МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
Факультет мехатроніки та комп'ютерних технологій
Кафедра комп'ютерних наук

ДИПЛОМНА БАКАЛАВРСЬКА РОБОТА

на тему

РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ТОРГІВЕЛЬНОГО ПІДПРИЄМСТВА ЗА ДОПОМОГОЮ CASE-СИСТЕМИ

Виконала: студентка групи БІТ-2-19
спеціальності 122 Комп'ютерні науки
Анастасія МАРТИШЕВА
Науковий керівник
Вікторія РЕЗАНОВА

Рецензент
Сергій КРАСНИТСЬКИЙ

Київ 2023

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет мехатроніки та комп’ютерних технологій

Кафедра комп’ютерних наук

Спеціальність 122 Комп’ютерні науки

Освітня програма Комп’ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп’ютерних наук
_____ Володимир ЩЕРБАНЬ

«_____» _____ 20 __ р.

ЗАВДАННЯ

НА ДИПЛОМНУ БАКАЛАВРСЬКУ РОБОТУ

студентці
Мартишевій Анастасії Павлівні

1. Тема роботи: Розробка інформаційної системи торгівельного підприємства за допомогою CASE-системи
Науковий керівник роботи Резанова Вікторія Георгіївна, доцент кафедри комп'ютерних наук, затверджені наказом КНУТД від “08” листопада 2022 року 224-уч.
2. Строк подання студентом дипломної роботи: 05.06.2023 р.
3. Вихідні дані до дипломної бакалаврської роботи: Розробки кафедри комп’ютерних наук та технологій, рекомендована література, додатки
4. Зміст дипломної роботи: Розділ 1. Аналіз предметної області, Розділ 2. Проектування інформаційнї системи, Розділ 3. Програмна реалізація.
5. Дата видачі завдання: 06.02.2023р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної бакалаврської роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	10.05.2023	
2	Розділ 1. Аналіз предметної області	11.05.2023	
3	Розділ 2. Проектування інформаційні системи	12.05.2023	
4	Розділ 3. Програмна реалізація	20.05.2023	
5	Висновки	21.05.2023	
6	Оформлення дипломної роботи (проекту) (чистовий варіант)	25.05.2023	
7	Здача дипломної роботи (проекту) на кафедру для рецензування (за 14 днів до захисту)		
8	Перевірка кваліфікаційної роботи на наявність текстових співпадінь та помилок		
9	Подання дипломної роботи (проекту) на затвердження завідувачу кафедри (за 7 днів до захисту)		

Студентка _____ Анастасія МАРТИШЕВА

Науковий керівник роботи Вікторія РЕЗАНОВА

Рецензент Сергій КРАСНИТСЬКИЙ

АНОТАЦІЯ

Мартишева А.П. Розробка інформаційної системи торгівельного підприємства за допомогою CASE-системи.

Дипломна робота за спеціальністю 122 - «Комп'ютерні науки». – Київский національний університет технологій та дизайну, Київ, 2023 рік.

Метою дипломної роботи є розробка інформаційної системи, для реалізації якої використовується CASE-система. У рамках цієї роботи проведений детальний аналіз потреб торгівельних підприємств та виявлені основні проблеми, з якими вони зіштовхуються в процесі своєї роботи. У процесі розробки системи була використана CASE-система, яка надає широкий спектр інструментів для моделювання, проектування та розробки програмного забезпечення. Для створення програмного забезпечення була використана мова програмування Python.

Для реалізації мети стоять такі завдання:

- Аналіз предметної області
- Проектування інформаційної системи
- Розробка програмного забезпечення

Ключові слова: Інформаційна система, CASE-системи, торгівельні підприємства.

ANNOTATION

Martysheva A.P. Development of the information system of the trading company using the CASE-system.

Thesis in specialty 122 - "Computer Science". - Kyiv National University of Technology and Design, Kyiv, 2023.

The aim of the thesis is to develop an information, for the implementation of which the CASE-system is used. The research includes a detailed analysis of the needs of trading enterprises and identifies the main challenges they face in their operations. The CASE tool, which offers a wide range of modeling, designing, and software development tools, was utilized in the system development process. The programming language Python was employed for software development.

The following tasks have been set to achieve the objective:

- Analysis of the subject area
- Designing the information system
- Software development.

Keywords: Information system, CASE-systems, trading company

ЗМІСТ

ВСТУП

Сучасні торгівельні підприємства все більше відчувають необхідність у використанні новітніх інформаційних технологій для оптимізації своєї діяльності та підвищення ефективності роботи. У сучасному змінному світі економіки, де конкуренція постійно зростає, використання інформаційних систем стає невід'ємною складовою ефективного управління торгівельними підприємствами.

Інформаційні системи відіграють важливу роль у допомозі торгівельним підприємствам у зборі, зберіганні, обробці та аналізі великих обсягів даних, пов'язаних з різними аспектами їх діяльності. Ці дані можуть стосуватись продажів, запасів, клієнтів, постачальників, фінансової інформації та інших факторів. Завдяки інформаційним системам можна автоматизувати багато процесів, які раніше вимагали значних ресурсів та ручної роботи.

Розробка інформаційної системи для торгівельного підприємства є актуальною задачею, яка вимагає поєднання знань з області технологій програмування, управління торгівельною діяльністю та дизайну інтерфейсу користувача. Головна мета розробки такої системи полягає в автоматизації процесів збору, обробки та зберігання інформації про діяльність підприємства. Це дозволить зменшити витрати часу та зусиль на виконання рутинних завдань і підвищити якість управлінських рішень.

Для досягнення поставленої мети обрано CASE-систему, що дозволяє візуалізувати процес розробки та зменшити кількість помилок. При проектуванні системи були враховані особливості роботи торгівельного підприємства, такі як управління складом, замовленнями, клієнтською базою та звітністю.

Основні завдання, які потрібно виконати в рамках дипломної роботи, включають:

1. Аналіз предметної області.

2. Аналіз діяльності підприємства та визначення вимог до системи.

3. Вибір та використання CASE-системи для моделювання системи.

4. Розробка моделей системи.

5. Розробка програми.

Об'єктом дослідження є розробка інформаційної системи для торгівельного підприємства з використанням CASE-системи. Предметом дослідження є детальний аналіз та розробка інформаційної системи для торгівельного підприємства з використанням CASE-системи.

Дослідження передбачає докладний аналіз потреб та вимог торгівельного підприємства щодо інформаційної системи, включаючи управління складом, замовленнями, продажами, фінансами та іншими ключовими аспектами діяльності підприємства. Крім того, дослідження передбачає вивчення різних методів та інструментів розробки інформаційних систем та аналіз їх переваг та недоліків. На основі цього аналізу буде обрана відповідна CASE-система, яка дозволить ефективно підтримувати процес розробки.

Головною частиною дослідження буде розробка самої інформаційної системи за допомогою обраної CASE-системи. Для цього будуть розроблені моделі системи, які враховують вимоги торгівельного підприємства та його особливості. Особлива увага буде приділена ефективності, надійності та зручності користування системою.

Усі ці завдання спрямовані на створення інформаційної системи, яка допоможе торгівельному підприємству забезпечити більш ефективне управління своєю діяльністю та забезпечити конкурентну перевагу на ринку. Виконання дипломної роботи дозволить отримати глибокі знання з області розробки інформаційних систем та їх впровадження в торгівельній сфері.

