на посаду прийнятих співробітників, інструктаж щодо прийнятих у роботі норм, стандартів та вимог.
- Забезпечення щоденної та періодичної звітності перед керівництвом про виконану роботу та залишки готової продукції на складі
- Надання даних для оформлення табеля обліку робочого часу персоналу складу готової продукції.
- Організація та проведення інвентаризації товарно-матеріальних цінностей;
- Контроль за справністю обладнання, інвентарю, протипожежних засобів;
- Дотримання правил з охорони праці, техніки безпеки, Правил внутрішнього трудового розпорядку, трудової дисципліни;

Перш ніж перейти до постановки задачі та проектування, необхідно визначити загальні стандарти та типи програмного забезпечення по управлінню підприємством.

ProofHub — це комплексне програмне забезпечення для керування бізнесом, яке допомагає компаніям керувати роботою таким чином, щоб усе було організовано в одному місці. Це центральна платформа (Рис 1.1), яка дозволяє командам легко сортувати безлад і візуалізувати свої щоденні завдання.

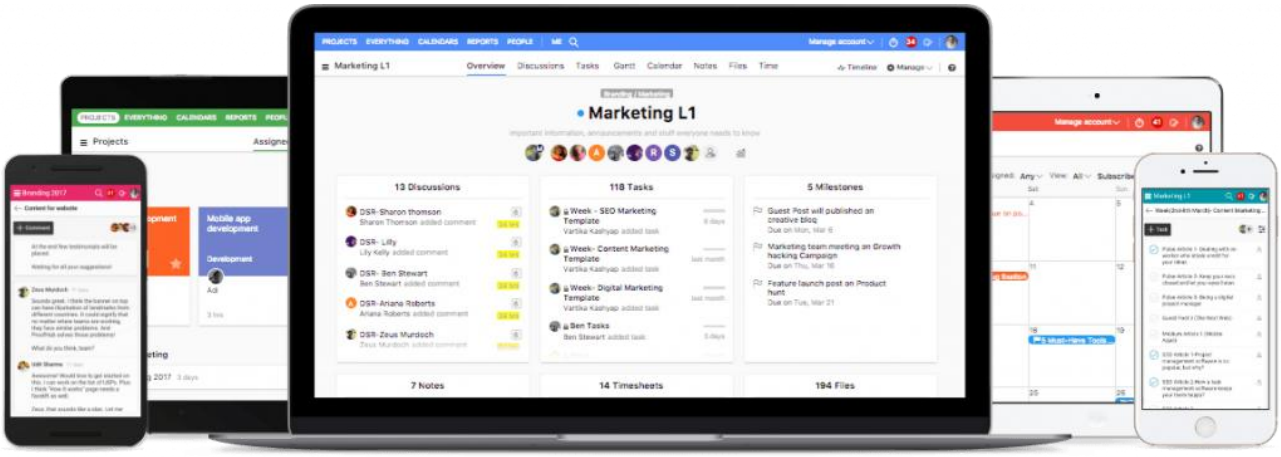


Рисунок 1.1 – Вигляд додатку ProofHub

Програмне забезпечення базується на концепції централізації всіх вимог проекту для простого доступу. ProofHub усуває необхідність у кількох програмах для обміну оновленнями, спільної роботи чи керування завданнями; все можна зробити в одному місці.

Користувач має змогу призначати завдання членам своєї команди, відстежувати їхній прогрес під час їх виконання та запитувати їх, щоб отримати оновлення, коли вони вам потрібні. Користувач може легко спланувати решту тижня та зробити нотатки для зустрічі. Це дозволяє надсилати побажання всім, дозволяючи робити оголошення для всієї компанії. Члени команди можуть навіть миттєво спілкуватися один з одним за допомогою особистого чи групового чату.

Після затвердження остаточних коригувань результатів роботи керівники та їхні підлеглі можуть легко позначати дії від «виконання» до «виконано». ProofHub також дозволяє керувати файлами та ділитися ними

ProofHub пропонує різноманітні функції продуктивності, щоб допомогти компаніям будь-якого розміру покращити результати проектів.

Основні переваги ProofHub:

- Розділ керування завданнями ProofHub відповідає за створення, керування та відстеження завдань у проекті. Завдання для всієї команди можна планувати та легко складати у зручному для візуального перегляду форматі Kanban та діаграмах Ганта.
- Перегляд таблиці в ProofHub організовує всі ваші завдання як динамічна електронна таблиця Excel. Це дуже корисно для тих, хто провів усе своє трудове життя, працюючи з таблицями, тому що він переповнений функціями стовпців. Ви також можете налаштувати перегляд завдань відповідно до ваших унікальних потреб за допомогою спеціальних полів . У режимі перегляду календаря можна переглянути всі завдання, призначені вам протягом тижня, тож ви завжди будете в курсі, коли вам потрібно виконати важливі справи.
- ProofHub також має низку інших корисних інструментів керування завданнями, як-от кінцевий термін виконання завдання, миттєві сповіщення про завдання (у додатку та електронною поштою) і робочі процеси завдань, які полегшують керування завданнями.
- Незалежно від того, чи є ви локально чи віддалено, ProofHub об’єднує вашу команду за допомогою функцій активної співпраці. Розділ обговорень у ProofHub допомагає вам проводити обговорення в реальному часі з членами вашої команди та розділяти їх на основі конкретних тем.
- Розділ нотаток у ProofHub дозволяє швидко занотувати важливу інформацію в одному місці, розділити нотатки на різні розділи, які називаються «блокнотами», і поділитися ними з товаришами по команді.

- За допомогою інструментів розмітки та анотованих коментарів у файлі інструмент перевірки дає змогу надавати детальний відгук. Ваші співавтори можуть бачити коментарі один одного, вносити необхідні зміни та надсилати нові версії на затвердження.
- ProofHub відстежує моделі витрачання часу вашою командою, що допомагає вам не відставати від розкладу. Використовуючи технології реєстрації часу, моніторингу та звітування, ви можете встановити підзвітність часу.
- Таймер ProofHub — це автоматизований інструмент реєстрації часу, який відстежує, скільки часу ви витрачаєте на кожну дію. Ви можете легко запустити таймер, коли ви починаєте завдання, і зупинити його, коли ви закінчите.
- Розклад у ProofHub схожий на електронні таблиці для відображення часу, зареєстрованого вашою командою. Ви можете переглядати чіткі звіти про те, як ваша команда витрачає час, на які завдання витрачається найбільше часу та скільки оплачуваних годин вони витратили.
- ProofHub — це чудове рішення для зберігання файлів, яке розумно керує копіюванням файлів, керуванням версіями та сортуванням в одному зручному місці. Він має добре структуровану файлову систему та 100 ГБ (з можливістю розширення) місця для зберігання всіх файлів вашого проекту.
- ProofHub дозволяє завантажувати всі файли вашого проекту та класифікувати їх у певні папки. Ви також можете прикріплювати файли до модулів спілкування, таких як командний чат, обговорення та завдання, і всі ваші вкладення будуть зручно доступні в розділі «Файли».
- Користувач також може створити численні версії одного файлу в розділі файлів, кожна з яких має свою історію версій. Завантажуйте

нові версії, повертайтеся до старішої версії та легко діліться остаточною роботою зі своєю командою.

- Кожен власник бізнесу хоче знати, чим займається його або її команда щодня, не перериваючи та не перериваючи роботу. ProofHub дозволяє відстежувати повсякденну діяльність вашої команди та безперешкодно просувати її.
- Звіти про проект у ProofHub допомагають отримати детальне уявлення про просування проекту. Ви дізнаєтесь, якщо члену команди потрібен невеликий поштовх або якщо проект відстає, перш ніж усе вийде з-під контролю.
- Засіб відстеження активності ProofHub дозволяє вам бачити всі модифікації та оновлення, внесені у ваші проекти. Він інформує вас про кожну дію, виконану членом команди. Отже, якщо хтось із членів вашої команди виконає завдання, залишить коментар, досягне віхи, зробить нотатку тощо, ви отримаєте сповіщення автоматично.

Flodesk - це служба електронного маркетингу для малого бізнесу, що швидко розвивається. Вона зосереджена на дизайні, що дозволяє власникам малого бізнесу створювати красиві електронні листи та демонструвати свій бренд (Рис 1.2).

Flodesk пропонує робочі процеси, які дозволяють користувачам автоматизувати доставку провідних магнітів, вітальні послідовності тощо за допомогою простого у використанні візуального конструктора.

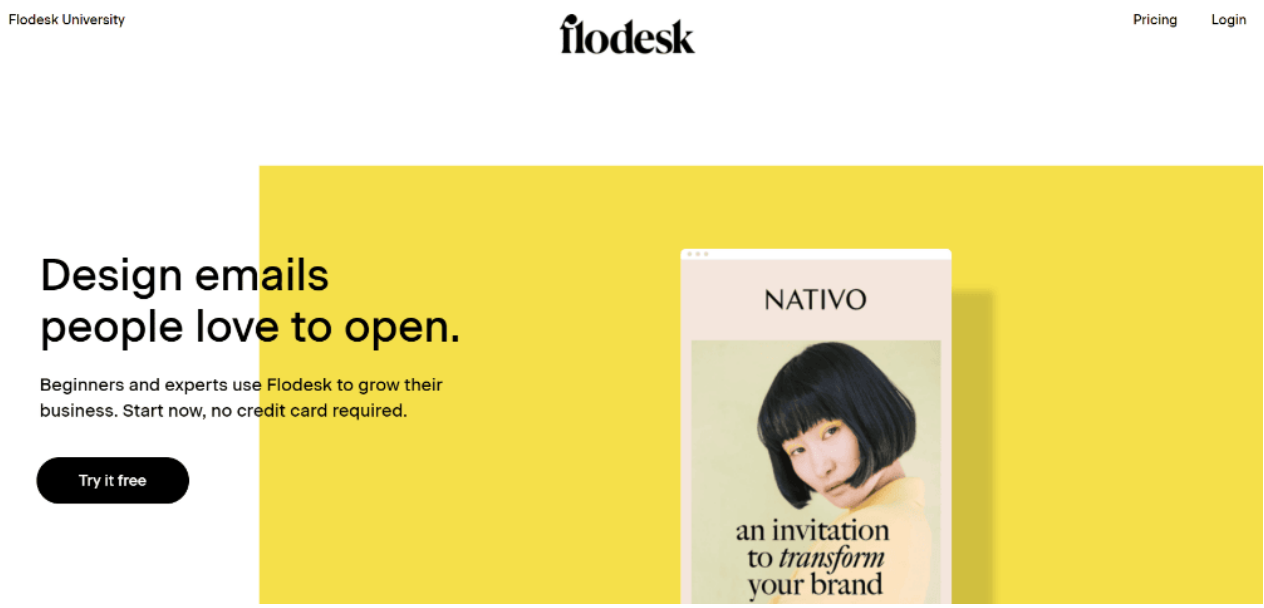


Рисунок 1.2 – Вигляд додатку Flodesk

Ключові особливості:

- Фіксована ціна 38 доларів на місяць, незалежно від розміру списку.
- Необмежена кількість підписок і необмежену кількість відправок електронної пошти.
- Конструктор електронних листів із функцією перетягування та скидання з блоками макета.
- Створювання форми реєстрації електронною поштою навіть без веб-сайту.
- Створювання автоматизованих послідовних електронних листів.

Timely - Елементарно знати, як ваш бізнес витрачає час, незалежно від того, виставляєте ви за це рахунок чи ні. Додаток своєчасно оптимізує весь процес відстеження робочого часу, автоматично записуючи все, над чим працює ваша команда (Рис 1.3). Це значно скорочує накладні витрати на управління часом, одночасно покращуючи звітність і точність виставлення рахунків.

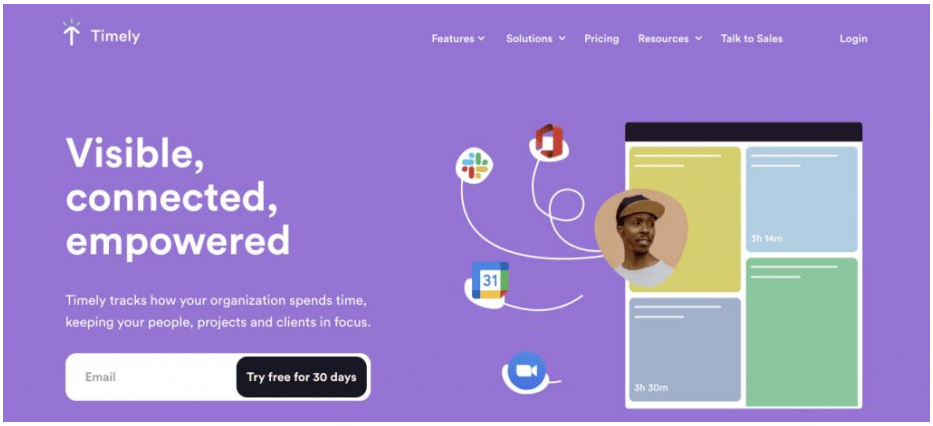


Рисунок 1.3 – Вигляд додатку Flodesk

Головні особливості:

- Автоматичний відлік часу.
- Панелі проектів у реальному часі.
- Продумана, зручна для клієнта звітність.
- Погодинна ставка, потужність і понаднормова робота.

HubSpot — це комплексне CRM-рішення для окремих осіб, малого бізнесу та підприємств (Рис 1.4). Хоча вони спеціалізуються на програмному забезпеченні електронного маркетингу, HubSpot також надає набір інструментів для маркетингу, продажів, обслуговування клієнтів і роботи від робочих процесів електронної пошти до спеціального конструктора веб-сайтів для керування всім вашим бізнесом в одному місці. Оскільки це комплексне рішення, усі ваші команди можуть працювати з базою даних HubSpot і спільно працювати над проектами, кампаніями та завданнями.

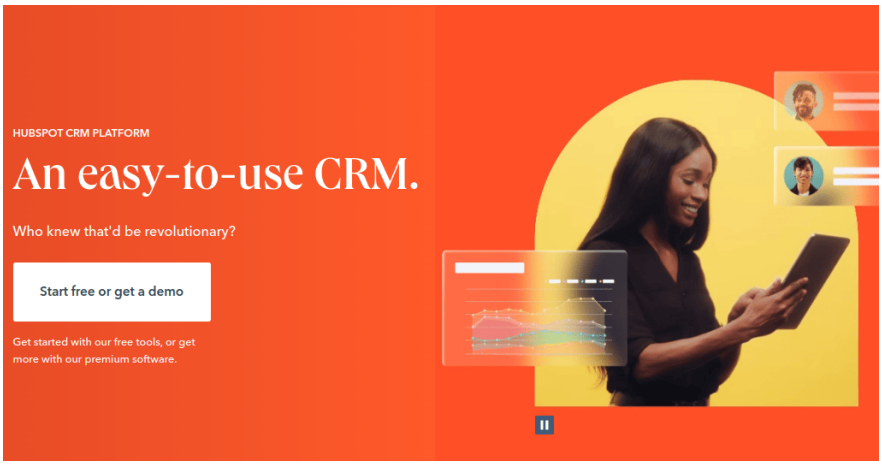


Рисунок 1.4 – Вигляд додатку Flodesk

Головні особливості:

- Синхронізація даних між усіма командами;
- Конструктор власних сторінок і веб-сайтів;
- Робочі процеси та форми електронного маркетингу;
- Системи обслуговування клієнтів;
- Звіти про продажі та аналітика;

iBE.net - програмне забезпечення для управління бізнесом/система управління компанією, iBE.net є відповідним вибором для компаній середнього розміру, оскільки пропонує відстеження витрат, звіти про рахунки-фактури, підтримку CRM разом із простою інтеграцією деталей проекту. Він широко використовується в консалтингу, маркетингу, менеджменті та інших технічних галузях (Рис 1.5).

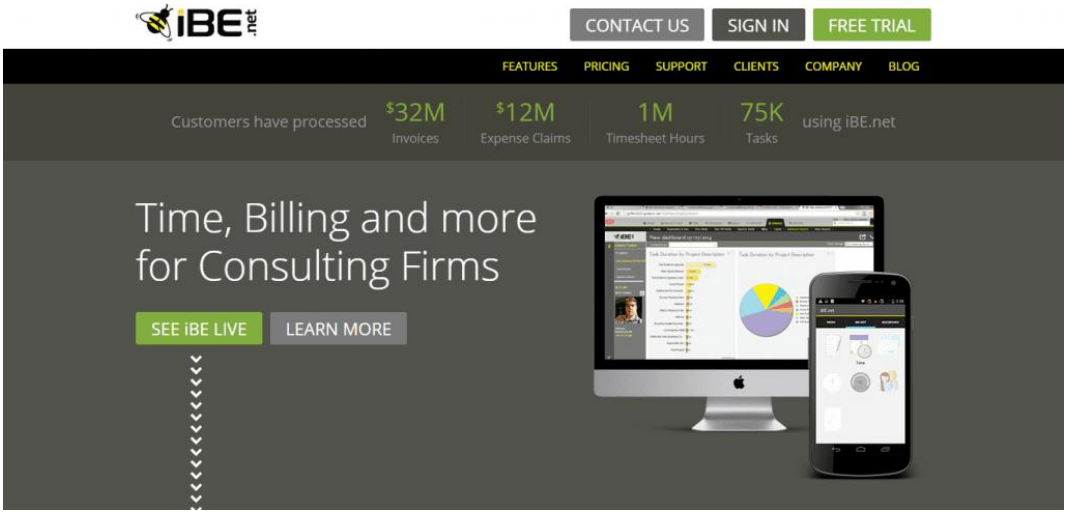


Рисунок 1.5 – Вигляд додатку Flodesk

Головні особливості:

- Налаштування власної робочої панелі;
- Використання усіх звітів;
- Створення рахунків-фактур за лічені секунди;
- Візуалізація, аналізування за допомогою цінною інформації.

РОЗДІЛ 2. ПОСТАНОВКА ЗАДАЧІ

2.1 Постановка задачі на розробку системи

Призначенням цього додатку є зручне управління клієнтами та звітності продукції і її процесу її реалізації. Вона поєднує у собі 2 типи систем:

- Одно-сторінковий додаток (Сайт не перезавантажується під час переходу по сторінками);
- Інформаційний ресурс.

Метою додатку є покращення інвентаризації та прозорості в реалізації продукції. Користувач зможуть за допомогою додатка отримати всю інформацію що до замовлень, клієнтів, товарів.