Дипломна робота складається зі вступу, 1 розділу, 2 розділу, 3 розділу, висновків, списку використаних джерел (30 найменувань) та додатків. Загальний обсяг дипломної роботи 49 сторінок комп’ютерного тексту (без додатків).

Розділ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Огляд торгівельних підприємств та аналіз поточних проблем

У сучасному торгівельному секторі спостерігаються ряд змін, що впливають на організацію та функціонування підприємств. Одна з головних тенденцій - це зростання конкуренції. З появою нових гравців на ринку та зміною покупців, торговельні підприємства стикаються з необхідністю швидкого реагування на зміни у попиті та вимогах споживачів.

Одною з важливих змін є зростання ролі онлайн-торгівлі та електронної комерції. Завдяки розвитку Інтернету, покупці мають доступ до широкого асортименту товарів та можливість зробити покупки в будь-який час. Це вимагає від торговельних підприємств наявності ефективних інформаційних систем, які забезпечують оптимальну організацію та управління процесами продажу, складськими запасами та логістикою.

Також варто зазначити зростання значення аналізу даних для торговельних підприємств. Збільшення обсягів даних, що генеруються в процесі торгівлі, вимагає використання аналітичних інструментів для отримання цінної інформації про споживачів, тенденції ринку та ефективність діяльності підприємства. Інформаційна система, розроблена за допомогою CASE-системи, може забезпечити збір, аналіз та використання даних для прийняття обґрунтованих рішень.

Аналізуючи предметну область, необхідно детально вивчити різні аспекти торговельного сектору, включаючи типи торговельних підприємств (роздрібні магазини, супермаркети, гіпермаркети, онлайн-платформи тощо), їх організаційну структуру, процеси закупівлі та управління запасами, маркетинг та рекламу, аналіз конкурентів, аналітику продажів та звітність.

Аналіз предметної області, включаючи огляд торгівельних підприємств, їх структури, організації та функціонування, є ключовим етапом у розробці інформаційної системи для торговельного підприємства з використанням CASE-системи. Цей аналіз надає конкретну інформацію для визначення вимог до системи, виокремлення основних функціональних

модулів та визначення оптимального рішення, що відповідає сучасним вимогам торговельного сектору.

Однією з головних проблем, яка виникає у торгівельних підприємствах, є управління запасами. Недостатня точність та ефективність управління запасами можуть спричиняти перевищення або недостатність товарів на складі, що негативно впливає на здатність підприємства задовольняти потреби клієнтів та оптимізувати витрати. Застосування інформаційної системи може допомогти вирішити цю проблему шляхом автоматизації процесу управління запасами, включаючи моніторинг запасів, прогнозування попиту, планування закупівель та оптимізацію складського управління.

Іншою важливою проблемою є управління продажами. Торговельним підприємствам часто важко ефективно відстежувати та керувати процесами продажу. Недостатня організація інформації про клієнтів, замовлення та продажі може призводити до втрати можливостей та недосягнення максимального прибутку. Впровадження інформаційної системи дозволить автоматизувати процеси продажу, включаючи управління клієнтською базою даних, обробку замовлень та моніторинг реалізації товарів. Це сприятиме покращенню обслуговування клієнтів, підвищенню ефективності продажів та збільшенню доходів підприємства.

Логістика також є важливою сферою, де торгівельні підприємства зіштовхуються з проблемами. Відсутність оптимального планування та управління логістичними процесами може призводити до затримок у доставці товарів, надмірного запасу або недостатньої наявності товарів на складі. Інформаційна система сприятиме ефективному управлінню логістичними процесами, включаючи моніторинг руху товарів, оптимізацію маршрутів доставки та планування розподілу запасів. Це дозволить зменшити затрати на логістику та поліпшити сервіс для клієнтів.

Однією з ключових проблем є звітність. Торгівельним підприємствам потрібно систематично виробляти звіти про продажі, фінансовий стан та аналітичні дані для прийняття обґрунтованих управлінських рішень.

Інформаційна система дозволить автоматизувати процес формування звітів, забезпечити точність та доступність інформації та покращити аналітичні можливості. Це сприятиме зростанню ефективності управління підприємством та дозволить приймати обґрунтовані стратегічні рішення.

Аналізуючи ці поточні проблеми, що стикаються торгівельні підприємства, та виявляючи їх потенційні рішення, можна виправдати впровадження інформаційної системи, розробленої за допомогою CASE-системи. Це допоможе підприємствам оптимізувати управління запасами, покращити планування та контроль продажів, оптимізувати логістичні процеси та поліпшити звітність. В результаті, торгівельні підприємства зможуть ефективно функціонувати в сучасному конкурентному середовищі та досягати стабільного росту та успіху.

1.2. Аналіз існуючих інформаційних систем та вимоги до інформаційної системи

Під час аналізу існуючих інформаційних систем, які використовуються торгівельними підприємствами, розглядалися різні аспекти їх функціональності, переваг та недоліків. Передбачалося вивчення ступеня задоволення потреб торговельних підприємств цими системами.

Один з аспектів, що був розглянутий, це система управління відносинами з клієнтами (CRM). Ця система дозволяє ефективно керувати взаємодією з клієнтами, створюючи цілісну базу даних клієнтів і покращуючи рівень обслуговування. Однак, CRM фокусується переважно на аспекті взаємодії з клієнтами, тоді як інші аспекти управління торговельним підприємством можуть залишатись недостатньо покритими.

Ще одна розглянута система - це система управління ланцюгом постачання (SCM). SCM спрямована на керування всіма етапами постачального ланцюга та оптимізацію процесів постачання товарів. Вона дозволяє планувати, координувати та оптимізувати рух товарів від постачальників до кінцевих споживачів. Однак, деякі системи SCM можуть

бути складними у використанні та вимагати значних фінансових та часових вкладень у впровадження та налаштування.

Третя розглянута система - це система управління ресурсами підприємства (ERP). ERP є комплексною системою, яка об'єднує всі аспекти діяльності підприємства. Вона надає інтегроване управління всіма процесами торговельного підприємства і сприяє покращенню координації між відділами та оптимізації роботи компанії. Однак, впровадження системи ERP може бути складним та вимагати значних інвестицій, а також великих зусиль для навчання персоналу.

Проведений аналіз показав, що кожна з розглянутих інформаційних систем має свої переваги та недоліки. CRM дозволяє зосередитися на взаємодії з клієнтами, SCM - на керуванні ланцюгом постачання, а ERP - на комплексному управлінні підприємством. Однак, ні одна з цих систем не забезпечує повного вирішення усіх проблем, з якими зіштовхуються торговельні підприємства.

На основі проведеного аналізу виявляється необхідність розробки нової інформаційної системи, яка б поєднала переваги цих існуючих систем та відповідала потребам торговельних підприємств. Розробка такої системи за допомогою CASE-системи дозволить раціоналізувати процеси розробки, забезпечити швидке впровадження та забезпечити потрібну функціональність та надійність для ефективного управління торговельним підприємством.