Цільова аудиторія використання додатку це малий бізнес та стартапи:

- Клієнти, Замовники;
- Потенційні Клієнти, Замовники;

- Потенційні працівники.

Завдання додатку містить наступні частини:

1) Інформаційну.

Додаток буде надавати користувачам доступ до інформації:

- про категорії товарів;
- про наявні товари;
- про замовлення;
- про клієнтів;

2) Навчальну.

Додаток є унікальним у своїй сфері, розробка подібного проекту покращить навички програмування.

Стилістичне оформлення додатку має відповідати таким вимогам:

- Дизайн повинен бути консервативним – використовувати неяскраві, пастельні кольори і тони;
- Дизайн має бути лаконічним, стильним та сучасним.
- Основними шрифтовими гарнітурами є лише гарнітури з стандартних «безпечних» шрифтів.
- Шрифти, які використовуються для оформлення графічних елементів не повинні суперечити загальному стилю.
- Розмір шрифтів повинен забезпечувати зручність сприйняття тексту при мінімально допустимому розмірі екрану.

Сайт повинен забезпечувати коректне та безпечно зберігати данні такі як:

- Інформацію що до клієнтів;
- Інформацію що до продукції;
- Інформацію що до замовлень;

Дизайн повинен правильно відображатися для будь-якого розширення монітору Пк.

У системі управління сайтом має бути передбачений механізм резервного копіювання структури, файлів та вмісту бази даних.

Інтерфейс додатку повинен забезпечувати наочне, інтуїтивно зрозуміле представлення структури розміщеної на ньому інформації, швидкий і логічний перехід до вікон.

Основними функціями системи є:

- Додання категорій товарів;
- Додавання товарів;
- Додання клієнтів;
- Редагування та видалення інформації про товари;
- Редагування та видалення інформації про категорії;
- Редагування та видалення інформації про клієнтів;
- Облік роботи;

Вимоги для додатку:

- Повинен в автоматичному режимі оновлювати інформацію у базі даних;
- Повинен мати можливість на доопрацювання;
- Має бути надійним, та захищати інформацію з бази даних;
- Має бути простим в використанні.

Таблиця 2.1 Вимоги до Пк користувача

Вимоги для персонального комп’ютера клієнта
Процесор Intel pentium 3 1GHz
Не менше 1 ram оперативної пам’яті;
OS Windows 10
Net framework 4.6

Вимоги надійності додатку:

- Додаток повинен функціонувати цілодобово;

- Повинен бути стійким до навантажень та витримувати більше 1000 записів;
- Повинен мати можливість на створення резервних копій;
- Для користування додатком не потрібно мати спеціальних навичок.

РОЗДІЛ 3. МЕТОДИ ВИРІШЕННЯ ЗАДАЧІ

3.1 Інструментальні засоби реалізації дипломного проекту

Для розробки інформаційної системи обрано мову програмування C# та як система бази даних буде використовуватись СУБД Microsoft SQL Server.

Для реалізації додатку обрано інтегроване середовище розробки Microsoft Visual Studio 2022.

C# - сучасна мова програмування загального призначення, яку можна використовувати для виконання широкого кола завдань і завдань, які охоплюють різні професії. C# в основному використовується на фреймворку Windows .NET, хоча він може бути застосований до платформи з відкритим

кодом. Ця дуже універсальна мова програмування є об'єктно-орієнтованою мовою програмування (ООП) і порівняно новою для гри, але надійним задоволенням натовпу [1].

Як і інші мови програмування загального призначення, C# може бути використаний для створення ряду різних програм і додатків: мобільних додатків, настільних додатків, хмарних сервісів, веб-сайтів, корпоративного програмного забезпечення та ігор. Багато і багато ігор. Хоча C# надзвичайно універсальний, є три області, в яких він найчастіше використовується.

C# був створений Microsoft для Microsoft, тому легко зрозуміти, чому він найбільш широко використовується для розробки настільних додатків Windows. Програми C# вимагають фреймворку Windows .NET для того, щоб функціонувати на висоті, тому найсильнішим випадком використання для цієї мови є розробка додатків і програм, специфічних для архітектури платформи Microsoft.

Мабуть, найбільшою перевагою є те, скільки часу можна заощадити, використовуючи C # замість іншої мови програмування. Будучи тим, що C # статично друкується і легко читається, користувачі можуть розраховувати витрачати менше часу на прочищення своїх скриптів для крихітних помилок, які порушують функцію програми.

C# також підкреслює простоту та ефективність, тому програмісти можуть витрачати менше часу на написання складних стеків коду, які неодноразово використовуються протягом проекту. Завершіть все це з великим банком пам'яті, і у вас є ефективна в часі мова, яка може легко скоротити робочі години і допомогти вам вкластися в стислі терміни.

C# - мова програмування, яка надзвичайно масштабована і проста в обслуговуванні. Через строгий характер того, як статичні коди повинні бути написані, програми C# надійно узгоджені, що робить їх набагато легше регулювати і підтримувати, ніж програми, які написані з використанням інших мов [2].

Якщо вам коли-небудь потрібно повернутися до старого проекту, написаного на C #, вам буде приємно виявити, що, хоча ваші процеси, можливо, змінилися протягом багатьох років, ваш стек C# залишився колишнім на всій лінії коду. Є місце для всього і все на своїх місцях.

У світі кодування та програмування важливість корисної спільноти, від якої ви можете залежати, просто не може бути завищена. Мови програмування не є платформою або службою з виділеною лінією допомоги або зручною підтримкою IT. Програмісти повинні покладатися на підтримку інших в тій же області, які випробували ті ж блокпости і розчарування.

Одну з таких спільнот корисних експертів з програмування можна знайти на StackOverflow. Оскільки цей сайт Q&A був побудований на C#, не дивно, що розробники C# складають величезну частину спільноти, де ви можете піти, щоб запитати, відповісти, мозковий штурм або вентиляцію.

Якщо ви віддаєте перевагу співпраці з однодумцями віч-на-віч, C# також має розгалужену спільноту на Meetup.com, де члени можуть приєднатися як онлайн, так і IRL-дискусії, які заплановані навмання або на послідовній основі.

C# повністю об'єктно-орієнтований, що є рідкісною характеристикою для мови програмування. Багато з найпоширеніших мов до певної міри включають об'єктну орієнтацію, але дуже мало хто досяг величини C #, не втрачаючи прихильності з боку людей.

Є багато різних переваг для об'єктно-орієнтованого програмування (або ООП), таких як ефективність і гнучкість, щоб назвати кілька. Деякі розробники, які не знайомі з ООП, можуть відчувати себе трохи неохоче вибирати нову мову з таким важким акцентом на ній, але не хвилюйтеся: розуміння об'єктно-орієнтованого програмування не все так складно.

Microsoft SQL Server - реляційна система управління базами даних (RDBMS). Його основною функцією є зберігання та отримання даних, запитаних іншими програмами. Він підтримує різні програми обробки

транзакцій, бізнес-аналітики та аналітики, як правило, в корпоративних IT-середовищах [4].

Оригінальний код SQL Server був розроблений в 1980-х роках колишньою Sybase Inc., яка зараз належить SAP. У 1988 році Microsoft створила SQL Server для OS/2 як спільні зусилля між Sybase, Microsoft і Ashton- Tate. Партнерство закінчилося в 1990 році, і Microsoft зберегла назву SQL Server. Сьогодні SQL Server доступний на 64-розрядних Windows, Linux і платформі Azure Cloud. SQL Server - RDBMS. Microsoft і Sybase випустили версію 1.0 в 1989 році.

Ештон-Тейт відійшла після цього, але Microsoft і Sybase продовжували партнерство до 1994 року, коли Microsoft взяла на себе всю розробку і маркетинг SQL Server для власних операційних систем. За рік до того, коли відносини Sybase почали розпадатися, Microsoft також зробила програмне забезпечення доступним на нещодавно випущеній Windows NT після зміни 16-бітної бази коду OS/2 для створення 32-бітної реалізації з додатковими функціями; він зосередився на розвитку коду Windows. У 1996 році Sybase перейменувала свою версію на Adaptive Server Enterprise, залишивши назву SQL Server на Microsoft.

SQL Server Management Studio надає графічний інтерфейс користувача адміністрування. Transact-SQL - процедурна мова, що зберігається і виконується в системі управління базами даних SQL Server.

Microsoft пропонує SQL Server у чотирьох первинних виданнях, які надають різні рівні пакетних сервісів. Повнофункціональне видання Developer є безкоштовним - це видання Express, яке може використовуватися для запуску невеликих баз даних з об'ємом дискового простору до 10 ГБ. Видання розробника не ліцензоване для використання у виробництві. Великі програми, які потребують підтримки виробничого рівня, ліцензовані як Enterprise edition. Видання Standard має зменшений набір функцій і обмежену масштабованість, обмежуючи кількість ядер процесора, які він може використовувати, і розмір пам'яті. Через посилення конкуренції, наприкінці

2016 року Microsoft зробила функції Enterprise доступними для Standard Edition. Вони включали In-Memory OLTP, PolyBase, індекси columnstore, розбиття, стиснення даних та можливості захоплення даних.

Microsoft SQL Server - реляційна система управління базами даних. Як сервер бази даних, який зберігає та отримує дані за запитом інших програмних програм на одному комп'ютері або віддаленому комп'ютері за допомогою моделі клієнт-сервер. Microsoft надає API для доступу до SQL Server через Інтернет як веб-сервіс. RDBMS робить набагато більше, ніж отримати дані для клієнтських додатків. Внутрішні функції, такі як управління буферами, забезпечують доступ до найбільш доступних даних у найшвидшому вигляді доступного сховища для швидкого доступу.

SQL Server заснований на реляційній моделі, а також забезпечує посилальну цілісність між об'єктами для підтримки послідовності даних. Як і в інших реляційних базах даних, для підтримки цілісності реалізуються принципи атомності, узгодженості, ізоляції транзакцій і довговічності, разом відомі як ACID-властивості.

Додатки використовують SQL Server за допомогою багатьох інтерфейсів. Інтерфейс ODBC забезпечує високорівневий SQL-інтерфейс, який дозволяє користувачам вбудовувати виклики баз даних у такі програми, як Microsoft Excel. Додатки Java використовують драйвер JDBC, щоб дозволити їм отримати доступ до баз даних за допомогою SQL. Розробники додатків використовують інтерфейси програмування додатків (API) для вбудовування SQL-операторів у свої програми, які можуть бути написані на C, Java та Python, наприклад. Рядки бази даних можна отримувати по одному або партіями або масивами. Microsoft Visual Studio включає нативну підтримку Microsoft SQL Server. Visual Studio включає дизайнера даних для створення, перегляду або редагування схем баз даних графічно. Запити також можуть бути створені візуально.

Інтегроване середовище розробки (IDE) — комплексне програмне рішення для розробки програмного забезпечення. Зазвичай, складається з

редактора початкового коду, інструментів для автоматизації складання та налагодження програм. Більшість сучасних середовищ розробки мають можливість автодоповнення коду [9].

Вибір середовища для розробки, що виходять за рамки базових і прикладних рішень, є ключовим кроком у загальному технологічному процесі розробки програми. Якщо вибір технології зроблений правильно, обрані інструменти та рішення повинні пройти через весь процес розробки та привести до вирішення поставленої задачі. Якщо вибором є не коректне середовище, це може привести до втрати ресурсів, незадоволеності клієнтів і труднощів в розробці програми.

Вибором є середовище Microsoft Visual Studio, що містить інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Даний продукт дозволяє розробляти як консольні додатки, так і додатки з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-додатки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, підтримуваних Microsoft Windows, Windows Mobile, Windows CE, NET Framework, NET Compact Framework і Microsoft Silverlight.

Visual Studio містить редактор вихідного коду з підтримкою технології Intelli Sense і можливістю найпростішого рефакторинга коду. Вбудований відладчик може працювати як відладчик рівня вихідного коду, так і як відладчик машинного рівня. Решта вбудовувані інструменти містить редактор форм для спрощення створення графічного інтерфейсу програми, веб-редактор, дизайнер класів і дизайнер схеми бази даних. Visual Studio дозволяє створювати й підключати сторонні додатки (плагіни) для розширення функціональності практично на кожному рівні, включаючи додавання підтримки систем контролю версій вихідного коду (як наприклад, Subversion і Visual SourceSafe), додавання нових наборів інструментів (наприклад, для редагування та візуального проектування коду на предметно-орієнтованих мовах програмування або інструментів для інших аспектів

циклу розробки програмного забезпечення (наприклад, клієнт Team Explorer для роботи з Team Foundation Server). Visual Studio 2010 (кодове ім'я Hawaii, для Ultimate - Rosario; внутрішня версія 10.0) - випущена 12 квітня 2010 разом с. NET Framework 4.0. Visual Studio включає підтримку мов C # 4.0 і Visual Basic. NET 10.0, а також мови F #, який був відсутній в попередніх версіях.

В цілому стратегія Visual Studio. NET, націлена на реалізацію концепції. NET внаслідок досягнення максимальної продуктивності, включає три напрями:

1. Забезпечення інтуїтивно зрозумілою, розширюваної, уніфікованого середовища для всіх мов, конструкторів та інструментальних засобів.
2. Надання розробникам набору модернізованих співзалежних мов. NET, відповідних наявними у розробників навичкам (про підтримку мов програмування в Visual Studio. NET ми розповімо в цьому огляді).
3. Надання високопродуктивних інструментальних засобів для всіх етапів життєвого циклу розробки – від визначення вимог і планування до подальшого супроводу продукту (ці інструментальні кошти ми також розглянемо в даному огляді).

Для створення клієнтських додатків у Visual Studio. NET використовуються Windows-форми. Як і у випадку з Web-формами, підтримку Windows-форм забезпечує ядро. NET, а Visual Studio. NET лише надає кошти, полегшують розробку програм цього класу на всіх підтримуваних мовами програмування.

3.2 Розробка інтерфейсу користувача

Інтерфейс користувача - це вигляд програми front-end, до якого користувач взаємодіє з метою використання програмного забезпечення.

Програмне забезпечення стає більш популярним, якщо його інтерфейс користувача:

- Привабливий;
- Швидкий в дії;
- Інтуїтивно зрозумілий;
- Узгодженість на всіх екранах інтерфейсу.

Існує два типи інтерфейсу користувача:

1. Інтерфейс командного рядка: інтерфейс командного рядка забезпечує командний рядок, де користувач вводить команду і подає систему. Користувачеві необхідно запам'ятати синтаксис команди і її використання.
2. Графічний інтерфейс користувача: графічний інтерфейс користувача забезпечує простий інтерактивний інтерфейс для взаємодії з системою. Графічний інтерфейс може бути комбінацією як апаратного, так і програмного забезпечення. Використовуючи графічний інтерфейс, користувач інтерпретує програмне забезпечення.

В випадку цього проекту буде використано другий тип інтерфейсу а саме «Графічний інтерфейс», таким чином інтерфейс стане простішим та інтуїтивно зрозумілішим [16].

Для успішного конструювання та розробки інтерфейсу використаємо вже зарекомендовані золоті правила, запропонавані Theo Mandel, яких необхідно дотримуватися при оформленні інтерфейсу.

1. Надати користувачеві контроль:
    - Визначте режими взаємодії таким чином, щоб не змушувати користувача до непотрібних або небажаних дій: користувач повинен бути в змозі легко увійти і вийти з режиму з невеликими або без зусиль.
    - Забезпечити гнучку взаємодію: різні люди будуть використовувати різні механізми взаємодії, деякі можуть

використовувати клавіатурні команди, деякі можуть використовувати мишу, деякі можуть використовувати сенсорний екран і т.д., отже, всі механізми взаємодії повинні бути забезпечені.

- Дозволити взаємодії з користувачем бути переривчастим і незмінним: коли користувач робить послідовність дій, користувач повинен мати можливість перервати послідовність, щоб зробити якусь іншу роботу без втрати виконаної роботи. Користувач також повинен мати можливість скасувати операцію.
- Оптимізувати взаємодію як прогрес рівня кваліфікації та дозволити налаштувати взаємодію: Просунутий або висококваліфікований користувач повинен мати можливість налаштувати інтерфейс, як хоче користувач, що дозволяє різні механізми взаємодії, щоб користувач не відчував себе нудно, використовуючи той же механізм взаємодії.
- Приховати технічні внутрішні дані від випадкових користувачів: користувач не повинен знати про внутрішні технічні деталі системи. Він повинен взаємодіяти з інтерфейсом тільки для того, щоб виконувати свою роботу.
- Дизайн для безпосередньої взаємодії з об'єктами, які з'являються на екрані: користувач повинен мати можливість використовувати об'єкти і маніпулювати об'єктами, які присутні на екрані для виконання необхідного завдання. Таким чином, користувач відчуває себе легко контролювати екран.

2. Зменшити навантаження на пам'ять користувача:
    - Зниження попиту на короткочасну пам'ять: коли користувачі беруть участь у деяких складних завданнях, попит на короткочасну пам'ять є значним. Так інтерфейс повинен бути

розроблений таким чином, щоб зменшити запам'ятовування раніше зроблених дій, заданих входами і результатами.

- Встановити значущі значення за замовчуванням: Завжди початковий набір типових значень повинен бути наданий середньому користувачеві, якщо користувачеві потрібно додати деякі нові функції, то він повинен мати можливість додавати необхідні функції.
- Визначте інтуїтивно зрозумілі ярлики: Mnemonics повинна використовуватися користувачем. Mnemonics означає комбінації клавіш для виконання деяких дій на екрані.
- Візуальне розташування інтерфейсу має базуватися на реальній метафорі: Все, що ви представляєте на екрані, якщо це метафора реальної сутності, то користувачі легко зрозуміють.
- Розкривати інформацію прогресивно: інтерфейс повинен бути організований ієрархічно, тобто на головному екрані інформація про завдання, об'єкт або якусь поведінку повинна бути представлена спочатку на високому рівні абстракції.

3. Зробити інтерфейс послідовним:
   - Дозволити користувачеві помістити поточну задачу в значущий контекст: багато інтерфейсів мають десятки екранів. Тому важливо послідовно надавати показники, щоб користувач знав про виконану роботу. Користувач також повинен знати, з якої сторінки перейшов на поточну сторінку і з поточної сторінки, куди можна перейти.
   - Підтримувати узгодженість у сімействі додатків: розробка деяких програм повинна слідувати і впроваджувати той же дизайн, правила, щоб узгодженість підтримувалася серед додатків.
   - Якщо минулі інтерактивні моделі створили очікування користувачів не вносять змін, якщо немає вагомої причини.