Формулювання вимог до розроблюваної інформаційної системи для торгівельного підприємства є критичним етапом в процесі розробки. Детальне визначення функціональних та нефункціональних вимог допомагає забезпечити, що система буде ефективно виконувати свої завдання і задовольняти потреби підприємства.

Серед функціональних вимог можуть бути наступні аспекти:

1. Управління запасами: Система повинна мати здатність точно відслідковувати запаси товарів, контролювати їх рух та автоматично

поповнювати запаси в разі необхідності. Також, вона має забезпечувати можливість планування запасів на основі аналізу попиту та прогнозування.

2. Обробка замовлень: Система повинна забезпечувати автоматизовану обробку замовлень від клієнтів, включаючи прийом, обробку, розподіл та відстеження стану замовлення. Крім того, вона має забезпечувати можливість генерації документів, таких як рахунки-фактури та накладні.

3. Управління продажами: Система повинна мати функціональності для керування процесом продажу, включаючи створення та керування ціновими політиками, облік знижок та акцій, звітність по продажам та аналітику результативності.

4. Логістика: Система повинна включати функції, пов'язані з управлінням поставками, відстеженням вантажів, маршрутизацією та управлінням транспортними засобами. Вона має забезпечувати оптимальне планування доставок та контроль їх виконання.

5. Фінансова звітність: Система повинна мати можливість автоматичного формування фінансової звітності, включаючи баланс, звіт про прибуток і збиток, звіти про грошові потоки. Вона має забезпечувати точність і надійність фінансової інформації для прийняття рішень.

Крім функціональних вимог, також важливо враховувати нефункціональні вимоги, які стосуються характеристик системи, її надійності, швидкодії, безпеки, масштабованості та зручності користування. Наприклад, система повинна мати інтуїтивно зрозумілий інтерфейс користувача, високу швидкодію обробки даних, захищеність інформації та можливість розширення функціональності.

Загальноприйнятим методом формулювання вимог є використання методології SMART. Методологія SMART (Specific, Measurable, Achievable, Relevant, Time-bound) є ефективним інструментом для формулювання вимог до інформаційної системи, оскільки дозволяє зробити їх конкретними, вимірюваними, досяжними, відповідними та обмеженими в часі.

1. Специфічні (Specific): Вимоги повинні бути конкретними та чітко сформульованими. Замість загальних тверджень, вимоги повинні бути точно визначеними, наприклад, "Система повинна мати можливість автоматичного формування фінансової звітності за період не менше ніж один місяць".

2. Вимірювані (Measurable): Вимоги повинні бути вимірюваними, щоб мати змогу перевірити їх виконання. Вимоги повинні містити критерії, за якими можна оцінити, чи були вони виконані. Наприклад, "Система повинна забезпечувати швидкодію обробки замовлень не менше ніж 5 секунд на одну транзакцію".

3. Досяжні (Achievable): Вимоги повинні бути досяжними з урахуванням обмежень і ресурсів. Вони повинні бути реалістичними з точки зору можливостей розробки системи та доступних ресурсів, таких як бюджет, час та експертиза. Наприклад, "Система повинна бути розроблена з використанням наявних технологічних рішень та бюджету підприємства".

4. Відповідні (Relevant): Вимоги повинні бути відповідними меті та потребам підприємства. Вони повинні вирішувати проблеми та відповідати стратегії торгівельного підприємства. Наприклад, "Система повинна мати можливість інтеграції з існуючою системою управління клієнтами для покращення обслуговування клієнтів та підвищення їх задоволеності".

5. Обмежені у часі (Time-bound): Вимоги повинні бути обмежені в часі, вказуючи конкретні терміни виконання. Це допомагає установити чіткі граничні строки для розробки системи. Наприклад, "Система повинна бути розроблена та впроваджена протягом 12 місяців з моменту початку проекту".

Використання методології SMART допомагає уникнути неоднозначності та розмитості вимог, забезпечує їх конкретність та можливість перевірки виконання. Це важливий крок у процесі розробки інформаційної системи торгівельного підприємства, що сприяє досягненню успішних результатів проекту.

Таким чином, вимоги до інформаційної системи для торгівельного підприємства детально формулюються, включаючи функціональні та

нефункціональні вимоги, з метою забезпечення створення системи, яка відповідає потребам підприємства, забезпечує його ефективність та конкурентоспроможність, а також задовольняє вимоги ринку та споживачів.

1.5. Ідентифікація CASE-системи

Ідентифікація CASE-системи є важливим етапом при розробці інформаційної системи торгівельного підприємства. В цьому розділі будуть визначені потреби та вимоги до CASE-системи, яка буде використовуватися для розробки проекту, а також проаналізовані можливі варіанти CASE-систем та їх підходящість для даного проекту.

Один з потенційних варіантів CASE-системи, який був вибраний для розробки інформаційної системи торгівельного підприємства, - Umbrello. Umbrello є повноцінною CASE-системою, заснованою на відкритих стандартах, що базується на UML (Unified Modeling Language). Ця система надає широкий набір функціональних можливостей та інструментів для моделювання, проектування та розробки інформаційних систем.

Однією з основних переваг Umbrello є його розширена функціональність, яка дозволяє розробникам створювати різні типи діаграм, такі як діаграми класів, діаграми послідовності, діаграми діяльності, діаграми прецедентів та багато інших. Це дозволяє детально визначити структуру системи, взаємодії між компонентами, поведінку системи та інші аспекти проекту. Крім того, Umbrello надає можливість генерації коду з моделей, що полегшує процес розробки та забезпечує високу продуктивність.

Зручність використання також є однією з важливих характеристик Umbrello. Його інтуїтивно зрозумілий інтерфейс користувача дозволяє розробникам швидко освоїти систему та працювати з нею ефективно. Крім того, Umbrello надає можливість зручного редагування та перегляду моделей, швидкого навігації по діаграмам та компонентам, що значно полегшує роботу розробників.

Однією з важливих переваг Umbrello є його сумісність з іншими CASE-системами та інструментами розробки. Він підтримує імпорт та експорт

діаграм у різних форматах, таких як XMI (XML Metadata Interchange), що дозволяє обмінюватися моделями з іншими розробниками та використовувати інші інструменти для подальшої обробки та редагування моделей.

Таким чином, використання CASE-системи Umbrello для розробки інформаційної системи торгівельного підприємства забезпечує систематичний та структурований підхід до проекту. Umbrello надає потужні інструменти моделювання, зручний інтерфейс, можливість генерації коду та сумісність з іншими CASE-системами, що сприяє ефективній розробці інформаційної системи.

Висновки до Розділу 1.

В данному розділі дипломної роботи був проведений аналіз предметної області, що стосується розробки інформаційної системи для торгівельного підприємства за допомогою CASE-системи. В ході аналізу були розглянуті основні аспекти, пов'язані з торгівельним сектором та функціонуванням торгівельних підприємств.

Перш за все, були досліджені сучасні тенденції та характеристики торгівельного сектору. Виявлено, що торгівельні підприємства стикаються з різноманітними проблемами в області управління запасами, продажів, логістики та звітності. Ці проблеми можуть значно впливати на ефективність та конкурентоспроможність підприємства.