В цілому, дизайн інтерфейсу користувача є ключовим компонентом програмної інженерії, оскільки він може мати значний вплив на зручність використання, ефективність та досвід користувача програми. Інженери-програмісти повинні дотримуватися найкращих практик та принципів проектування для створення інтерфейсів, орієнтованих на користувача, послідовних, простих та доступних [15].

Керуючись загальноприйнятими нормами реалізації інтерфейсу та дотримуючись прозорості в розробці додатку, розмістимо кожну форму інтерфейсу в одному стеку «View» (Рис 3.1), таким чином покращимо зручність розробки та можливих майбутніх оновлень додатку.

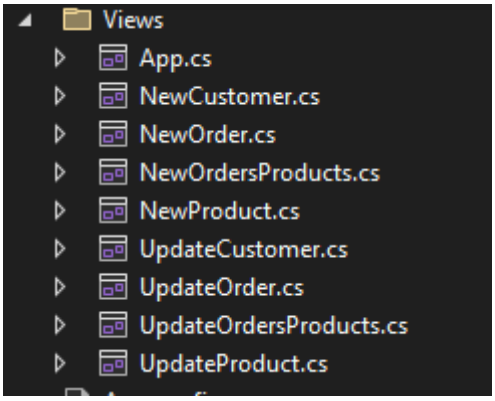


Рисунок 3.1 – Стек «View»

Головним інструментом в реалізації інтерфейсу буде елемент «TabControl», за допомогою цього елемента наше головне меню буде реалізовано вкладками, таким чином ми мінімізуємо відкриття нових вікон на робочому столі і зведемо це майже до мінімуму. Звісно вікна будуть присутні але вони будуть використовуватись для дій які потребують фокусування користувача, наприклад додання або редагування записів.

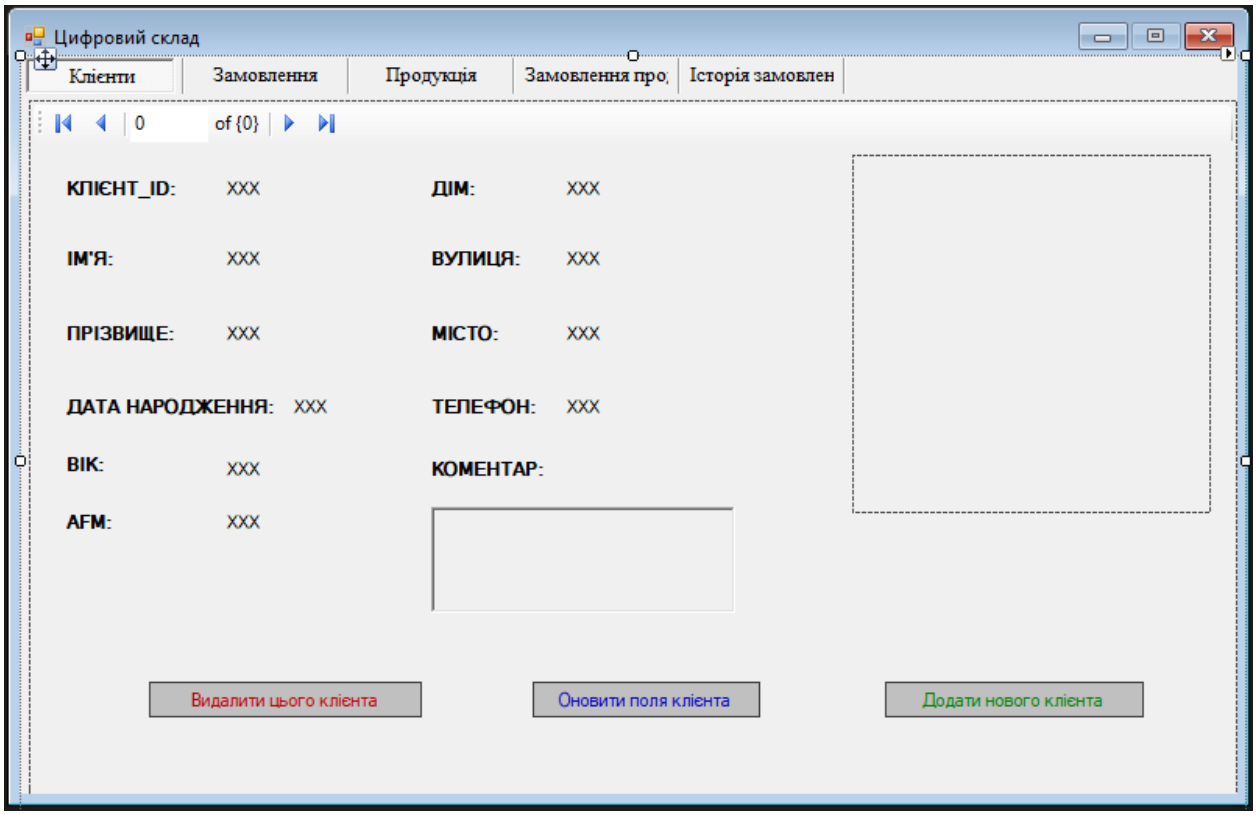


Рисунок 3.2 – Представлення інтерфейсу користувача

Як бачимо з малюнка 3.2, елементи «label» відзначені «XXX», будуть використовуватись для майбутнього їх наповнення відповідною інформацією з бази даних.

Кнопки інтерфейсу на кожній вкладці знаходяться в одному місці та мають однаковий стиль та розмір, що забезпечить інтуїтивно зрозуміле користування додатком та збільшить ефективність його застосування.

Після натискань кнопок відкриваються діалогові вікна, що розташовуються в центрі екрану та одразу фокусуються користувача на тій дії яку він обрав.

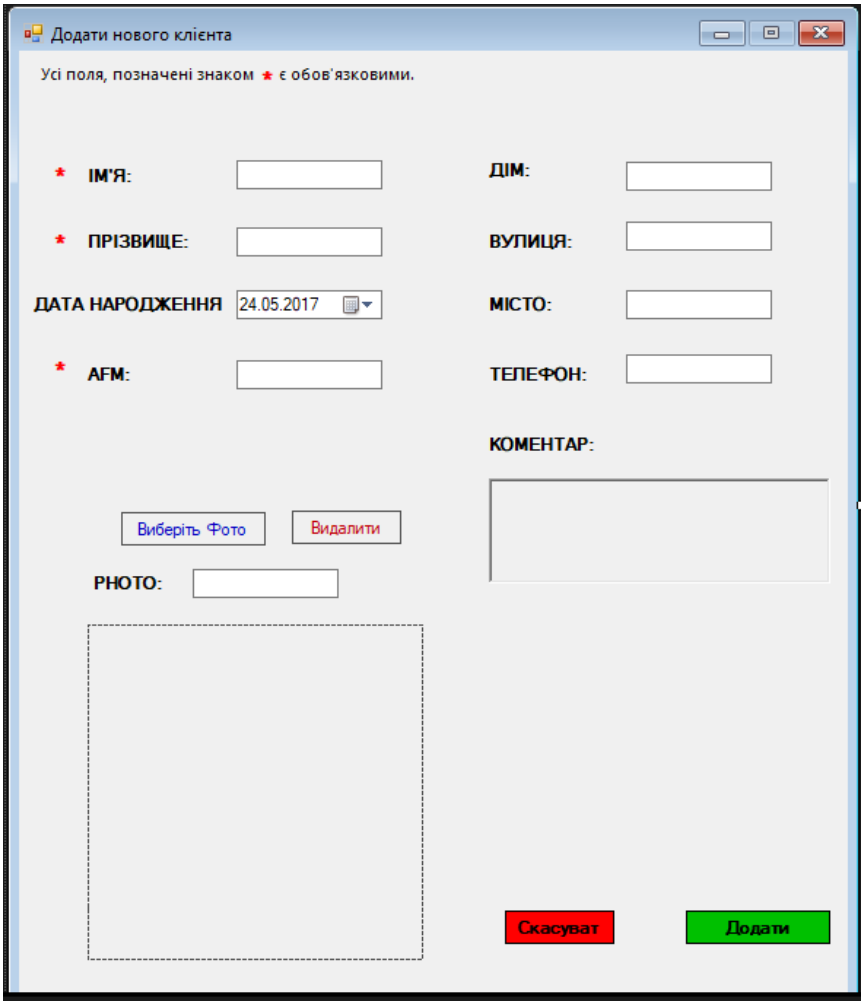


Рисунок 3.3 – Представлення інтерфейсу додання клієнта

Після натискання кнопки додати запис, відкриється нове вікно інтерфейсу, яке дасть нам змогу реалізувати ту чи іншу дію. На малюнку 3.4 можемо спостерігати, що в залежності від типу внесеної інформації використовується той елемент інтерфейсу який найбільш буде зручний. Наприклад поле для фото, або для коментаря використовується велике поле так як коментар може бути великого розміру. Для введення дати народження теж використовується не стандартний «textbox» а «datetimepicker» який спростить користувачеві взаємодію [9].

Також поля які не можуть залишатись пустими відмітимо червоною зірочкою, щоб користувач під час взаємодії точно знав які поля необхідно першочергово наповнити та не можна залишити їх пустими.

Підсумком стає інтуїтивно зрозумілий та простий інтерфейс, який дає змогу легко та ефективно взаємодіяти з додатком, демонстрація інтерфейсу в працюючому форматі представлена на малюнку 3.4.

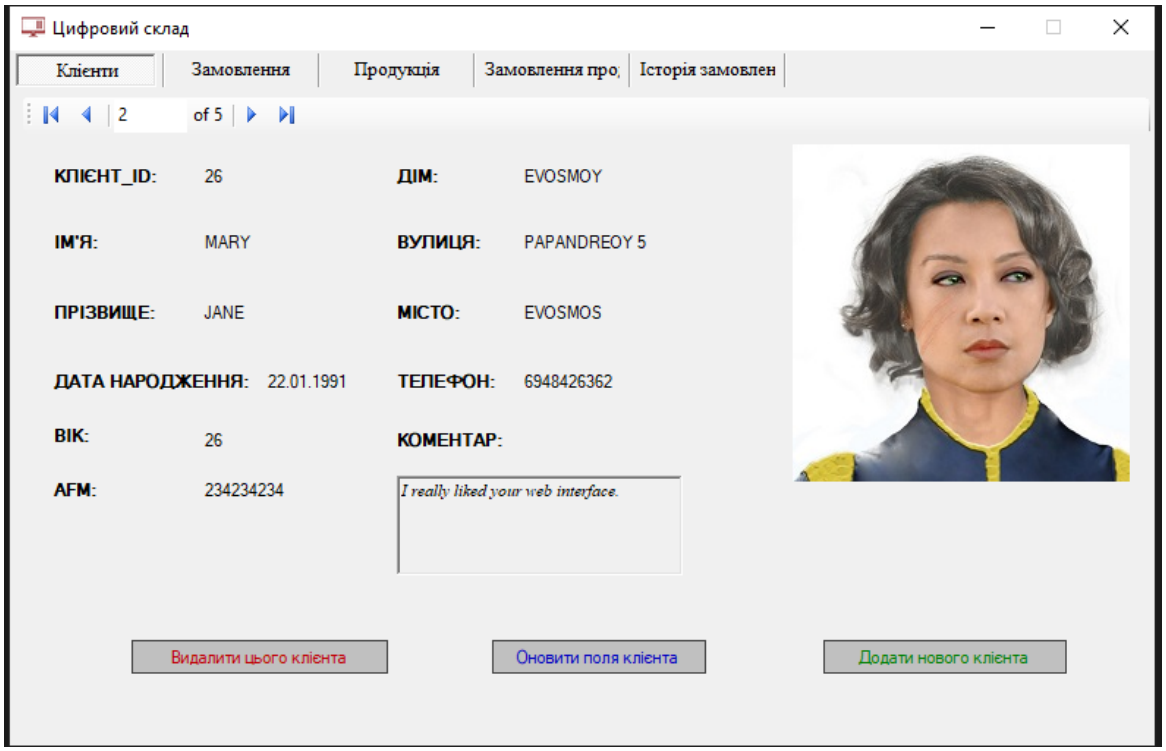


Рисунок 3.4 – Інтерфейс працюючого додатку

3.3 Розробка програмного виробу

Розроблюваний програмний засіб можна розбити на підсистеми, які взаємодіють одна з одною [7].

Рисунок 3.5 – Структура програми

Підсистема роботи з даними надає методи управління. Реалізовані функції створення образу, його збереження і завантаження, доступ до елементів образу, пошуку в образі елемента по його шляху. Підсистема роботи з базою надає доступ до управління базою:

- створення, видалення, перейменування даних;
- ручне сортування даних;
- створення, видалення, перейменування образів;
- переміщення і копіювання образу з однієї категорії в іншу;

Підсистема інтерфейсу реалізує користувацький інтерфейс програми. Категорії та образи представляються в зрозумілому для кінцевого користувача вигляді.

Підсистема управління налаштуваннями — реалізовані методи збереження і завантаження налаштувань призначеного для користувача інтерфейсу і параметрів.

Реалізація на програмному рівні буде за допомогою Windows Forms - назва інтерфейсу програмування додатків (API), що відповідає за графічний інтерфейс користувача і є частиною Microsoft .NET Framework. Windows Forms спрощує доступ до візуальних компонентів (віджетів) Microsoft Windows шляхом створення обгортки для наявного Win32 API в керованому коді [22].

Основний клас Windows Forms - форма, екземплярами якого є головні й діалогові вікна. Форми є нащадками класу Form, визначеного в просторі імен System.Windows.Forms. Інтерфейс Windows Forms дозволяє працювати в режимі конструктора, додаючи елементи управління (кнопки, поля для введення тексту, поля для показу тексту, меню та інші компоненти, характерні для Windows-додатків) простим «перетягуванням». У коді вони представлені як поля класу форми. Всі елементи управління вікна є об'єктами класів, що містяться в System.Windows.Forms і є нащадками базового класу

Control. Відповідна реакція програми на певну дію (подія), пов'язане з елементом управління — обробник події — оформляється у вигляді методу форми.

Клас TextBox надає елемент керування "текстове поле" Windows. Він містить багато членів, в тому числі властивість Text, що містить текст, і метод Clear, що очищає текстове поле.

Клас Button являє елемент управління Windows "Кнопка". Можна задати обробник події натискання кнопки, наприклад, функцію-член button1_Click, яка буде викликати метод textBox1.Clear (). Спостережуваний ефект — очищення текстового поля textBox1 при натисканні кнопки button1. При запуску програми за допомогою статичного методу Run класу Application створюється екземпляр класу форми. При цьому викликається конструктор форми, не започатковано вікно і всі задані елементи управління. При завершенні роботи всі елементи готуються до вивільнення ресурсів.

Простір імен System.Collections.Generic містить інтерфейси й класи, що визначають універсальні колекції, які дозволяють користувачам створювати строго типізовані колекції, що забезпечують підвищену продуктивність і безпеку типів у порівнянні з не універсальними строго типізований колекціями. Методи цих класів надають часто використовувані алгоритми обробки колекцій даних [23].

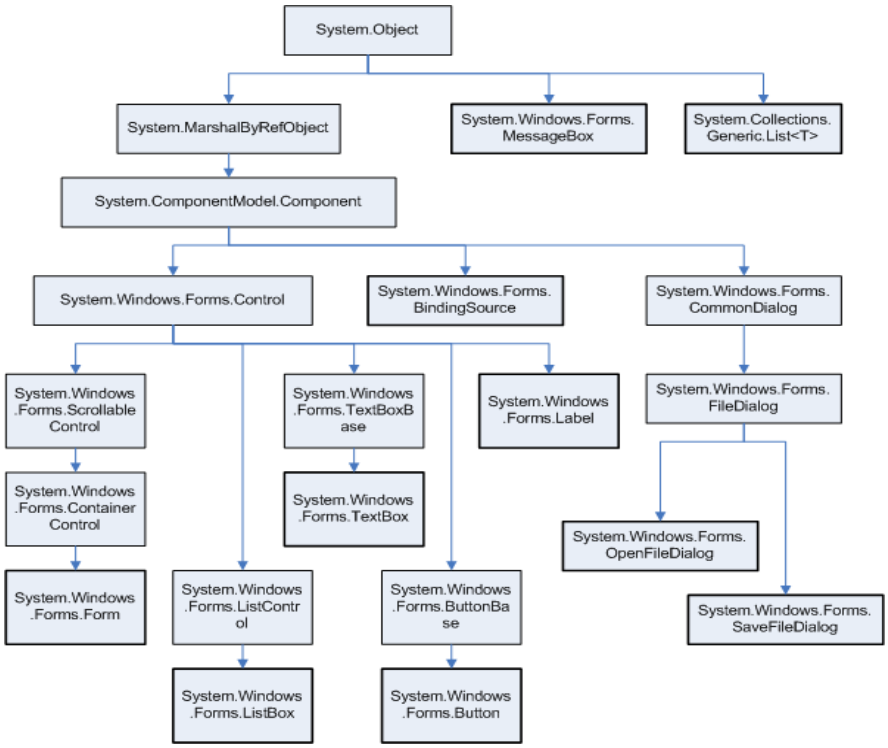


Рисунок 3.6 - Схема успадкування класів, використаних в роботі

Одним з найголовніших етапів розробки, є розробка та підключення бази даних, база даних розроблена за та розміщена в СУБД «Microsoft SQL Server», складається з 4 таблиць.