Далі було проведено аналіз існуючих інформаційних систем, які використовуються торгівельними підприємствами. Були розглянуті їх функціональні можливості, переваги та недоліки. Встановлено, що деякі існуючі системи можуть задовольняти потреби торговельних підприємств, проте не в повній мірі і не забезпечують відповідну ефективність та гнучкість.

Для розробки інформаційної системи було визначено вимоги, які система повинна задовольняти. Вимоги були сформульовані як функціональні, такі як зберігання та обробка даних, інтерфейс користувача,

безпека, так і нефункціональні, такі як швидкодія та масштабованість. Ці вимоги стануть основою для подальшої розробки інформаційної системи торгівельного підприємства.

Також була проведена ідентифікація CASE-системи для розробки інформаційної системи. Після аналізу можливих варіантів було вибрано CASE-систему Umbrello. Ця система володіє потужними інструментами моделювання, зручним інтерфейсом та сумісністю з іншими інструментами розробки. Використання Umbrello забезпечить структурований та систематичний підхід до проекту.

Отже, аналіз предметної області дозволив отримати глибоке розуміння проблем, які торгівельні підприємства зустрічають у своїй діяльності, а також встановити вимоги до інформаційної системи та вибрати відповідну CASE-систему для розробки. Це стане основою подальшого розділу, де буде розглянуто сам процес розробки інформаційної системи торгівельного підприємства з використанням обраної CASE-системи Umbrello.

РОЗДІЛ 2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Характеристика завдань інформаційної системи

У цьому розділі проведено аналіз бізнес-процесів торгівельного підприємства та визначено завдання, які підлягають автоматизації в рамках розробки інформаційної системи. Основними властивостями цієї системи є інтеграція та обмін даними, автоматизація бізнес-процесів, забезпечення доступу та безпеки, аналітика та звітність, а також масштабованість.

Завдання інформаційної системи включають:

1. Управління складом товарів: Система повинна дозволяти вести облік товарів, їхню категоризацію та класифікацію. Вона має забезпечувати контроль за рухом товарів на складі, відстежувати залишки та автоматично поповнювати запаси при необхідності.
2. Замовлення та продаж: Система повинна надавати можливість приймати замовлення від клієнтів, формувати рахунки-фактури та керувати процесом доставки товарів. Вона має автоматизувати обробку замовлень, розрахунок цін та знижок, а також управління оплатою та веденням обліку продажів.
3. Клієнтський сервіс: Система повинна забезпечувати зберігання інформації про клієнтів, їхні замовлення та історію взаємодії з компанією. Вона має дозволяти забезпечити персоналу доступ до цієї інформації для надання якісного та персоналізованого обслуговування клієнтів.
4. Аналітика та звітність: Система повинна мати вбудовані інструменти для аналізу даних, створення звітів та панелей управління. Вона має забезпечувати зручний доступ до ключових показників ефективності підприємства, включаючи звіти про продажі, залишки товарів, фінансову звітність тощо.
5. Безпека та доступ: Система повинна гарантувати захист даних, контроль доступу та автентифікацію користувачів. Вона має

забезпечувати резервне копіювання та відновлення даних, а також забезпечувати конфіденційність і цілісність інформації.

Ці завдання є критичними для ефективної роботи торгівельного підприємства і їх автоматизація за допомогою CASE-системи дозволить покращити продуктивність, оптимізувати бізнес-процеси та забезпечити точну та швидку обробку інформації.

2.2. Вибір програми для моделювання інформаційної системи

В процесі розробки інформаційної системи для торгівельного підприємства, важливим аспектом є вибір програмного забезпечення, яке дозволить моделювати та проектувати систему з максимальною ефективністю та точністю. Одним з потенційних варіантів, що було розглянуто, є програма Umbrello.

Umbrello є одним з популярних програмних засобів для моделювання та проектування інформаційних систем. Ця програма надає потужні інструменти для створення діаграм класів, діаграм прецедентів, діаграм послідовностей та багатьох інших, які допомагають розробникам уявити та уточнити структуру та функціональність системи перед переходом до її фактичної реалізації.

Однією з основних переваг Umbrello є його безкоштовна та відкрита ліцензія. Це означає, що програма доступна для використання без будь-яких витрат, що є важливим фактором для студента або дослідника, який працює над проектом з обмеженим бюджетом.

Програма має інтуїтивно зрозумілий інтерфейс, що дозволяє легко розібратися з основними функціями програми та ефективно використовувати їх. Зручність використання інтерфейсу сприяє зосередженості розробника на моделюванні та проектуванні системи, а не на вивченні складних та заплутаних інструментів.

Також Umbrello є багатоплатформовою програмою, що підтримується на різних операційних системах, таких як Windows, Linux і macOS. Це дає

можливість використовувати програму на будь-якій обраній платформі розробки.

Враховуючи всі переваги, а також відповідність вимогам дипломного проекту, вибір цієї програми став обґрунтованим. Umbrello надає зручний та потужний набір інструментів для моделювання інформаційної системи торгівельного підприємства, що дозволить ефективно реалізувати поставлені завдання проекту.

Таким чином, Umbrello було обрано в якості програмного забезпечення для моделювання інформаційної системи у цьому дипломному проекті на основі його безкоштовної ліцензії, інтуїтивно зрозумілого інтерфейсу та багатоплатформової підтримки.

Umbrello відокремлюється серед інших програм для моделювання інформаційних систем завдяки своїм особливим функціям. Ось деякі з них:

1. Інтуїтивний інтерфейс: Umbrello надає користувачам інтуїтивно зрозумілий інтерфейс, що спрощує процес моделювання інформаційних систем. Це дозволяє швидко орієнтуватися у програмі та приступити до роботи без великого навчання. На рис.2.1 зображено робоче середовище програми.

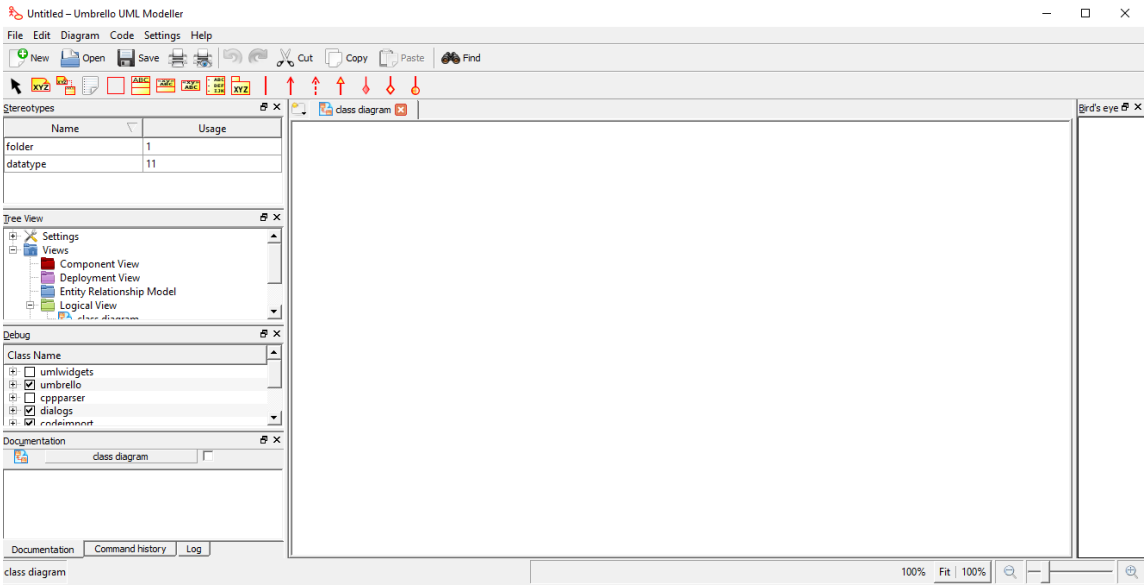

Рис.2.1. - робоче середовище Umbrello

2. Розширені можливості моделювання: Umbrello надає широкі можливості для моделювання різних аспектів інформаційних систем. Він підтримує, наприклад, створення таких діаграм:
   - діаграма класів
   - діаграма прецедентів
   - діаграма послідовностей
   - діаграма станів