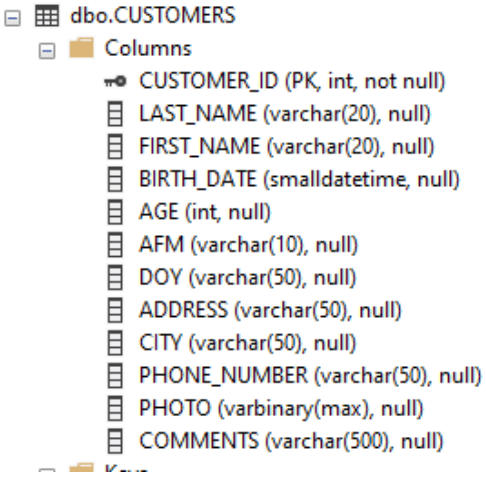


Рисунок 3.7 – Представлення таблиці клієнтів

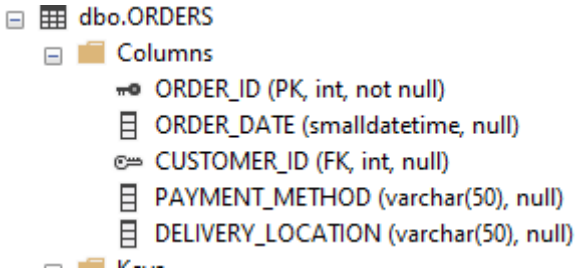


Рисунок 3.8 – Представлення таблиці замовлень

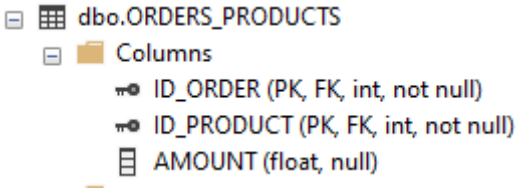


Рисунок 3.9 – Представлення таблиці зв’язку замовлення та продукції

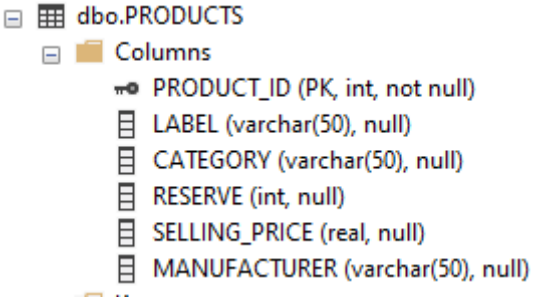


Рисунок 3.10 – Представлення таблиці продукції

Загальна схема створеної бази даних представлена на рисунку 3.11.

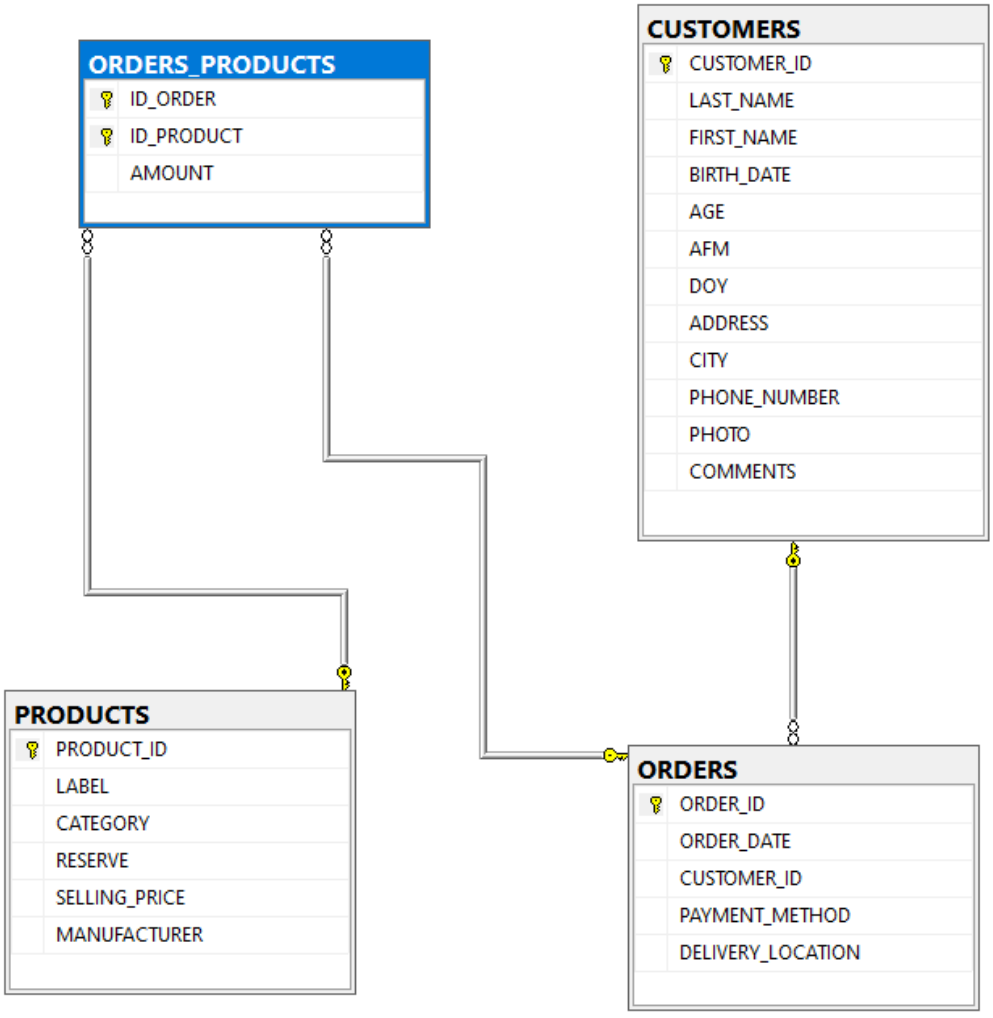


Рисунок 3.11 – Схема бази даних

Підключення до бази даних реалізується в окремому класі «Container», спочатку вказуються змінні, окрім звичайного підключення локальної бази даних є можливість підключення бази даних через доступ мережі.

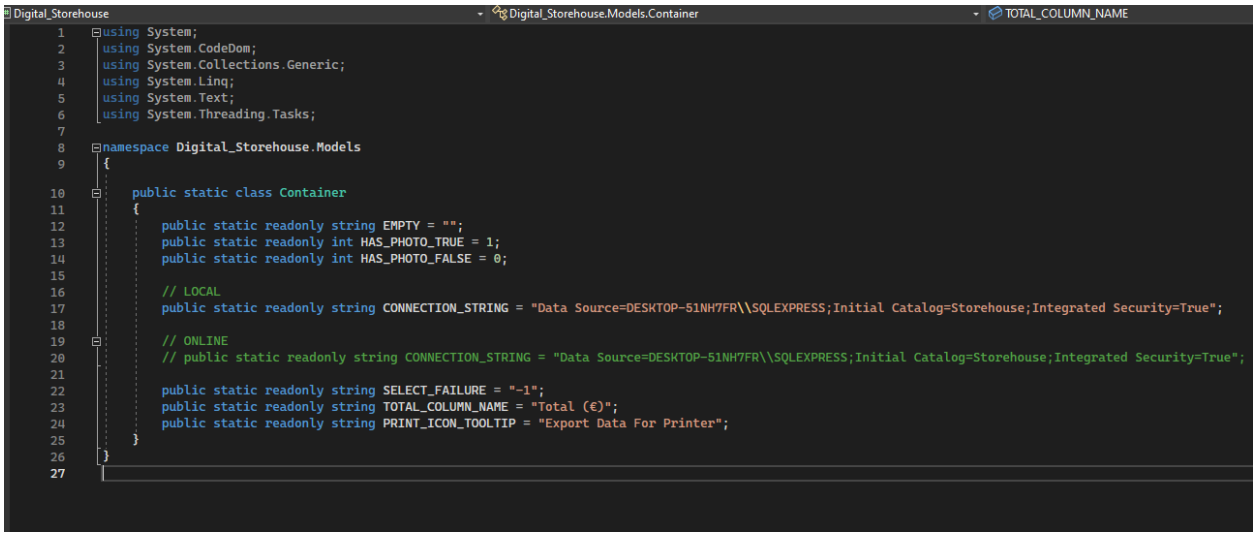


Рисунок 3.12 – Підключення до бази даних в класі «Container»

Додання та оновлення даних для кожної таблиці оголошується в окремому класі, таким чином система зручна для майбутніх правок та повністю прозора і демонструє усі загальноприйняті норми чистого коду.

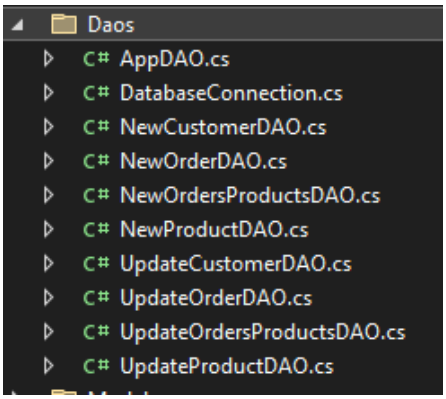


Рисунок 3.13 – Стек класів з запитами до бази даних

Для дій з даними які зберігаються в базі даних і для збереження їх туди, використовуються SQL запити, які передаються через оголошений запит. Далі відкривається підключення до бази даних, і передається оголошений запит, який і вносить дані або зміни, також таким методом реалізовано видалення даних, тому завдяки SQL є змога зручно та швидко спілкуватись з базою даних.

```
1 reference
class UpdateOrdersProductsDAO : DatabaseConnection
{
    1 reference
    public static void updateOrdersProducts(float amount, int idOrder, int idProduct)
    {
        string query = "UPDATE ORDERS_PRODUCTS SET AMOUNT = @AMOUNT_P WHERE ID_ORDER = @IDORDER_P AND ID_PRODUCT = @IDPRODUCT_P";

        SqlCommand command = new SqlCommand(query, Conn);

        command.Parameters.AddWithValue("@AMOUNT_P", amount);
        command.Parameters.AddWithValue("@IDORDER_P", idOrder);
        command.Parameters.AddWithValue("@IDPRODUCT_P", idProduct);

        openConnectionIfClosed();

        try
        {
            command.ExecuteNonQuery();
            Conn.Close();
        }
        catch (SqlException e)
        {
            Conn.Close();
            Controllers.ViewMessages.ExceptionOccured(e);
        }
    }
}
```

Рисунок 3.14 – Лістинг коду оновлення запису

Після оголошення змінної яка містить SQL інструкцію, проходить через оператор команда з відкриттям підключення до бази даних, далі оголошуються параметри через змінні які безпосередньо присутні на формі та в які вноситься інформація. Далі з цих змінних інформація переміщається в SQL інструкцію, після чого закривається підключення.

Також реалізовано оброку виключень, в разі помилки програма не зазнає виключення або раптового закінчення роботи, це здійснюється за допомогою оператора «try catch». Загальне «catch» речення не має списку параметрів ключового «catch» слова. Він відповідає будь-якому винятку, викликаному в «try» блоці.

За допомогою подібної обробки винятків реалізуємо захист додатку він закінчення роботи через помилки, це є дуже важливим, адже під час застосування вимикання або зависання системи не є допустимим, це може нашкодити веденню бізнесу та принести значну шкоду підприємству.

Для того щоб систематизувати усі дані додатку, в класі «ViewMessages» (Рис 3.15), оголошені методи, що викликають повідомлення з інформацією в залежності від типу передбаченої ситуації, таким чином усі

повідомлення збережені в одному класі та можуть швидко редагуватись в разі потреби або змінюватись.

В разі редагування додатку не потрібно шукати повідомлення так як вони структуровані усі в одному класі.

```
using System;
using System.Windows.Forms;

namespace Digital_Storehouse.Controllers
{
    40 references
    class ViewMessages
    {
        0 references
        public static void AgeIsNotValid()
        {
            MessageBox.Show("Поле віку має бути дійсним цілим значенням.", "Помилка формату", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }

        0 references
        public static void CustomerAdded()
        {
            MessageBox.Show("Клієнта успішно додано.", "Успіх", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }

        16 references
        public static void ExceptionOccured(Exception e)
        {
            MessageBox.Show(e.ToString(), "Щось пішло не так:", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }

        0 references
        public static void CustomerUpdated()
        {
            MessageBox.Show("Клієнта успішно оновлено.", "Успіх", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }

        5 references
        public static void FillRequiredFields()
        {
            MessageBox.Show("Будь ласка, заповніть усі необхідні поля",
                "Обов'язкові поля порожні", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }

        0 references
        public static void NonIntegerAge()
        {
No issues found
```

Рисунок 3.15 – Лістинг коду класу «ViewMessages»

Реалізація оновлення даних в базі даних є не менш важливим аспектом реалізації програмного додатку, так як користувач може ввести некоректні дані або заповнити не усі обов’язкові поля, потрібно реалізувати програмний код таким чином, щоб в разі помилок клієнта йому давались підказки.

Це ми реалізуємо за допомогою оператора «if» (Рис 3.16), яким ми перевіряємо чи поле заповнене вірним типом інформації, та в разі його некоректного наповнення викликаємо повідомлення які ми підготували в класі «ViewMessages».

```
1 reference
public static bool validateFields(Dictionary<String, TextBox> productValueLabels)
{
    if (productValueLabels["SELLING_PRICE"].Text.Equals("") || productValueLabels["LABEL"].Text.Equals("")
            || productValueLabels["RESERVE"].Text.Equals(""))
    {
        ViewMessages.FillRequiredFields();
        return false;
    }

    int n;
    if (!int.TryParse(productValueLabels["RESERVE"].Text, out n))
    {
        ViewMessages.ReserveNotValid();
        return false;
    }

    float k;
    if (!float.TryParse(productValueLabels["SELLING_PRICE"].Text, out k))
    {
        ViewMessages.SellingPriceNotValid();
        return false;
    }

    return true;
}
```

Рисунок 3.17 – Лістинг коду перевірки полів

В програмі реалізуємо можливість завантаження фото клієнта (Рис 3.18), для початку оголошуємо метод діалогу за допомогою якого ми обираємо картинку яка буде в майбутньому аватаром клієнта, після чого оголошуємо ще два методи, які реалізують видалення та відвантаження зображення в додаток.

```
1 reference
public static void ShowPhotoDialogChooser(OpenFileDialog dialog, TextBox photoTextbox)
{
    if (dialog.ShowDialog() == DialogResult.OK)
    {
        dialog.Filter =
            "Image files (*.jpg, *.jpeg, *.jpe, *.jfif, *.png) | *.jpg; *.jpeg; *.jpe; *.jfif; *.png";
        dialog.InitialDirectory = @"C:\";
        dialog.Title = "Виберіть фотографію нового клієнта";
        photoTextbox.Text = dialog.FileName;
    }
}


1 reference
public static void removePhoto(PictureBox pictureBox1)
{
    pictureBox1.Image = Properties.Resources.no_photo;
}

1 reference
public static void loadPhotoAfterDialog(PictureBox pictureBox, string photoLabelPathText)
{
    if (!photoLabelPathText.Equals(Container.EMPTY))
    {
        pictureBox.Image = Image.FromFile(photoLabelPathText);
    }

}
```

Рисунок 3.18 – Лістинг коду завантаження аватара клієнта

Заповнення полів на елементах форм, необхідно оголосити окремим методом, через підключення до бази даних, витягування SQL запитом даних з таблиці, та заповненням конкретним полем таблиці відповідного елементу на формі. Таким чином ми можемо прикріпити те чи інше поле з таблиці бази даних до елементу на формі, щоб відображати його.

```
4 references
public void BindCustomerData(Dictionary<string, Label> customerValueLabels, RichTextBox commentsRichTextbox, BindingNavigator bindingNavigator)
{
    openConnectionIfClosed();

    string query = "SELECT * FROM CUSTOMERS";
    AdapterCustomer = new SqlDataAdapter(query, Conn);
    DatasetCustomer = new DataSet();
    AdapterCustomer.Fill(DatasetCustomer, "Customers_Table");

    BindsourceCustomer = new BindingSource
    {
        DataSource = DatasetCustomer.Tables[0].DefaultView
    };

    customerValueLabels["CUSTOMER_ID"].DataBindings.Add(new Binding("Text", BindsourceCustomer, "CUSTOMER_ID", true));
    customerValueLabels["LAST_NAME"].DataBindings.Add(new Binding("Text", BindsourceCustomer, "LAST_NAME", true));
    customerValueLabels["FIRST_NAME"].DataBindings.Add(new Binding("Text", BindsourceCustomer, "FIRST_NAME", true));
    customerValueLabels["BIRTH_DATE"].DataBindings.Add(new Binding("Text", BindsourceCustomer, "BIRTH_DATE", true));
    customerValueLabels["AGE"].DataBindings.Add(new Binding("Text", BindsourceCustomer, "AGE", true));
    customerValueLabels["AFM"].DataBindings.Add(new Binding("Text", BindsourceCustomer, "AFM", true));
    customerValueLabels["DOY"].DataBindings.Add(new Binding("Text", BindsourceCustomer, "DOY", true));
    customerValueLabels["ADDRESS"].DataBindings.Add(new Binding("Text", BindsourceCustomer, "ADDRESS", true));
    customerValueLabels["CITY"].DataBindings.Add(new Binding("Text", BindsourceCustomer, "CITY", true));
    customerValueLabels["PHONE_NUMBER"].DataBindings.Add(new Binding("Text", BindsourceCustomer, "PHONE_NUMBER", true));
    commentsRichTextbox.DataBindings.Clear(); //TODO temporary bug fix
    commentsRichTextbox.DataBindings.Add(new Binding("Text", BindsourceCustomer, "COMMENTS", true));

    bindingNavigator.BindingSource = BindsourceCustomer;
    Conn.Close(); // Close Connection
}
```

Рисунок 3.19 – Лістинг перенесення даних з таблиці до елементів форми

На формі «App», яка є головним меню додатку, під час завантаження слід підключити елементи навігації між записами до їх відповідних записів в базі даних, таким чином буде можливість навігації через ці елементи між записами (Рис 3.20).

```
1 reference
private void App_Load(object sender, EventArgs e)
{
    db.BindCustomerData(CustomerLabels, commentsRichTextBoxGlobal, bindingNavigator_Customer);
    db.BindProductsData(ProductLabels, bindingNavigator_Products);
    db.BindOrderData(OrderLabels, bindingNavigator_Orders);
    db.BindOrdersProductsData(OrdersProductsLabels, bindingNavigator_OrdersProducts);

    NewOrderController.fillCustomerIdCombobox(customerIds_combobox);

    AppController.syncCustomersTab(AppDAO.getBindSourceCustomer(), CustomerLabels, pictureBox_customer,
        deleteCustomer_button, updateCustomer_button);

    AppController.syncProductsTab(AppDAO.getBindSourceProducts(), ProductLabels,
        deleteProduct_button, updateProduct_button);

    AppController.syncOrdersTab(AppDAO.getBindSourceOrders(), OrderLabels,
        deleteOrder_button, updateOrder_button);

    AppController.syncOrdersProductsTab(AppDAO.getBindSourceOrdersProducts(), OrdersProductsLabels,
        deleteRelationship_button, updateRelationship_button);


}
```

Рисунок 3.20 – Лістинг підключення даних до елементів на формі

Класи що, безпосередньо описують інструкції з тих чи інших дій викликаються на головній формі, таким чином пов’язуючи інтерфейс з програмними інструкціями.

```
private void toolStripLabel1_TextChanged(object sender, EventArgs e)
{
    AppController.syncOrdersTab(AppDAO.getBindSourceOrders(), OrderLabels,
        deleteOrder_button, updateOrder_button);
}

1 reference
private void bindingNavigatorCountItem4_TextChanged(object sender, EventArgs e)
{
    AppController.syncCustomersTab(AppDAO.getBindSourceCustomer(), CustomerLabels, pictureBox_customer,
        deleteCustomer_button, updateCustomer_button);
}

2 references
public static RichTextBox getCommentsRichTextbox()
{
    return commentsRichTextBoxGlobal;
}


1 reference
private void customerIds_combobox_TextChanged(object sender, EventArgs e)
{
    AppController.createCustomersHistoryDataGrid(ordersHistory_dataGridView, customerIds_combobox, totalColumn, ordersHistoryTotal_label);
}

1 reference
private void pictureBox1_Click(object sender, EventArgs e)
{
    printDocument1.Print();
}


1 reference
private void printDocument1_PrintPage_1(object sender, System.Drawing.Printing.PrintPageEventArgs e)
{
    AppController.exportDataForPrinter(ordersHistory_dataGridView, e);
}

1 reference
private void orderId_label_Click(object sender, EventArgs e)
{

}
}
```

Рисунок 3.21 – Лістинг виклику методів з класів які описують дії

З метою реалізації звітності в додатку реалізована можливість друку даних по покупка конкретного клієнта (Рис 3.22).