   Та інших типів діаграм, що дозволяє розробникам детально відобразити різні аспекти системи.
3. Генерація коду: Umbrello дозволяє автоматично генерувати початковий код зі створених діаграм. Це значно спрощує процес розробки, оскільки програма може згенерувати скелет коду на основі структури, що була визначена в діаграмах. На рис.2.2. список мов програмування, якими Umbrello може згенерувати код.

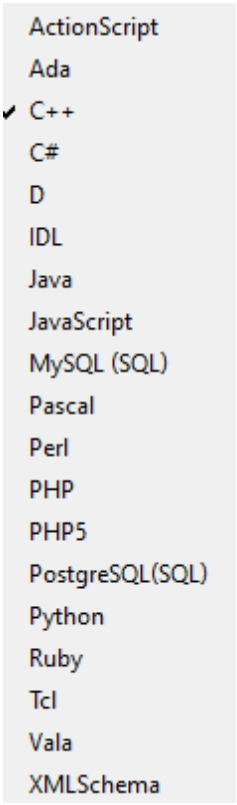

Рис.2.2. – мови програмування Umbrello

4. Багатоплатформова підтримка: Umbrello підтримується на різних операційних системах, включаючи Windows, Linux і macOS. Це дає

можливість користуватися програмою на будь-якій обраній платформі без обмежень.

5. Інтеграція з іншими інструментами: в програмі є підтримка інтеграції з іншими популярними інструментами розробки, такими як IDE (Integrated Development Environment) або системи контролю версій. Це дозволяє зручно працювати з іншими інструментами, що використовуються в процесі розробки інформаційних систем.

Ці функції роблять Umbrello привабливим вибором для моделювання інформаційних систем. Вони допомагають розробникам зосередитися на проектуванні та аналізі системи, спрощуючи процес і забезпечуючи потужні інструменти для втілення ідей та концепцій у діаграми та код.

2.3. Графічний інтерфейс

2.3.1 Огляд Umbrello

У процесі проектування інформаційної системи торгівельного підприємства необхідно використовувати потужні інструменти для моделювання та візуалізації. Один з таких інструментів, який використовується в цьому дослідженні, є Umbrello.

Umbrello є безкоштовним і відкритим інструментом для моделювання, який надає зручний графічний інтерфейс для створення діаграм UML (Unified Modeling Language). Цей інструмент забезпечує можливість проектувати та візуалізувати структуру та поведінку інформаційної системи, що дозволяє зрозуміти взаємозв'язки між компонентами системи та їх функціональність.

2.3.2 Функціональність Umbrello

Umbrello має широкий спектр функціональних можливостей, які допомагають розробникам проектувати інформаційну систему торгівельного підприємства. Основні можливості включають:

1. Створення діаграм UML: програма дозволяє створювати різні типи діаграм UML, такі як діаграми класів, діаграми послідовностей, діаграми діяльності та багато інших. Це дозволяє розробникам

моделювати структуру системи, поведінку компонентів та взаємодію між ними.

2. Візуалізація взаємозв'язків: можливість зображувати зв'язки між різними компонентами системи, наприклад:
   - Асоціації
   - Залежності
   - Агрегації
   - композиції

   Це допомагає розробникам зрозуміти структуру системи та взаємодію між її компонентами.
3. Генерація коду: Umbrello може згенерувати початковий код на основі створених діаграм. Це значно спрощує процес розробки системи, оскільки частину коду можна автоматично згенерувати з графічного представлення системи.
4. Експорт імпорт діаграм: підтримка експорту та імпорту діаграм у різних форматах, таких як XML, XMI та інші. Це дозволяє обмінюватися діаграмами з іншими інструментами для моделювання та спільно працювати над проектом.

2.4. Детальний огляд діграм Umbrello

Umbrello надає зручний інтерфейс, який дозволяє створювати різні типи діаграм UML, такі як діаграми класів, діаграми послідовностей, діаграми діяльності, діаграми станів та багато інших. Для розробки інформаційної системи було розглянуто не всі діаграми.

2.4.1. Діаграма классів (Class Diagram)

Діаграма класів є одним з найпоширеніших типів діаграм UML і головним інструментом для моделювання структури інформаційної системи. Вона дозволяє візуалізувати класи, атрибути, методи та зв'язки між ними. Давайте детальніше розглянемо основні характеристики та компоненти діаграми класів.

1. Класи: Класи представляють основні будівельні блоки діаграми класів. Кожен клас відображає конкретну сутність або об'єкт у системі. Вони містять атрибути (змінні) та методи (функції), які описують характеристики та поведінку цього класу. Наприклад, у торговій системі можуть бути класи, такі як "Товар", "Замовлення", "Клієнт" тощо.
2. Атрибути: Атрибути відображають властивості або характеристики класу. Вони можуть бути представлені у вигляді змінних, які зберігають інформацію про клас. Наприклад, у класі "Товар" атрибутами можуть бути "назва", "ціна", "кількість" тощо. Атрибути також можуть мати модифікатори доступу, які визначають, як вони доступні для інших класів у системі.
3. Методи: Методи представляють функції або операції, які можуть бути виконані класом. Вони описують поведінку або функціональність класу. Наприклад, у класі "Замовлення" можуть бути методи, такі як "додатиТовар()", "видалитиТовар()", "розрахуватиСуму()" тощо. Крім того, методи також можуть мати модифікатори доступу, які визначають, як вони доступні для виклику з інших класів.
4. Зв'язки: Зв'язки відображають взаємодію між класами. Вони показують, які класи взаємодіють між собою та які дані або повідомлення пересилаються між ними. Існує кілька типів зв'язків, таких як асоціація, композиція, агрегація, спадкування та залежність. Кожен тип зв'язку має свої особливості та семантику.
5. Ключові слова і стереотипи: Діаграма класів також може використовувати ключові слова та стереотипи для додаткової інформації про класи, атрибути, методи та зв'язки. Наприклад, ви можете використовувати ключові слова, такі як "абстрактний" або "інтерфейс", для вказівки особливих властивостей класу. Стереотипи допомагають виокремити певні типи класів або вказати їхню роль у системі.

Діаграма класів дозволяє візуалізувати структуру системи, ідентифікувати класи та їх зв'язки, а також описати атрибути та методи кожного класу. Вона є потужним інструментом для проектування інформаційної системи торгового підприємства, допомагаючи вам розуміти її структуру та взаємозв'язки між компонентами.