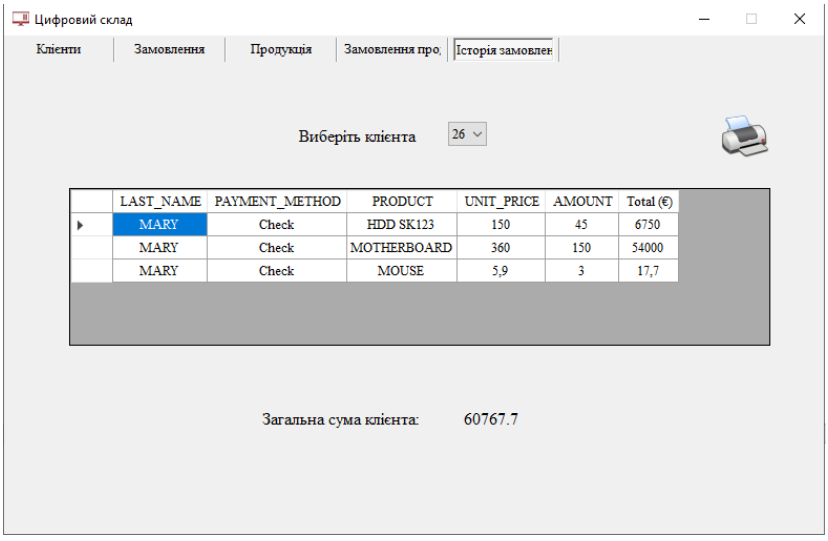


Рисунок 3.22 – Меню друку звіту по конкретному клієнту

Охарактеризуємо методику розділення всіх методів по їх типу до відповідного класу.

Клас є концепцією об'єктно-орієнтованих мов програмування. Клас схожий на структуру, але надає більше функцій. Це дуже важливо, оскільки зменшує складність та узагальнює дані для зменшення довжини коду. Але використовувати його не обов'язково, більшість програм можна зробити без нього, хоча вони можуть стати складними і поганими в читанні їх коду.

Структура надає вам індивідуальний тип даних, тоді як клас надає вам функціональність не тільки оголошувати змінну (відому як об'єкт) індивідуального типу даних, але й узагальнювати функції для об'єктів цього класу.

Розгляньте розташування пам'яті, ви оголосите його цілим числом і присвоєння деякого імені (ім'я змінної) надасть вам змінну для зберігання даних 4/8 біт. Тепер, якщо ви хочете іншу подібну змінну, ви оголошуєте іншу змінну того ж типу даних. Але ви, можливо, чули про масиви, вони зменшують ваші кодові рядки і надають вам подібну концепцію.

Тепер припустимо, що одна змінна є цілочисельною, тоді як інша є рядком (ім'я, слово) - тепер ви будете оголошувати дві змінні різних типів даних, але структура дозволяє, ущільнити цей процес, дозволяючи створювати користувацький тип даних. Аналогічним чином, клас також надасть вам створювати певні функції, обмежені тільки цим класом, наприклад, друк тільки цих.

Хороший код легко читається і розуміється, частково і повністю, іншими (а також автором в майбутньому, намагаючись уникнути синдрому «Чи дійсно я це написав?»).

Під «частково» мається на увазі, що, якщо авто відкриє якийсь модуль або функцію в коді, він повинен бути в змозі зрозуміти, що він робить без необхідності також читати всю решту кодової бази. Він повинен бути максимально інтуїтивним і самодокументованим.

Код, який постійно посилається на найдрібніші деталі, які впливають на поведінку інших (на перший погляд, не мають значення) частин кодової бази, схожий на читання книги, де ви повинні посилатися на виноски або додаток в кінці кожного речення [11].

Добре інкапсульований код має тенденцію бути більш читабельним, розділяючи проблеми на кожному рівні.

Імена мають значення. Активувати мислення швидкий і повільний спосіб, в якому мозок формує думки і покласти деякі фактичні, обережні думки в змінних і методів імен. Кілька додаткових секунд можуть виплатити значні дивіденди. Добре названа змінна може зробити код набагато більш інтуїтивним, тоді як погано названа змінна може призвести до головних підробок і плутанини.

Кмітливість - ворог. При використанні фантазійних методів, парадигм або операцій (наприклад, розуміння списків або трійкових операторів), потрібно бути обережним, щоб використовувати їх таким чином, щоб зробити код більш читабельним, а не тільки коротшим.

Послідовність - це добре. Узгодженість за стилем, як з точки зору того, як ви розміщуєте блоки коду, так і з точки зору операцій, значно покращує читабельність.

Враховуючи усі загальноприйняті норми чистого коду, які повністю відображені в реалізованому коді додатку, ми отримаємо дійсно якісний додаток, який відображає набуті навички з програмування.

РОЗДІЛ 4. ЕКОНОМІЧНИЙ РОЗДІЛ

4.1 Бізнес план

Функції адміністрування включають життєво важливі частини структури організації, допомагаючи організації ефективно управляти ресурсами і людьми. Планування, складання бюджету та організація є трьома основними функціями адміністрування в будь-якій компанії, і брак якості в будь-який з них досить часто завдає шкоди компанії.

Планування життєво важливо для успіху будь-якого підприємства. Це відноситься не тільки до планів на наступний місяць, квартал або рік, а й до п'яти, 10 і 25 років в майбутньому. Коли керівництво IBM на початку 1970-х років заявило, що малоймовірно, що будь-хто захоче використовувати домашній комп'ютер, це показало відсутність передбачення, яке переслідує компанію на довгі роки [14].

Складання бюджету частково є елементом планування, але фінансова організація в компанії повинна мати свою власну інфраструктуру, щоб підтримувати дійсний рівень контролю над організацією. Великі ідеї є важливою частиною будь-якої успішної компанії, але без бюджету, достатнього для фінансування цих великих ідей, вони залишаються ідеями, а не реальністю.

Організація є третьою частиною основних функцій адміністрації. Багато що з цього пов'язане з призначенням окремих людей і відділів для конкретних завдань і забезпечення виконання всіх допоміжних завдань для більш широкої мети.

Прибуток – це та частина виручки, що залишається після відшкодування всіх витрат на виробничу і комерційну діяльність підприємства.

Дохід – це загальна сума коштів, яка поступає підприємству за певний період і за вирахуванням податків може бути використана на інвестування і споживання.

Перелік обладнання та програмного забезпечення, необхідного для розробки та функціонування створеного програмного продукту показаний у таблиці 4.1.

Таблиця 4.1 - Кошторис витрат на придбання обладнання, необхідного для функціонування програмного продукту «Цифровий склад».

Назва обладнання, програм	Кількість	Ціна за одиницю, грн.	Загальна вартість, грн.	Примітка
Ноутбук Asus ASUS VivoBook 17	1	11999,00	11999	Придбано раніше
Мишка A4Tech G3-200N Wireless Black	1	199,00	199	Придбано раніше
Принтер	1	8160,00	8160	Придбано раніше
Операційна система Microsoft Windows 10 Pro	1	8950,00	8950	Придбано раніше
Пакет офісних програм **Microsoft Office 2016**	1	7009,00	7009	Придбано раніше
Всього:			36317	

Проектування програмного продукту для організації менеджменту підприємства складається з низки послідовних, цілеспрямованих, взаємозв’язаних та взаємообумовлених етапів, на які можна поділити весь часовий відрізок, що відводиться на розробку проекту. Такі етапи приведені у таблиці 4.2. Головна мета планування процесу розробки – це визначення необхідних ресурсів на всіх його етапах.

Таблиця 4.2 – Перелік етапів та робіт по розробці проекту

Найменування	Вид роботи	Виконавець, посада
стадії		
1 Підготовча стадія	Дослідження проблеми	Програміст
	Вивчення та аналіз аналогічних розробок і вибір систем для реалізації рішення	Програміст
	Економічне обґрунтування доцільності виконання бази даних	Бухгалтер
	Складання ТЗ	Програміст
	Узгодження ТЗ із зацікавленою стороною	Програміст, комерційний директор
	Складання плану та розрахунок розробки проекту розпорядку пар	Програміст, бухгалтер
2 Технічна пропозиція	Доведення техніко-економічного обґрунтування	Бухгалтер
	Аналіз ТЗ та визначення пріоритетних аспектів розробки	Бухгалтер
	Доведення та уточнення загального обсягу робіт, строків виконання та витрат	Програміст, бухгалтер
3 Теоретична розробка	Дослідження технічних особливостей майбутнього програмного продукту	Програміст
	Визначення переліку технологій, які використовуватимуться при розробці програми	Програміст
	Визначення форматів даних	Програміст
4 Практична	Розробка структури даних та схеми	Програміст

реалізація	взаємодії її компонент	
	Розробка графічного користувацького інтерфейсу	Програміст
	Автономна відладка окремих модулів експорту	Програміст
5 Доробка всього комплексу програмного забезпечення	Тестування системи у реальних умовах	Програміст
	Доопрацювання програмного продукту відповідно до результатів тестування	Програміст
	Підготовка звіту про розробку програми	Програміст
6 Заключна стадія	Підготовка презентації	Програміст
	Демонстрація програмного продукту	
	Навчання роботи з програмою	

На основі даного переліку визначається кількість виконавців, тривалість виконання робіт в днях, та рівень оплати праці виконавців по видам робіт, що виконуються. Такі дані приведені у таблиці 4.3.

Таблиця 4.3 – Зведені результати тривалості та трудомісткості розробки та реалізації проекту.

Найменування роботи	Виконавець, посада, спеціальність	Кількість виконавців	Тривалість виконання роботи, годин	Годинна тарифна ставка, грн	Оплата праці, грн
1	2	3	4	5	6
Дослідження проблеми Вивчення та аналіз аналогічних розробок і вибір систем для реалізації рішення Економічне	 Програміст Бухгалтер Комерційний директор	 1 1 1	 4 2 2	 24,80 24,80 33,23	 99,20 49,60 66,46

обґрунтування доцільності виконання бази даних Складання ТЗ Узгодження ТЗ із зацікавленою стороною Складання плану та розрахунок розробки проекту розпорядку пар					
Доведення техніко-економічного обґрунтування Аналіз ТЗ та визначення пріоритетних аспектів розробки Доведення та уточнення загального обсягу робіт, строків виконання та витрат	Бухгалтер Програміст	1 1	2 2	24,80 24,80	49,80 49,80
Дослідження технічних особливостей майбутнього програмного продукту Визначення переліку технологій, які використовуватимуться при розробці програми	Програміст	1	20	24,80	496,00

Визначення форматів даних					
Розробка структури даних та схеми взаємодії її компонент Розробка графічного користувацького інтерфейсу Автономна відладка окремих модулів експорту	Програміст	1	44	24,80	1003,20
Тестування системи у реальних умовах Доопрацювання програмного продукту відповідно до результатів тестування Підготовка звіту про розробку програми	Програміст	1	16	24,80	396,80
Підготовка презентації Демонстрація програмного продукту Навчання роботи з програмою	Програміст	1	2	24,80	49,60
Всього					2260,46

Витрати на науково-дослідницьку роботу по розробці програмних засобів та апаратури, щодо даного дипломного проекту, включають наступні елементи:

- витрати на основну заробітну плату виконавців. Це сума часткових заробітних плат за кожну роботу і вона приведена в таблиці 4.3.

$$ЗП_{основна}= 2260,46 \text{ грн}$$

- додаткова заробітна плата. Її можна обчислити за формулою:

$$ЗП_{додаткова}=0,2*ЗП_{основна} \quad (4.1)$$

$$ЗП_{додаткова}=0,2*2260,46=452.09 \text{ грн}$$

Таким чином загальний фонд заробітної плати, що обчислюється за формулою:

$$ФЗП=ЗП_{основна}+ЗП_{додаткова}. \quad (4.2)$$

$$ФЗП=2260,46 + 452.09 =2712.55 \text{ грн}$$

- обов’язкові відрахування на заробітну плату (єдиний соціальний внесок - 22%). Таким чином обов’язкові відрахування складають:

$$Відр = ФЗП * 22/100$$

$$Відр = 2712.55 * 22/100 =596.76 \text{ грн}$$

- накладні витрати розраховуємо за формулою:

$$НВ=0,4*ЗП_{основна}. \quad (4.3)$$

$$НВ = 0,4*2260,46 =904.19 \text{ грн}$$

Результати розрахунку інвестиційних витрат приведені в таблиці 4.4.

Таблиця 4.4 - Перелік інвестиційних витрат

Найменування елементу витрат	Сума, грн
Капітальні затрати	-
Основна заробітна плата виконавців	2260,46
Додаткова заробітна плата виконавців	452.09
Нарахування на заробітну плату (єдиний соціальний внесок)	596.76
Накладні витрати	904.19
Витрат всього	4213.50

Загальні інвестиційні витрати таким чином складають 4213.50 гривень.

Визначимо витрати на амортизацію необхідного обладнання, програмного забезпечення за формулою:

$$Ва = \frac{Вобл}{100} \times На \qquad (4.4)$$

$$Ва = \frac{29927}{100} \times 20 \times \frac{11}{249} = \text{грн}$$

Де Вобл – вартість обладнання, програмного забезпечення, грн.;

На – норма амортизаційних відрахувань.

Таблиця 4.5 - Собівартість створення бази даних

Найменування елементу витрат	Сума, грн
Основна заробітна плата виконавців	2260,46
Додаткова заробітна плата виконавців	452.09
Нарахування на заробітну плату (єдиний соціальний внесок)	596.76
Накладні витрати	904.19
Витрати на амортизацію обладнання	299,27
Витрат всього	4512,77

Таким чином собівартість проекту складає: СП = 4512,77 грн

Експлуатаційними витратами є такі витрати, які забезпечують нормальне функціонування певного технічного рішення в період його експлуатації. Величина експлуатаційних витрат в розрахунку за 1 рік може бути спрогнозована за формулою:

$$Е = k \cdot Ц \cdot \beta = k \cdot А \cdot СП \cdot \beta \qquad (4.5)$$

де Ц — ціна реалізації нової розробки, якщо вона була визначена раніше, грн/од.,

k — коефіцієнт, який ураховує витрати на амортизацію, електроенергію, обслуговування, ремонти тощо. Для обчислювальної техніки k = 0,5…0,7.

А — коефіцієнт, який ураховує прогнозований прибуток та податки, які повинен сплачувати виробник А ≈ 1,7…2,3;

СП — собівартість нової розробки;

β — доля часу, який витрачає працівник на обслуговування технічної розробки в загальному часі своєї роботи. ($\beta \approx 0,07$)

$$E = 0,5*1,7*4512,77*0,07=268.51 \text{ грн}$$

Використання бази даних відпуску продукції ТОВ «Український кристал» дозволить суттєво зменшити витрати часу на її проведення.

За рахунок використання бази даних витрати часу на обробку замовлень зменшаться на 160 год на рік. Тому річна економія коштів від використання бази даних становитиме:

$$Еф = 160 *24,80- 268.51=3699.49 \text{ грн}$$

Строк окупності капітальних затрат розраховується за формулою:

$$Ток = \frac{СП}{Еф} \tag{4.6}$$

$$Ток = \frac{4512,77}{3699,49} = 1,22$$

Коефіцієнт економічної ефективності розраховується за формулою:

$$Кеф = \frac{1}{Ток} \tag{4.7}$$

$$Кеф = \frac{1}{1,22} = 0,82$$

Таблиця 4.6 - Основні економічні показники

Показники	Значення
Собівартість проекту, грн.	4512,77
Сума умовно-річної економії, грн.	3699.49
Строк окупності, років	1,22
Коефіцієнт економічної ефективності	0,82

В результаті виконаних розрахунків техніко-економічних показників загальна сума інвестицій на створення проекту складає 4512,77 грн. термін окупності становить 1,22 року.

Коефіцієнт ефективності інвестицій, який показує отримання прибутку на 1 грн. капітальних вкладень, дорівнює 0,82 грн.

В наслідок проведеного аналізу основних аспектів, пов’язаних з розробкою програмного продукту можна зробити висновок, що розробка такої програми в наш час є актуальною ідеєю.

РОЗДІЛ 5. ОХОРОНА ПРАЦІ

5.1 Політика безпечного використання ПК

Коли ми працюємо з нашими комп’ютерами та нашим обчислювальним обладнанням, одна з найбільших проблем безпеки, яку ми маємо, – це робота з джерелом живлення. Електроенергія, очевидно, може бути дуже небезпечною, і ми хочемо бути впевненими, незалежно від того, чи працюємо ми на дуже маленькому комп’ютері чи в центрі обробки даних, що завжди дбаємо про електрику [27].