2.4.2. Діаграма використання (Use Case Diagram)

Діаграма використання є одним з ключових типів діаграм UML і використовується для моделювання функціональності системи з точки зору її користувачів. Вона дозволяє ідентифікувати основні актори (користувачів системи) та їх взаємодію з системою через використання варіантів використання (use cases). Основні характеристики та компоненти діаграми використання:

1. Актори: Актори представляють ролі або користувачів системи, які взаємодіють з нею. Вони можуть бути особами, іншими системами, зовнішніми пристроями або будь-якими сутностями, які взаємодіють з системою. Наприклад, в торговій системі акторами можуть бути "продавець", "клієнт", "менеджер" тощо. Актори визначаються на основі їх ролі у системі і їхнього впливу на її функціональність.
2. Варіанти використання (Use Cases): Варіанти використання представляють конкретні функціональність або дії, які можуть бути виконані користувачами системи або акторами. Кожен варіант використання описує специфічний сценарій або послідовність подій, які відбуваються між актором і системою. Наприклад, варіантом використання може бути "створення замовлення", "пошук товару", "обробка платежу" тощо.
3. Взаємодія: Діаграма використання візуалізує взаємодію між акторами і варіантами використання. Вона показує, як актори використовують варіанти використання для досягнення своїх цілей або виконання певних дій. Зв'язки між акторами та варіантами використання

показують, які варіанти використання доступні для кожного актора та які актори взаємодіють з кожним варіантом використання.

4. Розширення та угруповання: Діаграма використання також може використовувати розширення та угруповання для подробнішого опису функціональності системи. Розширення вказує на випадки, коли певний варіант використання може бути розширений іншими варіантами використання. Угруповання дозволяє групувати схожі варіанти використання під загальною категорією.

Діаграма використання допомагає зрозуміти функціональність системи з перспективи користувачів та взаємодію між ними. Вона є потужним інструментом для визначення вимог до системи та уточнення її функціональності. Використання діаграми використання допоможе створити систему, яка задовольняє потреби та очікування користувачів торгового підприємства.

2.4.3. Діаграма послідовності (Sequence Diagram)

Діаграма послідовності є одним з ключових типів діаграм UML і використовується для моделювання взаємодії між об'єктами або компонентами системи впродовж певного часу. Вона дозволяє візуалізувати послідовність повідомлень, які передаються між об'єктами та дозволяє зрозуміти, як взаємодіють різні частини системи під час виконання певного сценарію. Основні характеристики та компоненти діаграми послідовності:

1. Об'єкти: Об'єкти представляють конкретні екземпляри класів або компоненти системи, які беруть участь у взаємодії. Кожен об'єкт відображається у вершині діаграми та має своє ім'я. Об'єкти можуть бути пов'язані з класами або компонентами, які вони представляють.
2. Повідомлення: Повідомлення представляють передачу даних або виклик методів між об'єктами. Вони відображаються у вигляді стрілок, які показують напрямок передачі повідомлення. Повідомлення можуть мати назву, аргументи та значення, які передаються. Вони показують

послідовність викликів методів між об'єктами під час виконання сценарію.

3. Зв'язки та активація: Зв'язки між об'єктами відображаються у вигляді ліній, які показують взаємозв'язок між об'єктами. Це можуть бути зв'язки асоціації, залежності або зв'язки спадкування між класами. Активація показує період активності об'єкта під час виконання сценарію та показує, коли об'єкт виконує певні дії або методи.
4. Умови та цикли: Діаграма послідовності також може включати умови та цикли, які показують розгалуження або повторення виконання певних частин сценарію. Умови вказують на перехід до іншого варіанту сценарію, в залежності від певної умови, а цикли показують повторення певних дій протягом певного періоду часу.

Діаграма послідовності дозволяє візуалізувати взаємодію між об'єктами або компонентами системи під час виконання певного сценарію. Вона є інструментом для аналізу та моделювання послідовності дій у системі торгового підприємства та допомагає вам зрозуміти, як компоненти взаємодіють між собою під час виконання певних операцій або функцій.

Висновки до розділу 2

У даному розділі було проведено проектування інформаційної системи для торговельного підприємства з використанням CASE-системи. Були визначені характеристики завдань, вибрана програма для моделювання інформаційної системи та розглянуто графічний інтерфейс Umbrello. Детально розглянуті діаграми, такі як Class Diagram, Use Case Diagram та Sequence Diagram, дозволили визначити функціональність системи, взаємодію користувачів та компонентів системи.

Проектування інформаційної системи є важливим етапом в розробці технологічного рішення для торговельного підприємства. Відповідно до характеристик завдань, інформаційна система повинна забезпечувати ефективне управління торговельними операціями, облік та керування

запасами, обробку замовлень та платежів, а також забезпечувати зручний інтерфейс для користувачів.

Вибір програми для моделювання інформаційної системи базується на вимогах проекту та можливостях програмного забезпечення. В даному випадку, Umbrello було обрано як програму для моделювання, яка надає інструменти для створення діаграм та візуалізації функціональності системи.

Діаграми, такі як Class Diagram, дозволяють моделювати структуру системи, відображають класи, їх атрибути та методи, а також зв'язки між ними. Вони допомагають зрозуміти організацію даних та взаємозв'язок між компонентами системи.

Діаграма використання (Use Case Diagram) візуалізує функціональність системи з точки зору користувачів та їх взаємодію з системою. Вона дозволяє ідентифікувати варіанти використання системи та взаємозв'язок між акторами та варіантами використання.

Діаграма послідовності (Sequence Diagram) дозволяє візуалізувати послідовність повідомлень між об'єктами або компонентами системи під час виконання певного сценарію. Вона допомагає зрозуміти взаємодію між компонентами системи та послідовність викликів методів.

У результаті проектування інформаційної системи було отримано розроблені діаграми, які дають чітке уявлення про функціональність системи та взаємодію між її компонентами. Це важлива підстава для подальшої реалізації та розробки інформаційної системи для торговельного підприємства.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Моделювання діаграми використання інформаційної системи торгового підприємства

Моделювання діаграми використання є важливим етапом в проектуванні інформаційної системи для торговельного підприємства. Діаграма використання дозволяє візуалізувати взаємодію між користувачами системи та її функціональністю. Це інструмент, який допомагає розібратися в потребах користувачів і визначити ключові сценарії використання системи.

Мета даного розділу полягає в розробці та описі діаграми використання інформаційної системи для торговельного підприємства. Це дозволить краще зрозуміти, як система буде використовуватися користувачами та які основні функції вона повинна виконувати.

Для початку було визначено акторів інформаційної системи:

- Клієнт
- Постачальник
- Працівник
- Менеджер

Ці актори виповнюють основні функції торгового підприємства, автоматизація або напівавтоматизація покращить робочий процес:

- Купівля продукції
- Продаж продукції
- Обробка замовлень клієнтів
- Управління запасами
- Обробка платежів
- Формування звітів

Тож для початку в програмі Umbrello було створено акторів. Для того, щоб добавити акторів, треба на панелі інструментів обрати інструмент Actor

(рис.3.1.), або скористатися панеллю Tree View – Use Case View – New – Actor (рис.3.2.).