Таким чином, одна з найпоширеніших найкращих практик полягає в тому, щоб відключити будь-яке джерело живлення, перш ніж безпосередньо працювати з будь-яким типом компонента. Ніколи не варто вмикати цей пристрій або навіть підключати його до джерела живлення, коли ви працюєте всередині нього.

Не торкайтеся нічого, що знаходиться всередині комп’ютера, якщо ви не впевнені, що немає проблем із живленням. Дуже часто джерела живлення, особливо деякі зі старих блоків живлення, зберігають велику потужність у своїх конденсаторах. І очевидно, що деякі з старих ЕПТ, безумовно, зберігають велику потужність у своїх конденсаторах. Тож ви хочете бути абсолютно впевнені, коли працюєте з цими компонентами, що ніколи не виникне ризик будь-якого типу удару.

Ось чому ви часто побачите, як люди замінюють цілі блоки живлення всередині комп’ютера, а не намагаються відремонтувати окремі компоненти. Зазвичай набагато дешевше просто замінити джерело живлення та заощадити час. І, зрештою, ймовірно, набагато безпечніше тримати вас подалі від компонентів усередині цього джерела живлення та просто замінити його на те, що, як ви знаєте, працює належним чином.

Довгострокове використання комп'ютерів було пов'язано з низкою потенційних проблем зі здоров'ям, або «розлади, пов'язані з комп'ютером»

(CRD). У цьому розділі описані ці потенційні ризики для здоров'я та наведено огляд правил які допоможуть вберегти здоров’я користувача.

За останні роки виникло безліч питань щодо зв'язків, які можуть існувати між використанням комп'ютерів і здоров'ям і безпекою тих, хто їх використовує. Деякі наслідки для здоров'я - такі як біль у суглобах і напруга очей після тривалого періоду тулилися на екрані і клавіатурі - визнаються як актуальність багатьма. Однак довести з будь-яким ступенем, безумовно, довгострокові наслідки для здоров'я використання комп'ютера залишається проблематичним. Не в останню чергу це тому, що широке використання комп'ютерів все ще є відносно сучасним явищем, причому межі між комп'ютерами та іншими електронними пристроями також продовжують розмиватися.

Чинне законодавство, наприклад, дає зрозуміти, що використання комп'ютера не повинно викликати прилягання до епілептика. Однак, враховуючи, що у світі також прийнято, що перегляд миготливих відеозображень на телебаченні може викликати таке прилягання, відразу стає очевидно, що і чинне керівництво, і законодавство є неадекватними. Існуючі

Таким чином, в той час як наступне звітує про чинне законодавство та керівництво щодо використання комп'ютерної техніки, його історичний контекст також потрібно пам'ятати.

Проблеми зі здоров'ям найбільш сильно пов'язані з використанням комп'ютерної техніки - це розлади верхніх кінцівок, проблеми з очима, стрес і втома, скарги на шкіру. Розлади верхніх кінцівок - термін, що використовується для опису ряду станів, що впливають на пальці, руки, руки і плечі. Такі умови можуть варіюватися від легких болів і болів, до хронічної тканини і/або м'язової скарги. Повторювана травма напруги (RSI) - одна з таких умов. Це пояснюється надмірним виконанням повторюваних, спритних операцій. В результаті такої повторюваної активності може розвинутися теносиновіт (набряклі м'язи) або синдром зап'ястного тунелю (набряклі сухожилля) [27].

Повторювана травма напруги може бути результатом тривалого високошвидкісного набору тексту, інтенсивного використання миші або дійсно тривалого використання комп'ютерної ігрової панелі управління. Ранні ознаки повторюваної травми напруги включають поколювання або оніміння в пальці або пальці, а також біль або навіть набряк через руки і навіть верхні руки. У Сполучених Штатах, Національний інститут охорони праці повідомив, що 40 відсотків людей, які працюють переважно з комп'ютерами, страждають від деяких симптомів RSI, причому понад десять відсотків відчувають постійний дискомфорт.

Чи є проблеми очей результатом коротко- або довгострокового використання комп'ютера залишається предметом суперечок. Однак будь-яка форма надмірної тісної роботи, де очі змушені зосереджуватися на досить фіксованій відстані на відносно обмеженому місці, ймовірно, викличе принаймні короткочасну напругу очей і дискомфорт. Це просто тому, що така діяльність не те, що наші очі коли-небудь були «призначені» робити! Дійсно, я пам'ятаю, як одного разу запитав оптика, якщо використання комп'ютера пошкоджує очі. Відповідно до рекомендацій уряду Великобританії, він повідомив мені, що використання комп'ютера не може призвести до скарг очей. Потім він відразу порадив мені ніколи не користуватися комп'ютером безперервно більше півтори години!

Хоча зв'язок між довгостроковими проблемами очей офіційно оскаржується (навіть якщо практичний досвід тих, хто працює на комп'ютері весь день передбачає інше), короткострокові комп'ютерні проблеми очей незаперечні. Вони, як правило, пов'язані з хворими очима, головними болями, розмитим або пом'якшеним зором, а іноді і залишковим після зображення, випробуваним протягом коротких періодів.

Як і заходи, які можуть бути прийняті, щоб уникнути повторюваної травми напруги, зміни в робочому режимі, щоб дозволити перерви, а також зміни постави, щоб уникнути фокусування з тієї ж фіксованої відстані протягом дуже довгих періодів, може бути досить ефективним у боротьбі з

проблемами очей. Більшість людей також вважають, що плоский екран (TFT) дисплеї, як правило, більш зручні для використання протягом тривалого періоду часу. Великі екрани також схильні викликати меншу напругу очей, ніж невеликі дисплеї, оскільки вони заохочують користувача дивитися навколо дисплея (рухаючи як головою, так і очима), а не просто постійно фокусуватися на одному місці.

Багато людей вважають, що використання комп'ютера протягом тривалого часу призводить до надмірного стресу і втоми. Це може бути в результаті задіяних високих рівнів спритної активності і візуальної концентрації. Як вже зазначалося, це не «природна» форма людської діяльності, а значить, потенційне джерело стресу. Деякі люди також вважають, що комп'ютер використовує незручно або дратує, коли їм доводиться відповідати програмному або апаратному забезпеченню, що обмежує те, як вони хочуть зробити щось, або що постійно збиває або змушує їх постійно чекати завершення дій. Технологія, здатна виконувати роботу, може уникнути останньої (якщо вона є!), в той час як різноманітна робота з відповідними перервами знову є принаймні частковою відповіддю.

Нарешті, невелика кількість людей відчувають скарги шкіри, пов'язані з довгостроковим використанням комп'ютера. Такі умови можуть включати свербіж, висипання на шиї, обличчі або руках, а також суху шкіру. Цілком чому це може бути знову обговорено, хоча потенційно скарги на шкіру можуть бути результатом електростатичних розрядів (створених в лазерних принтерах та екранах дисплеїв електронно-променевої трубки (CRT)) та/або сухого повітря та статичної електроенергії, яка накопичується в офісах, наповнених комп'ютерним обладнанням. Відповідна вентиляція, звичайно, завжди хороша ідея і може зменшити проблему. Також може допомогти використання зволожувачів повітря та антистатичного матування. Однак багато людей, мабуть, навіть розуміють, що використання комп'ютера впливає на їх шкіру після тривалої відпустки або зміни роботи в інше робоче середовище.

Згідно з поточними офіційними медичними дослідженнями, електромагнітне випромінювання, випромінюване моніторами електронно-променевої трубки, не має наслідків для здоров'я, без пов'язаних згодом ризиків збільшення викиднів або вроджених дефектів, пов'язаних з використанням візуального дисплея вагітними жінками. (Це говорить, що офіційні рекомендації дозволяють припустити, що вагітним жінкам, які стурбовані використанням комп'ютера, слід дати можливість висловити свої побоювання). На практиці більшість (якщо не всі) CRT-дисплеїв не були замінені на плоскі TFT-екрани, тому принаймні знімають можливість будь-якого CRT-випромінювання для більшості користувачів.

Деякі працівники - наприклад, секретарі, типісти, адміністратори дат і оператори телепередач - можуть бути легко визначені як звичайні користувачі (якщо вони працюють у компанії) або оператори (якщо вони самозайняті). В інших випадках класифікації зазвичай можуть бути зроблені з посиланням на наступні питання:

- Чи залежить працівник від екрану дисплея обладнання, щоб зробити свою роботу?
- Чи має працівник свій розсуд щодо використання обладнання екрану дисплея?
- Чи отримав працівник значну підготовку у використанні екранного обладнання?
- Працівник зазвичай використовує обладнання екрану дисплея протягом години або більше за раз?
- Працівник використовує дисплей екрану обладнання більш-менш щодня?
- Чи є швидка передача даних між працівником і екраном важливою вимогою до роботи?
- Чи обладнання екрану дисплея вимагає високого рівня уваги та концентрації працівників?

Якщо відповідь на більшість або всі перераховані вище питання «так», то працівник повинен бути класифікований як користувач екранного обладнання (якщо вони є працівником) або як оператор (якщо вони самозайняті, але працюють за місцем розташування клієнта).

Оцінки ризиків, як зазначено вище, досить очевидно призначені для забезпечення того, щоб всі робочі станції екранного обладнання відповідали законним вимогам охорони здоров'я та безпеки. Ці вимоги стосуються компонентів самої робочої станції, робочого середовища та використовуваного програмного забезпечення.

З точки зору компонентів робочої станції, вимоги полягають в тому, що:

- Зображення повинно бути чітким, стабільним і вільним від мерехтіння.
- Символи повинні бути чітко визначені, чітко сформовані, відповідного розміру і з достатнім інтервалом між лініями.
- На екрані не повинно бути відблисків або відблисків, які можуть викликати дискомфорт користувача/оператора.
- Має бути регулювання яскравості та контрастності користувача/оператора.
- Дисплей повинен вільно регулюватися як нахилом, так і поворотом.
- Клавіатура повинна бути нахилена і відокремлена від екрану.
- Перед клавіатурою має бути достатньо місця для підтримки зап'ястя.
- Клавіатура повинна мати матову поверхню, щоб уникнути відблисків/відбиття.
- Легенди клавіатури повинні бути розбірливими з адекватним контрастом і визначенням.
- Будь-який наданий власник документа повинен бути стабільним і регульованим.

- Робоче крісло має бути стабільним, дозволяти легку свободу пересування та досягнення відповідної робочої позиції.
- Висота робочого крісла повинна регулюватися.
- Робочий стілець задньої висоти і нахилу повинен бути регульований.
- Підставка для ніг повинна бути надана кожному користувачеві або оператору, який запитує його

З точки зору більш широкого робочого середовища, вимоги полягають в тому, що [27]:

- Є достатньо місця для роботи в зручному положенні, з можливостями для різних рухів і постави.
- Освітлення має бути задовільним і забезпечувати відповідний контраст між екраном дисплея та фоновим середовищем. Відблиски і відблиски (наприклад, з вікон) слід уникати, при цьому жалюзі (бажано вертикальні) використовуються для запобігання попадання сонячного світла на екрани дисплеїв. Поверхні повинні мати матове покриття, щоб уникнути відблисків і відблисків.
- Рівні шуму, створені обладнанням робочої станції, не повинні відволікати користувача/оператора.
- Жоден елемент обладнання робочої станції не повинен генерувати надлишкове тепло, яке може викликати дискомфорт користувача/оператора. Всі електромагнітні випромінювання за межами видимого спектру повинні бути екрановані до незначних рівнів (досягається покупкою TFT-монітора з плоским екраном або CRT-дисплея, сумісного з MRP II).

З точки зору програмного забезпечення, вимоги полягають в тому, що:

- Програмне забезпечення має підходити до поставленого завдання.

- За умови відповідного навчання програмне забезпечення повинно бути простим у використанні та адаптованим до досвіду та знань користувача/оператора.
- Жоден моніторинг продуктивності (наприклад, вимірювання натискань клавіш на годину) не повинен використовуватися без відома відповідного користувача/оператора.
- Програма повинна надавати зворотній зв "язок користувачеві/оператору щодо його стану та продуктивності
- Відповідні ергономічні принципи повинні бути використані в розробці програмного забезпечення.

На практиці більшість вищезазначених вимог буде виконано шляхом придбання сучасного комп'ютерного обладнання та програмного забезпечення, а також шляхом встановлення його в робочому місці з достатнім простором і там, де було приділено належну увагу освітленню (включаючи заходи щодо зменшення відблисків та відбиття екрану). У багатьох випадках предметом обладнання, що вимагає найбільшої уваги для задоволення вищевказаних вимог, буде регулювання або інше крісло користувача/оператора (з великою кількістю дешевших офісних стільців, які не мають регульованої задньої висоти та нахилу).

5.2 Обов’язки програміста

- піклуватися про особисту безпеку і здоров’я, а також про безпеку і здоров’я оточуючих людей у процесі виконання будь-яких робіт або під час знаходження на території підприємства;
- знати і виконувати вимоги інструкцій з охорони праці і по видах робіт на своєму робочому місці;
- виконувати роботу відповідно до вимог інструкційно-технологічної карти;

- вміти користуватися засобами індивідуального і колективного захисту;
- знати і виконувати Правила поводження з устаткуванням, інвентарем, користуватися технічним паспортом на устаткування;
- знати і виконувати обов'язки з охорони праці, передбачені колективним договором (трудовим договором), правилами внутрішнього трудового розпорядку підприємства, в тому числі:
- вчасно починати і закінчувати роботу, дотримуватися розкладу технологічної і обідньої перерв;
- не виконувати роботи, що не передбачені змінним завданням;
- не перебувати на роботі в неробочій час без відповідного розпорядження керівника;
- дотримуватись правил корпоративного поводження;
- проходити в установленому порядку медичні огляди;
- вміти надати першу допомогу потерпілому від нещасного випадку;
- перед початком роботи перевіряти справність устаткування, огороджень, інженерно-технічних засобів безпеки, інвентарю, засобів пожежогасіння;
- співпрацювати з роботодавцем у справі організації безпечних і нешкідливих умов праці, особисто вживати можливих заходів щодо усунення будь-якої ситуації, що створює загрозу її життю чи здоров'ю або людям, які її оточують та навколишньому природному середовищу;
- при виявленні недоліків чи небезпеки зобов'язана повідомляти безпосереднього керівника або іншу посадову особу.

Перевірити:

- справність обладнання, інструменту, приладів;
- наявність і справність достатнього освітлення, вентиляції, обладнання тощо;
- перевірити справність рубильників, розеток, штепсельних з'єднань тощо.

У випадку виявлення будь-яких відхилень, несправностей, пошкоджень негайно повідомити директора підприємства.

ВИСНОВКИ

Управління складом є важливою частиною роботи всієї компанії. Розподіл обов'язків і, перш за все, спосіб, яким працює система розподілу, вилучення і вибору предметів на складі, визначає правильне і ефективне функціонування складу. Розроблена складська інформаційна система є актуальним рішенням для ефективного функціонування складу готової продукції.

Метою підсумкової кваліфікаційної роботи було вдосконалення роботи складу готової продукції на етапі розміщення, пошуку, переміщення та списання товарів зі складу через впровадження інформаційної системи складського обліку, що забезпечить цільове зберігання товарів на складі.

Для досягнення цієї мети були виконані наступні завдання:

- Вивчено предметну область;
- Розглянуто ряд існуючих розробок для вирішення поставленої мети;
- Обрано методи та інструменти реалізації;
- Сформовано технічне завдання на проектування.
- Розроблено інформаційну систему;
- Проаналізовано бізнес доцільність розробки.

Розроблена інформаційна система забезпечує:

- Ведення обліку руху товарів на складі;
- Повний контроль всіх складських операцій;
- швидкий пошук і підбір даних про можливі характеристики номенклатури;
- Прискорення процесу завершення замовлення та відвантаження;

В результаті виконаної роботи було розроблено систему управління роботи підприємства з реалізації продукції. Користувачі за допомогою інтуїтивно зрозумілого інтерфейсу, можуть в зручній формі вести облік клієнтів та товарів, категорій товарів, вести звітність по замовлення клієнтів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Маккі А. Введення в NET 4.0 і Visual Studio 2010 для професіоналів / Алекс Маккі., 2012. – 416 с.
2. Роберт С. М. Чистий Код / Сесіл Мартін Роберт., 2019. – 448 с. – (Фабула). – (#PROSystem).
3. Гайдаржи В. Бази даних в інформаційних системах / В. Гайдаржи, І. Ізварін. – м. Київ: Університет "Україна", 2018. – 418 с.
4. Коноваленко І. В. Програмування мовою C# 6.0 / І. В. Коноваленко. – Тернопіль: Тернопіль, ТНТУ, 2016. – 227 с. – (Hurtom).
5. Correspondence of the entry in "select" to the entry in the field [Електронний ресурс] // Stack Exchange Inc. – 2023. – Режим доступу до ресурсу: https://stackoverflow.com/.
6. Тарасюк Г.М. Управління проектами [Текст]: навчальний посібник для студентів вищих навчальних закладів / Г. М. Тарасюк– К.: Каравела, 2004. – 344с.
7. Вступ до алгоритмів / Т.Кормен, Ч. Лейзерсон, Р. Рівест, К. Стайн. – м. Київ: К.І.С., 2019. – 1288 с.
8. Кобиляцький Л.С. Управління проектами [Текст]: навч.посіб. / Л.С. Кобиляцький –К.: МАУП, 2002. – 200 с.
9. Троєлсен Е. Мова програмування C# 9 та платформа .NET 5: основні принципи та практики програмування / Е. Троєлсен, Ф. Джепікс. – Київ: Науковий Світ, 2023. – 770 с. – (10-е видання).
10. Гаст Х. Об'єктно-орієнтоване проектування. Концепції та програмний код / Хольгер Гаст., 2019. – 1040 с.
11. Кнут Д. Мистецтво програмування, том 1. Основні алгоритми (3-е видання) / Дональд Кнут.. – 720 с.