Рис.3.1. – інструмент Actor

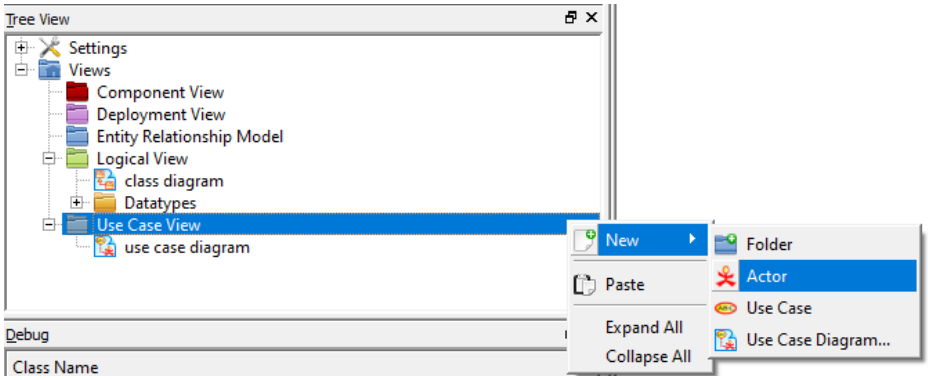

Рис.3.2. – інший спосіб створити актора

Створених акторів можна побачити на рис.3.3.

Рис.3.3. – актори інформаційної системи

Далі було добавлено прецеденти (Use Cases) до робочого середовища. Для цього треба на панелі інструментив обрати інструмент Use Case (рис.3.4.), або знову скористатись панеллю Tree View – Use Case View – New – Use Case (рис.3.5.).

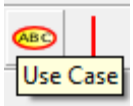

Рис.3.4. – інструмент Use Case

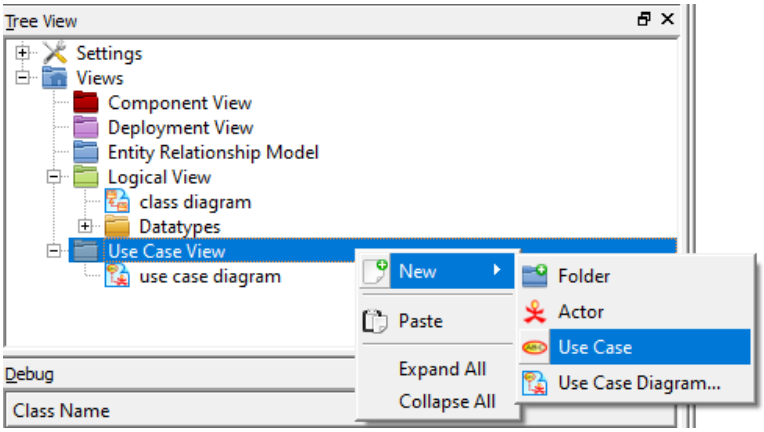

Рис.3.5. – інший спосіб створити прецедент

Робоче середовище після створення прецедентів виглядатиме так, як зображено на рис.3.6.

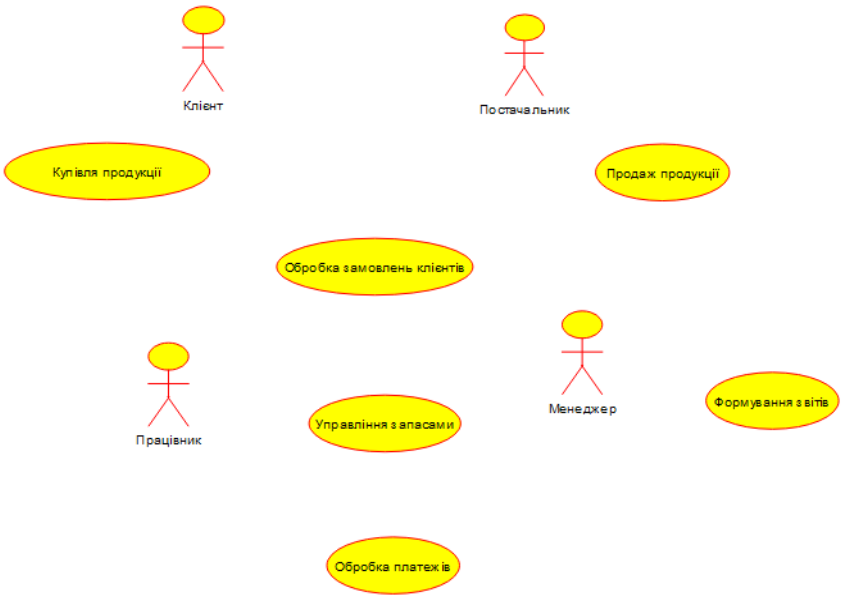

Рис.3.6. – актори та прецеденти

Основою діаграми використання є визначення взаємодій. У цій діаграмі використовуються різні типи зв'язків для показу різних видів взаємодії. Нижче наведено опис чотирьох зв'язків, які можуть бути використані в діаграмі використання:

1. Асоціація (Association): Асоціація відображає взаємозв'язок між акторами та варіантами використання. Вона показує, що актори можуть взаємодіяти з системою та користуватися її функціональністю.

2. Напрямлена асоціація (Directional Association): Напрямлена асоціація вказує на напрямок взаємодії між акторами та системою. Вона показує, що взаємодія відбувається в одному напрямку.
3. Залежність (Dependency): Залежність вказує на те, що зміни в одному елементі можуть вплинути на інший елемент. Вона показує, що один елемент залежить від іншого для своєї роботи. Наприклад, на діаграмі використання може бути залежність між актором.
4. Реалізація (Implements): Реалізація вказує на те, що деякий актор або система реалізує певний функціонал або інтерфейс. Вона показує, що актор або система забезпечує реалізацію певного функціоналу або виконує певні дії.

Використання цих зв'язків на діаграмі використання допомагає відобразити різні аспекти взаємодії між акторами та системою, що сприяє кращому розумінню функціоналу системи та її використання.

Наступним кроком треба встановити зв’зки між акторами та прецедентами. На рис.3.7. зображено те, як зв’язки виглядають на панелі інструментів.

Рис.3.7. – типи зв’язків на панелі інструментів

Після встановлення зв’язків діаграма має вигляд, зображений на рис.3.8. Зрозуміння діаграми використання допоможе нам уникнути недорозумінь та помилок під час подальшої розробки системи. Вона стане основою для подальшого проектування та розробки інформаційної системи, дозволяючи нам краще враховувати потреби користувачів та забезпечити ефективну та зручну функціональність системи.