12. Галісєєв Г. Системне програмування / Геннадій Галісєєв. – м. Київ: Університет "Україна", 2019. – 113 с.
13. Кривий С. Вступ до методів створення програмних продуктів / Сергій Кривий. – м. Чернівці: Букрек, 2012. – 424 с.
14. Інформаційні технології та моделювання бізнес-процесів / О.Томашевський, Г. Цегелик, М. Вітер, В. Дубук., 2012. – 296 с.
15. Береза А.М. Основи створення інформаційних систем / Навч. посіб. – К.: КНЕЧ, 1998. – 140 с.
16. Фрімен Е. Head First. Патерни проєктування / Е. Фрімен, Е. Робсон. – м. Харків: Фабула, 2020. – 672 с.
17. Інформаційні системи, їх види; апаратне та програмне забезпечення інформаційної системи. [Електронний ресурс] // Kievoit. – 2013. – Режим доступу до ресурсу: http://www.kievoit.ippo.kubg.edu.ua/kievoit/2013/95/95.html.
18. Єдина система конструкторської документації. Основні написи (ГОСТ 2.104-2006, IDT). ДСТУ ГОСТ 2.104:2006 / Вид. Офіційне. - Київ: Держспоживстандарт України, 2007р-16 с.
19. Текстові документи. Загальні вимоги СОУ 207.01:2004. - 3-тє вид., із змінами / В. Каплун, В. Олександренко, Л. Першина, Л. Безсмертна, О. Снозик. - Хмельницький: ХНУ, 2008. - 40 с.
20. Буч. Об'єктно-орієнтований аналіз та проектування з прикладами додатків на C#, 2-е вид. / Пер. з англ. - М .: «Видавництво Біном», СПб: «Невський діалог», 2008 р - 560 с.
21. Б.М. Голуб – C#. Концепція та синтаксис / C#. Концепция и синтаксис, Видавництво: Львів: Видавничий центр ЛНУ ім. Івана Франка, 2006 - Кількість сторінок: 136 с.
22. Фаулер М. UML. Основи. / М. Фаулер, К. Скотт., 2018. – 192 с. – (Символ-Плюс). – (3).
23. Купчик М.П., Гандзюк, Литвиненко А.М., Іваненко. О. В. Основи Охорони праці. - К.: ДУТ, 2014. – 215 с.

24. Соколов В. Ю. Інформаційні системи і технології: Навч. посіб. – К.: ДУІКТ, 2010. – 138с.
25. Пономаренко В.С. Проектування інформаційних систем: навч. посіб. / Пономаренко В.С. – К.: Академія, 2002. – 544 с.
26. Недашківський О.Л. Планування та проектування інформаційних систем. / О.Л. Недашківський. – К.: ДУТ, 2014. – 215 с.
27. Охорона праці в IT компанії [Електронний ресурс] // Legal IT Group. – 2022. – Режим доступу до ресурсу: https://legalitgroup.com/service/ohorona-praci-v-it-kompanii/.

ДОДАТКИ

Додаток А клас «AppDAO»

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

namespace Digital_Storehouse.Daos
{
    public class AppDAO : DatabaseConnection
    {

        public static void LoadCustomerPhoto(PictureBox picbox, int
customerId)
        {
            string sql = "SELECT PHOTO FROM CUSTOMERS WHERE CUSTOMER_ID =
@id;";
            using (SqlCommand command = new SqlCommand(sql, Conn))
            {
                command.Parameters.AddWithValue("@id", customerId);

                openConnectionIfClosed();

                object value = command.ExecuteScalar();
                Conn.Close();

                if (value != null)
                {
                    Image image = null;
                    byte[] data = (byte[])value;

                    if (data != null)
                    {
                        using (Stream ms = new MemoryStream())
                        {
                            ms.Write(data, 0, data.Length);
                            ms.Position = 0L;
                            image = new Bitmap(ms);
                            picbox.Image = image;
                        }
                    }
                }
            }
        }

        public static void DeleteCustomer(int customerId)
        {
            openConnectionIfClosed();

            using (var command = Conn.CreateCommand())
            {
```

```
                command.CommandText = "DELETE FROM CUSTOMERS WHERE
CUSTOMER_ID = @param";
                command.Parameters.AddWithValue("@param", customerId);
```

Продовження додатку А

```
                command.ExecuteNonQuery();
            }
            Conn.Close();
        }

        public static void DeleteOrder(int ordeId)
        {
            openConnectionIfClosed();

            using (var command = Conn.CreateCommand())
            {
                command.CommandText = "DELETE FROM ORDERS WHERE ORDER_ID =
@param";
                command.Parameters.AddWithValue("@param", ordeId);
                command.ExecuteNonQuery();
            }
            Conn.Close();
        }

        public static void DeleteProduct(int productId)
        {
            openConnectionIfClosed();

            using (var command = Conn.CreateCommand())
            {
                command.CommandText = "DELETE FROM PRODUCTS WHERE PRODUCT_ID
= @param";
                command.Parameters.AddWithValue("@param", productId);
                command.ExecuteNonQuery();
            }
            Conn.Close();
        }

        public static void DeleteRelationship(int idOrder)
        {
            openConnectionIfClosed();

            using (var command = Conn.CreateCommand())
            {
                command.CommandText = "DELETE FROM ORDERS_PRODUCTS WHERE
ID_ORDER = @param";
                command.Parameters.AddWithValue("@param", idOrder);
                command.ExecuteNonQuery();
            }
            Conn.Close();
        }
    }
}
```

Додаток Б клас «DatabaseConnection»

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Data.SqlClient;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Digital_Storehouse.Controllers;
using Digital_Storehouse.Models;

namespace Digital_Storehouse.Daos
{
    public class DatabaseConnection
    {
        public static SqlDataAdapter AdapterCustomer, AdapterProduct,
AdapterOrder, AdapterOrdersProducts;
        public static DataSet DatasetCustomer, DatasetProducts, DatasetOrder,
DatasetOrdersProducts;
        public static BindingSource BindsourceCustomer, BindsourceProducts,
BindsourceOrders, BindsourceOrdersProducts;
        public static SqlConnection Conn;


        public DatabaseConnection()
        {
            Connect();
        }
        public void Connect()
        {
            Conn = new SqlConnection
            {
                ConnectionString = Container.CONNECTION_STRING
            };
        }

        public void BindCustomerData(Dictionary<string, Label>
customerValueLabels, RichTextBox commentsRichTextbox, BindingNavigator
bindingNavigator)
        {
            openConnectionIfClosed();

            string query = "SELECT * FROM CUSTOMERS";
            AdapterCustomer = new SqlDataAdapter(query, Conn);
            DatasetCustomer = new DataSet();
            AdapterCustomer.Fill(DatasetCustomer, "Customers_Table");

            BindsourceCustomer = new BindingSource
            {
                DataSource = DatasetCustomer.Tables[0].DefaultView
            };
```

```
            customerValueLabels["CUSTOMER_ID"].DataBindings.Add(new
Binding("Text", BindsourceCustomer, "CUSTOMER_ID", true));
```

Продовження додатку Б

```
            customerValueLabels["LAST_NAME"].DataBindings.Add(new
Binding("Text", BindsourceCustomer, "LAST_NAME", true));
            customerValueLabels["FIRST_NAME"].DataBindings.Add(new
Binding("Text", BindsourceCustomer, "FIRST_NAME", true));
            customerValueLabels["BIRTH_DATE"].DataBindings.Add(new
Binding("Text", BindsourceCustomer, "BIRTH_DATE", true));
            customerValueLabels["AGE"].DataBindings.Add(new Binding("Text",
BindsourceCustomer, "AGE", true));
            customerValueLabels["AFM"].DataBindings.Add(new Binding("Text",
BindsourceCustomer, "AFM", true));
            customerValueLabels["DOY"].DataBindings.Add(new Binding("Text",
BindsourceCustomer, "DOY", true));
            customerValueLabels["ADDRESS"].DataBindings.Add(new
Binding("Text", BindsourceCustomer, "ADDRESS", true));
            customerValueLabels["CITY"].DataBindings.Add(new Binding("Text",
BindsourceCustomer, "CITY", true));
            customerValueLabels["PHONE_NUMBER"].DataBindings.Add(new
Binding("Text", BindsourceCustomer, "PHONE_NUMBER", true));
            commentsRichTextbox.DataBindings.Clear(); //TODO temporary bug
fix
            commentsRichTextbox.DataBindings.Add(new Binding("Text",
BindsourceCustomer, "COMMENTS", true));

            bindingNavigator.BindingSource = BindsourceCustomer;
            Conn.Close(); // Close Connection
        }
        public void BindOrderData(Dictionary<String, Label> orderValueLabels,
BindingNavigator bindingNavigator)
        {
            openConnectionIfClosed();
            string query = "SELECT * FROM ORDERS";
            AdapterOrder = new SqlDataAdapter(query, Conn);
            DatasetOrder = new DataSet();
            AdapterOrder.Fill(DatasetOrder, "Orders_Table");

            BindsourceOrders = new BindingSource
            {
                DataSource = DatasetOrder.Tables[0].DefaultView
            };

            orderValueLabels["ORDER_ID"].DataBindings.Add(new Binding("Text",
BindsourceOrders, "ORDER_ID", true));
            orderValueLabels["ORDER_DATE"].DataBindings.Add(new
Binding("Text", BindsourceOrders, "ORDER_DATE", true));
            orderValueLabels["CUSTOMER_ID_F"].DataBindings.Add(new
Binding("Text", BindsourceOrders, "CUSTOMER_ID", true));
            orderValueLabels["PAYMENT_METHOD"].DataBindings.Add(new
Binding("Text", BindsourceOrders, "PAYMENT_METHOD", true));
            orderValueLabels["DELIVERY_LOCATION"].DataBindings.Add(new
Binding("Text", BindsourceOrders, "DELIVERY_LOCATION", true));

            bindingNavigator.BindingSource = BindsourceOrders;
            Conn.Close(); // Close Connection
        }

        public void BindProductsData(Dictionary<String, Label>
productsValueLabels, BindingNavigator bindingNavigator)
        {
```

```
            openConnectionIfClosed();
```

Продовження додатку Б

```
            string query = "SELECT * FROM PRODUCTS";
            AdapterProduct = new SqlDataAdapter(query, Conn);
            DatasetProducts = new DataSet();
            AdapterProduct.Fill(DatasetProducts, "Products_Table");

            BindsourceProducts = new BindingSource
            {
                DataSource = DatasetProducts.Tables[0].DefaultView
            };
           productsValueLabels["PRODUCT_ID"].DataBindings.Add(new
Binding("Text", BindsourceProducts, "PRODUCT_ID", true));
            productsValueLabels["LABEL"].DataBindings.Add(new Binding("Text",
BindsourceProducts, "LABEL", true));
            productsValueLabels["CATEGORY"].DataBindings.Add(new
Binding("Text", BindsourceProducts, "CATEGORY", true));
            productsValueLabels["RESERVE"].DataBindings.Add(new
Binding("Text", BindsourceProducts, "RESERVE", true));
            productsValueLabels["SELLING_PRICE"].DataBindings.Add(new
Binding("Text", BindsourceProducts, "SELLING_PRICE", true));
            productsValueLabels["MANUFACTURER"].DataBindings.Add(new
Binding("Text", BindsourceProducts, "MANUFACTURER", true));

            bindingNavigator.BindingSource = BindsourceProducts;
            Conn.Close(); // Close Connection
        }

        public void BindOrdersProductsData(Dictionary<String, Label>
ordersProductsValueLabels, BindingNavigator bindingNavigator)
        {
            openConnectionIfClosed();

            string query = "SELECT * FROM ORDERS_PRODUCTS";
            AdapterOrdersProducts = new SqlDataAdapter(query, Conn);
            DatasetOrdersProducts = new DataSet();
            AdapterOrdersProducts.Fill(DatasetOrdersProducts,
"OrdersProducts_Table");

            BindsourceOrdersProducts = new BindingSource
            {
                DataSource = DatasetOrdersProducts.Tables[0].DefaultView
            };

            ordersProductsValueLabels["ORDER_ID_F"].DataBindings.Add(new
Binding("Text", BindsourceOrdersProducts, "ID_ORDER", true));
            ordersProductsValueLabels["PRODUCT_ID_F"].DataBindings.Add(new
Binding("Text", BindsourceOrdersProducts, "ID_PRODUCT", true));
            ordersProductsValueLabels["AMOUNT"].DataBindings.Add(new
Binding("Text", BindsourceOrdersProducts, "AMOUNT", true));

            bindingNavigator.BindingSource = BindsourceOrdersProducts;
            Conn.Close(); // Close Connection
        }
        public void BindOrdersHistoryDataGrid(DataGridView
ordersHistory_dataGridView, string customerId)
        {
            int customerID;
            if (! int.TryParse(customerId, out customerID)){
                return;
```

```
            }

            openConnectionIfClosed();

            string query =
            "SELECT LAST_NAME, PAYMENT_METHOD, LABEL AS PRODUCT,
SELLING_PRICE AS UNIT_PRICE, AMOUNT FROM CUSTOMERS "+
            "INNER JOIN ORDERS "+
                "ON CUSTOMERS.CUSTOMER_ID = ORDERS.CUSTOMER_ID " +
            "INNER JOIN ORDERS_PRODUCTS "+
                "ON ORDERS_PRODUCTS.ID_ORDER = ORDERS.ORDER_ID "+
            "INNER JOIN PRODUCTS "+
                "ON PRODUCTS.PRODUCT_ID = ORDERS_PRODUCTS.ID_PRODUCT "+
            "WHERE CUSTOMERS.CUSTOMER_ID = "+customerId;

            try
            {
                SqlDataAdapter adapter = new SqlDataAdapter(query, Conn);
                DataSet dataset = new DataSet();
                adapter.Fill(dataset);
                BindingSource bindingSource = new BindingSource();
                bindingSource.DataSource = dataset.Tables[0].DefaultView;
                ordersHistory_dataGridView.DataSource = bindingSource;
            }
            catch (SqlException e)
            {
                ViewMessages.ExceptionOccured(e);
            }
            Conn.Close(); // Close Connection
        }
        public void BindProductsHistoryDataGrid(DataGridView
productsHistoryDataGridView, string productId)
        {
            int productID;
            if (!int.TryParse(productId, out productID)){
                return;
            }
            openConnectionIfClosed();

            string query =
                "SELECT LABEL AS PRODUCT, LAST_NAME AS ORDERED_BY,
PAYMENT_METHOD, SELLING_PRICE AS UNIT_PRICE, AMOUNT FROM ORDERS " +
                "INNER JOIN ORDERS_PRODUCTS " +
                    "ON ORDERS_PRODUCTS.ID_ORDER = ORDERS.ORDER_ID " +
                "INNER JOIN PRODUCTS " +
                    "ON PRODUCTS.PRODUCT_ID = ORDERS_PRODUCTS.ID_PRODUCT " +
                "INNER JOIN CUSTOMERS " +
                    "ON CUSTOMERS.CUSTOMER_ID = ORDERS.CUSTOMER_ID " +
                "WHERE ORDERS.ORDER_ID = " + productId;

            try
            {
                SqlDataAdapter adapter = new SqlDataAdapter(query, Conn);
                DataSet dataset = new DataSet();
                adapter.Fill(dataset);
                BindingSource bindingSource = new BindingSource();
                bindingSource.DataSource = dataset.Tables[0].DefaultView;
                productsHistoryDataGridView.DataSource = bindingSource;
            }
            catch (SqlException e)
            {
                ViewMessages.ExceptionOccured(e);
```

Продовження додатку Б

```
        }
        Conn.Close(); // Close Connection  }
    public static DataSet GetDataSet()
    {
        openConnectionIfClosed();

        string query = "SELECT * FROM CUSTOMERS";
        SqlDataAdapter adapter = new SqlDataAdapter(query, Conn);
        DataSet dataset = new DataSet();

        adapter.Fill(dataset, "Test_Table");

        return dataset;
    }

    public static BindingSource getBindSourceCustomer()
    {
        return BindsourceCustomer;
    }

    public static BindingSource getBindSourceProducts()
    {
        return BindsourceProducts;
    }

    public static BindingSource getBindSourceOrders()
    {
        return BindsourceOrders;
    }
    public static BindingSource getBindSourceOrdersProducts()
    {
        return BindsourceOrdersProducts;
    }
    public static byte[] GetImageBytesFromPath(string path)   {
        Image img = Image.FromFile(path);
        MemoryStream tmpStream = new MemoryStream();
        img.Save(tmpStream, ImageFormat.Png); // change to other format
        tmpStream.Seek(0, SeekOrigin.Begin);
        byte[] imgBytes = new byte[256000];
        tmpStream.Read(imgBytes, 0, 256000);
        return imgBytes;   }
  public static byte[] GetImageBytesFromImage(Image image)
    {
        MemoryStream tmpStream = new MemoryStream();
        image.Save(tmpStream, ImageFormat.Png); // change to other format
        tmpStream.Seek(0, SeekOrigin.Begin);
        byte[] imgBytes = new byte[256000];
        tmpStream.Read(imgBytes, 0, 256000);
        return imgBytes;
    }

    public static void openConnectionIfClosed()
    {
        if (Conn.State == ConnectionState.Closed)
        {
            Conn.Open();
        }
    }
}
```

Додаток В клас «AppController»

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Digital_Storehouse.Daos;
using Digital_Storehouse.Views;

namespace Digital_Storehouse.Controllers
{
    public class AppController
    {
        private static DatabaseConnection db = new DatabaseConnection();

        public static void DeleteCustomer(Dictionary<string, Label>
customerValueLabels, RichTextBox commentsRichTextBox, BindingNavigator
bindingNavigatorCustomer)
        {
            if
(!customerCanBeDeleted(customerValueLabels["CUSTOMER_ID"].Text))
            {
                ViewMessages.CannotDeleteCustomer();
                return;
            }

            if (ViewMessages.DeleteCustomerDialog() == DialogResult.No)
            {
                return;
            }

AppDAO.DeleteCustomer(Int32.Parse(customerValueLabels["CUSTOMER_ID"].Text));
            foreach (KeyValuePair<string, Label> entry in
customerValueLabels)
            {
                entry.Value.DataBindings.Clear();
            }
            db.BindCustomerData(customerValueLabels, commentsRichTextBox,
bindingNavigatorCustomer);
        }

        public static void DeleteOrder(Dictionary<string, Label>
orderValueLabels, BindingNavigator bindingNavigatorOrder)
        {
            if (! orderCanBeDeleted(orderValueLabels["ORDER_ID"].Text))
            {
                ViewMessages.CannotDeleteOrder();
                return;
            }

            if (ViewMessages.DeleteOrderDialog() == DialogResult.No)
            {
                return;
            }

AppDAO.DeleteOrder(Int32.Parse(orderValueLabels["ORDER_ID"].Text));
            foreach (KeyValuePair<string, Label> entry in orderValueLabels)
            {
```

Продовження додатку В

```
                entry.Value.DataBindings.Clear();
            }
            db.BindOrderData(orderValueLabels, bindingNavigatorOrder);
        }

        public static void DeleteProduct(Dictionary<string, Label>
productValueLabels, BindingNavigator bindingNavigatorProduct)
        {
            if (!productCanBeDeleted(productValueLabels["PRODUCT_ID"].Text))
            {
                ViewMessages.CannotDeleteProduct();
                return;
            }

            if (ViewMessages.DeleteProductDialog() == DialogResult.No)
            {
                return;
            }

AppDAO.DeleteProduct(Int32.Parse(productValueLabels["PRODUCT_ID"].Text));
            foreach (KeyValuePair<string, Label> entry in productValueLabels)
            {
                entry.Value.DataBindings.Clear();
            }
            db.BindProductsData(productValueLabels, bindingNavigatorProduct);
        }

        public static void DeleteRelationship(Dictionary<string, Label>
ordersProductsValueLabels, BindingNavigator bindingNavigatorOrdersProducts)
        {
            if (ViewMessages.DeleteRelationshipDialog() == DialogResult.No)
            {
                return;
            }

AppDAO.DeleteRelationship(Int32.Parse(ordersProductsValueLabels["ORDER_ID_F"]
.Text));
            foreach (KeyValuePair<string, Label> entry in
ordersProductsValueLabels)
            {
                entry.Value.DataBindings.Clear();
            }
            db.BindOrdersProductsData(ordersProductsValueLabels,
bindingNavigatorOrdersProducts);
        }

        public static void syncCustomersTab(BindingSource customerBindSource,
Dictionary<String, Label> customerValueLabels,
            PictureBox pictureBox, Button deleteCustomerButton, Button
updateCustomButton)
        {
            if (customerBindSource == null)
            {
                // Binding Source has not yet loaded - cancel
                return;
            }

            if (customerBindSource.Count == 0)
            {
```

Продовження додатку В

```
                foreach (KeyValuePair<String, Label> entry in
customerValueLabels)
                {
                    entry.Value.Text = "XXX";
                }
                pictureBox.Image = null;
                deleteCustomerButton.Enabled = false;
                updateCustomButton.Enabled = false;
                return;
            }
            deleteCustomerButton.Enabled = true;
            updateCustomButton.Enabled = true;
            AppDAO.LoadCustomerPhoto(pictureBox,
Int32.Parse(customerValueLabels["CUSTOMER_ID"].Text));
        }

        public static void syncProductsTab(BindingSource productsBindSource,
            Dictionary<String, Label> productsValueLabels,
            Button deleteProductButton, Button updateProductButton)
        {
            if (productsBindSource == null)
            {
                // Binding Source has not yet loaded - cancel
                return;
            }

            if (productsBindSource.Count == 0)
            {
                foreach (KeyValuePair<String, Label> entry in
productsValueLabels)
                {
                    entry.Value.Text = "XXX";
                }
                deleteProductButton.Enabled = false;
                updateProductButton.Enabled = false;
                return;
            }
            deleteProductButton.Enabled = true;
            updateProductButton.Enabled = true;
        }


        public static void syncOrdersTab(BindingSource
ordersBindSource,Dictionary<String, Label> ordersValueLabels,
                                         Button deleteOrderButton, Button
updateOrderButton)
        {
            if (ordersBindSource == null)
            {
                // Binding Source has not yet loaded - cancel
                return;
            }

            if (ordersBindSource.Count == 0)
            {
                foreach (KeyValuePair<String, Label> entry in
ordersValueLabels)
                {
                    entry.Value.Text = "XXX";
                }
```

Продовження додатку В

```
                deleteOrderButton.Enabled = false;
                updateOrderButton.Enabled = false;
                return;
            }
            deleteOrderButton.Enabled = true;
            updateOrderButton.Enabled = true;
        }

        public static void syncOrdersProductsTab(BindingSource
ordersProductsBindSource,
            Dictionary<String, Label> ordersProductsValueLabels,
            Button deleteOrdersProductsButton, Button
updateOrdersProductsButton)
        {
            if (ordersProductsBindSource == null)
            {
                // Binding Source has not yet loaded - cancel
                return;
            }

            if (ordersProductsBindSource.Count == 0)
            {
                foreach (KeyValuePair<String, Label> entry in
ordersProductsValueLabels)
                {
                    entry.Value.Text = "XXX";
                }
                deleteOrdersProductsButton.Enabled = false;
                updateOrdersProductsButton.Enabled = false;
                return;
            }
            deleteOrdersProductsButton.Enabled = true;
            updateOrdersProductsButton.Enabled = true;
        }



        public static void OpenNewCustomerForm(BindingNavigator
bindingNavigatorCustomer)
        {
            NewCustomer form = new NewCustomer(bindingNavigatorCustomer);
            form.Show();
        }

        public static void OpenUpdateCustomerForm(Dictionary<string, Label>
customerValueLabels, PictureBox pictureBox,
                                          RichTextBox commentsRichTextBox,
BindingNavigator bindingNavigatorCustomers)
        {
            UpdateCustomer form = new UpdateCustomer(customerValueLabels,
pictureBox, commentsRichTextBox, bindingNavigatorCustomers);
            form.Show();
        }

        public static void OpenUpdateOrderForm(Dictionary<string, Label>
orderValueLabels, BindingNavigator bindingNavigatorCustomers)
        {
            UpdateOrder form = new UpdateOrder(orderValueLabels,
bindingNavigatorCustomers);
            form.Show();
```

Продовження додатку В

```
        }

        public static bool orderCanBeDeleted(string orderId)
        {
            foreach (var id in
NewOrdersProductsDAO.getOrderIdsInOrdersProductsTable())
            {
                if (id.Equals(orderId))
                {
                    return false;
                }
            }
            return true;
        }

        public static bool productCanBeDeleted(string productId)
        {
            foreach (var id in
NewOrdersProductsDAO.getProductIdsInOrdersProductsTable())
            {
                if (id.Equals(productId))
                {
                    return false;
                }
            }
            return true;
        }

        public static bool customerCanBeDeleted(string customerId)
        {
            foreach (var id in
NewOrderDAO.getCustomerIdsInOrdersProductsTable())
            {
                if (id.Equals(customerId))
                {
                    return false;
                }
            }
            return true;
        }

        public static void createCustomersHistoryDataGrid(DataGridView
ordersHistory_dataGridView, ComboBox customerIds_combobox,
DataGridViewTextBoxColumn totalColumn, Label ordersHistoryTotal_label)
        {
            db.BindOrdersHistoryDataGrid(ordersHistory_dataGridView,
customerIds_combobox.GetItemText(customerIds_combobox.SelectedItem));

            if (!ordersHistory_dataGridView.Columns.Contains(totalColumn))
            {
                ordersHistory_dataGridView.Columns.Add(totalColumn);
            }

            int index =
ordersHistory_dataGridView.Columns.IndexOf(totalColumn);

            float grandTotal = 0;

            for (int i = 0; i < ordersHistory_dataGridView.RowCount; i++)
            {
```

Продовження додатку В

```
                float total =
float.Parse(ordersHistory_dataGridView.Rows[i].Cells["UNIT_PRICE"].Value.ToSt
ring()) *

float.Parse(ordersHistory_dataGridView.Rows[i].Cells["AMOUNT"].Value.ToString
());

                ordersHistory_dataGridView.Rows[i].Cells[index].Value =
total;
                grandTotal += total;
            }
            ordersHistoryTotal_label.Text = Convert.ToString(grandTotal,
CultureInfo.InvariantCulture);

            foreach (DataGridViewColumn column in
ordersHistory_dataGridView.Columns)
            {
                column.SortMode = DataGridViewColumnSortMode.NotSortable;
            }
        }

        public static void exportDataForPrinter(DataGridView
ordersHistory_dataGridView, System.Drawing.Printing.PrintPageEventArgs e)
        {
            Bitmap bm = new Bitmap(ordersHistory_dataGridView.Width,
ordersHistory_dataGridView.Height);
            ordersHistory_dataGridView.DrawToBitmap(bm, new Rectangle(0, 0,
ordersHistory_dataGridView.Width, ordersHistory_dataGridView.Height));
            e.Graphics.DrawImage(bm, 0, 0);
        }
    }
}
```

Додаток Г клас «UpdateProductController»

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

namespace Digital_Storehouse.Controllers
{
    class UpdateProductController
    {
        private static DatabaseConnection db = new DatabaseConnection();

        public static bool validateFields(Dictionary<String, TextBox>
productValueLabels)
        {
            if (productValueLabels["SELLING_PRICE"].Text.Equals("") ||
productValueLabels["LABEL"].Text.Equals("")
                    || productValueLabels["RESERVE"].Text.Equals(""))
            {
                ViewMessages.FillRequiredFields();
                return false;
            }
```

```
            int n;
            if (!int.TryParse(productValueLabels["RESERVE"].Text, out n))
            {
                ViewMessages.ReserveNotValid();
                return false;
            }

            float k;
            if (!float.TryParse(productValueLabels["SELLING_PRICE"].Text, out
k))
            {
                ViewMessages.SellingPriceNotValid();
                return false;
            }

            return true;
        }



        public static void updateProduct(Dictionary<String, TextBox>
productValueLabels, BindingNavigator bindingNavigatorProducts,
                                    UpdateProduct updateProductForm, int
productId)
        {
            if (!validateFields(productValueLabels)){
                return;
            }

            try
            {
                int currentPage =
bindingNavigatorProducts.BindingSource.Position;
```

Продовження додатку Г

```
UpdateProductDAO.updateProduct(productValueLabels["LABEL"].Text,
productValueLabels["CATEGORY"].Text,
int.Parse(productValueLabels["RESERVE"].Text),

float.Parse(productValueLabels["SELLING_PRICE"].Text),
productValueLabels["MANUFACTURER"].Text, productId);


                foreach (KeyValuePair<string, Label> entry in
App.GetProductLabels())
                {
                    entry.Value.DataBindings.Clear();
                }

                db.BindProductsData(App.GetProductLabels(),
bindingNavigatorProducts);

                bindingNavigatorProducts.BindingSource.Position =
currentPage;

                // Updated!
                updateProductForm.Close();
            }
            catch (SqlException e)
            {
                ViewMessages.ExceptionOccured(e);
```

```
                }
            }


        }
    }
```

ДОДАТОК Д

Інформаційна підсистема управління роботою підприємства

Виконала: студентка групи КН-42с Воленко Т.О.

Керівник: д.т.н., доц. Цюцюра М.І.

1

Актуальність теми

Ведення складського обліку автоматизованих шляхом, дозволяє зробити роботу складу «прозорою» і мінімізувати тимчасові витрати на проведення різних складських операцій.

2

Об’єкт дослідження

Робота малого підприємства яке займається реалізаціє продукції та обліком замовлень і клієнтів.

3

Мета

Реалізація системи для працівників підприємства. Система буде підтримувати можливості ведення звітності, та збереження даних в хмарі, дані які зберігаються: товари, замовлення, клієнти, категорії товарів.

4

Завдання для досягнення мети

- Вивчення предметної області;
- Розглянути ряд існуючих розробок для вирішення мети;
- Обрати спосіб здійснення розробки;
- Розробка інформаційної системи;
- Аналіз бізнес доцільності розробки;
- Аналіз охорони праці та вимог безпеки під час розробки;

5

Вивчення предметної області

Сьогодні все більше організацій і компаній інвестують у програмне забезпечення для управління бізнесом, щоб виконувати завдання, прогнозувати ризики та підвищувати загальну ефективність. Таким чином, не буде помилкою сказати, що інструменти управління бізнесом пройшли довгий шлях і постійно покращують ситуацію для будь-кого, хто займається певним бізнесом.

Існуючі розробки

7

Існуючі розробки

8

Постановка задачі

•Основними функціями системи є:

1. Додання категорій товарів;
2. Додавання товарів;
3. Додання клієнтів;
4. Редагування та видалення інформації про товари;
5. Редагування та видалення інформації про категорії;

9

Постановка задачі

- Вимоги для додатку:

1. Повинен в автоматичному режимі оновлювати інформацію у базі даних;
2. Повинен мати можливість на доопрацювання;
3. Має бути надійним, та захищати інформацію з бази даних;

10

Вимоги до ПК користувача

Процесор Intel pentium 3.1GHz
Не менше 1 ram оперативної пам’яті;
OS Windows 10
Net framework 4.6

11

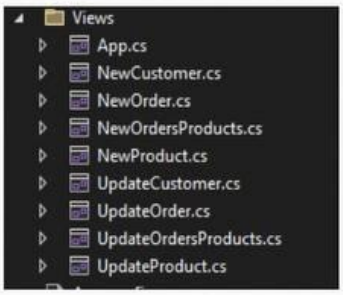
Інструменти реалізації:

- Microsoft Visual Studio 2022
- Microsoft SQL Server.
- Мова програмної розробки C#.

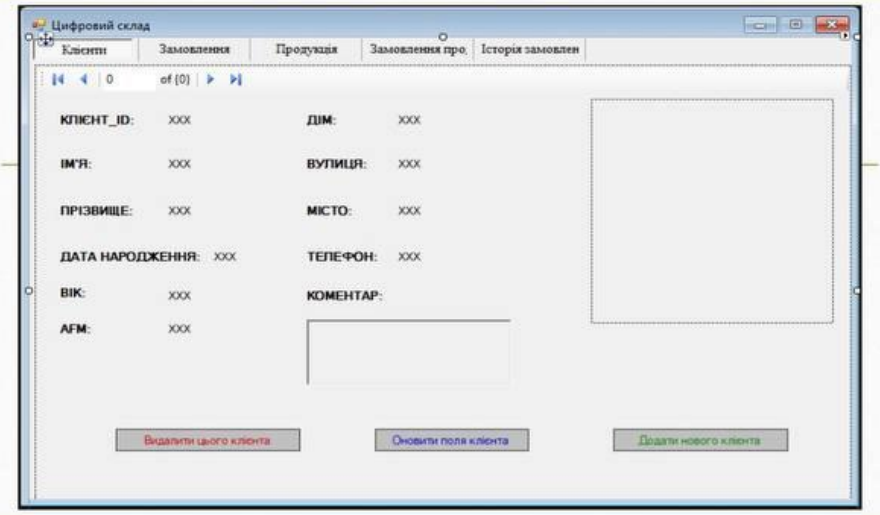
Microsoft SQL Server

12

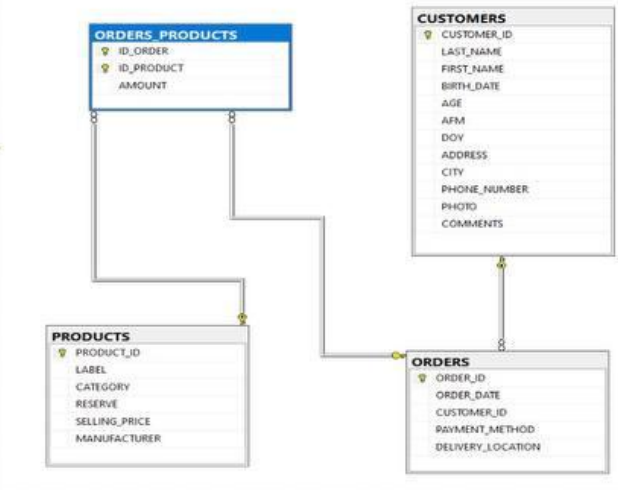
Стек «View», класи інтерфейсу користувача



13



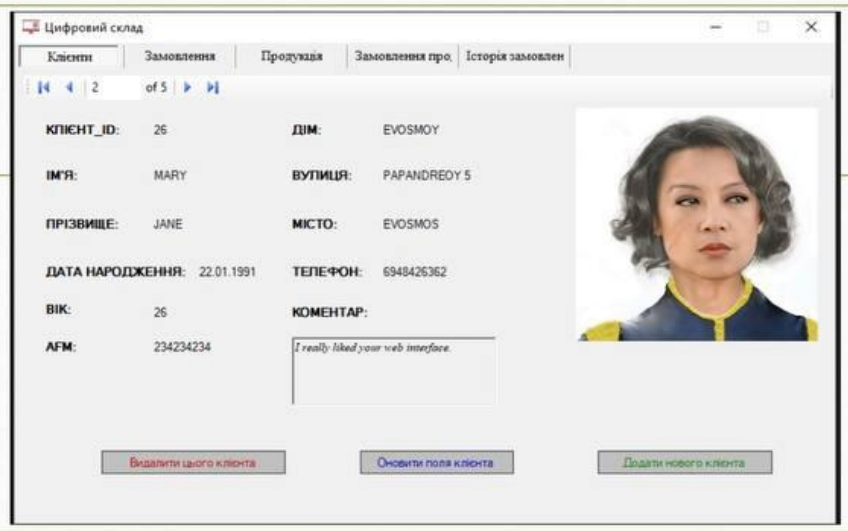
14



15

```
1 reference
class UpdateOrdersProductsDAO : DatabaseConnection
{
    1 reference
    public static void updateOrdersProducts(float amount, int idOrder, int idProduct)
    {
        string query = "UPDATE ORDERS_PRODUCTS SET AMOUNT = @AMOUNT_P WHERE ID_ORDER = @IDORDER_P AND ID_PRODUCT = @IDPRODUCT_P";

        SqlCommand command = new SqlCommand(query, Conn);

        command.Parameters.AddWithValue("@AMOUNT_P", amount);
        command.Parameters.AddWithValue("@IDORDER_P", idOrder);
        command.Parameters.AddWithValue("@IDPRODUCT_P", idProduct);

        openConnectionIfClosed();

        try
        {
            command.ExecuteNonQuery();
            Conn.Close();
        }
        catch (SqlException e)
        {
            Conn.Close();
            Controllers.ViewMessages.ExceptionOccured(e);
        }
    }
}
```

16



17

Економічна доцільність

- В результаті виконаних розрахунків техніко-економічних показників загальна сума інвестицій на створення проекту складає 4512,77 грн. термін окупності становить 1,22 року.

$$Ток = \frac{4512,77}{3699,49} = 1,22$$

18

Розроблена інформаційна система забезпечує:

- Ведення обліку руху товарів на складі;
- Повний контроль всіх складських операцій;
- швидкий пошук і підбір даних про можливі характеристики номенклатури;
- Прискорення процесу завершення замовлення та відвантаження;

19

Дякую за увагу!

Слава Україні!

20