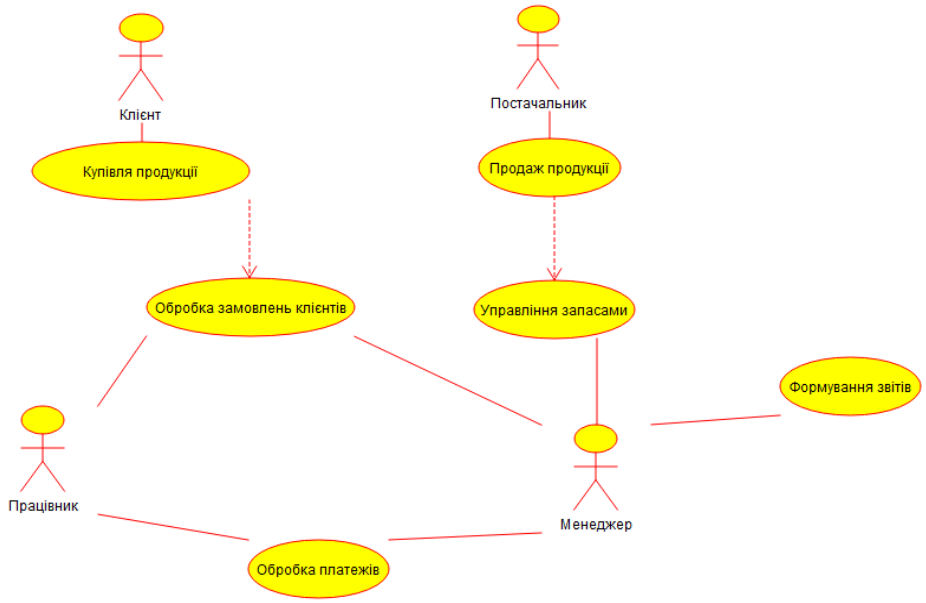

Рис.3.8. – загальний вигляд діаграми використання

3.2. Моделювання діаграми класів інформаційної системи торгового підприємства

Модель класів є центральним елементом в розробці інформаційних систем, оскільки вона дозволяє описати структуру системи та взаємозв'язки між її складовими частинами. Вона допомагає уявити, які об'єкти, процеси та функції присутні в системі та як вони взаємодіють між собою. Модель класів відображає основні сутності та їх властивості, а також забезпечує зрозумілу та однозначну специфікацію системи для розробників, аналітиків та інших зацікавлених сторін.

Після зробленої діаграми використання можна визначити такі класи:

- Клієнт
- Постачальник
- Працівник
- Менеджер
- Продукт
- Замовлення
- Звіт
- Склад

Для того, щоб добавити клас в робоче середовище, треба на панелі інструментів обрати інструмент Class (рис.3.9.), або на панелі Tree View – Logical View – New – Class (рис.3.10.)

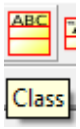

Рис.3.9. – інструмент Class

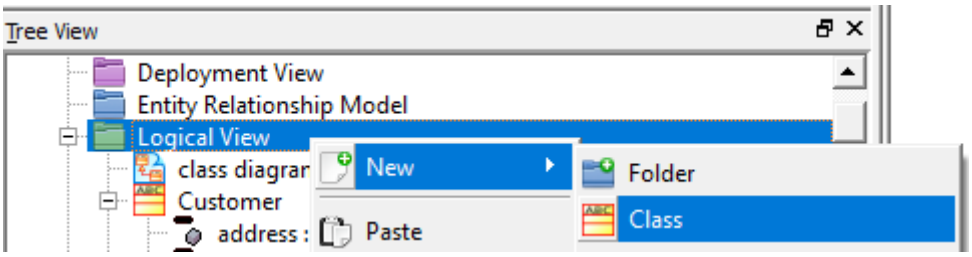

Рис.3.10. – інший спосіб створити клас

Після створення класів робоче середовище має вигляд як на рис.3.11.

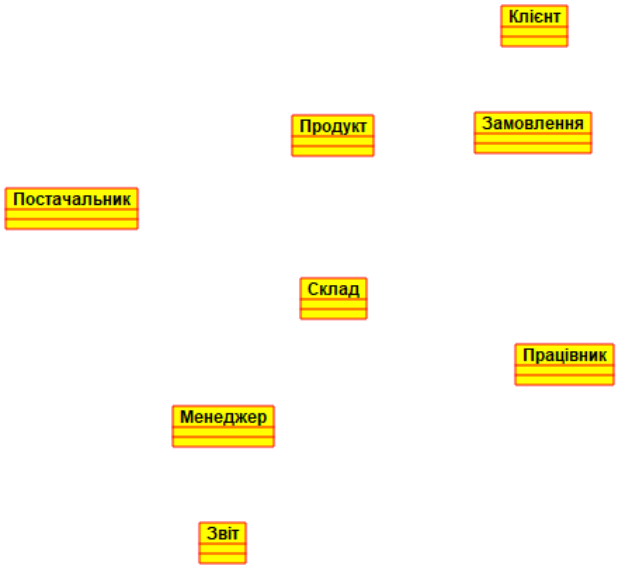

Рис.3.11. – робоче середовище діаграми класів

У кожного класа є свої атрибути, які треба встановити. Наприклад, у класа «Клієнт» є такі атрибути:

- ІдНомер
- ПІБ
- Email
- Адреса
- Контакт

Для того, щоб призначити класу атрибути, треба клікнути на клас у робочому середовищу два рази, або скористатись панеллю Tree View – назва класу – New – Attribute (рис.3.12.)

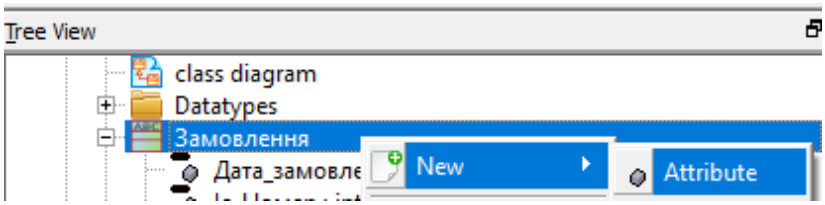

Рис.3.12. – створення атрибутів

Після цього відкриється панель властивостей (рис.3.13.), у якій треба вибрати вкладку Attrubutes. Щоб призначити атрибут класу дали треба нажати кнопку New Attribute.

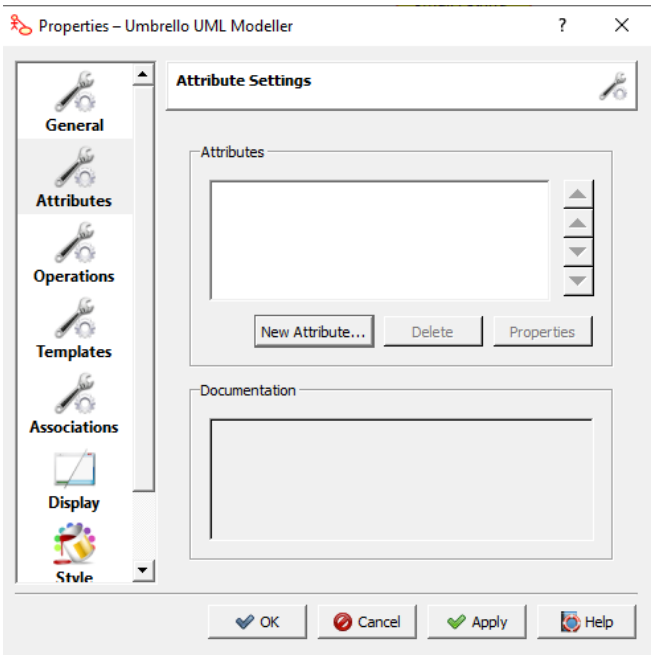

Рис.3.13. – панель властивостей класу

Далі у вікні Властивості атрибуту треба ввести назву атрибуту та тип (рис.3.14).

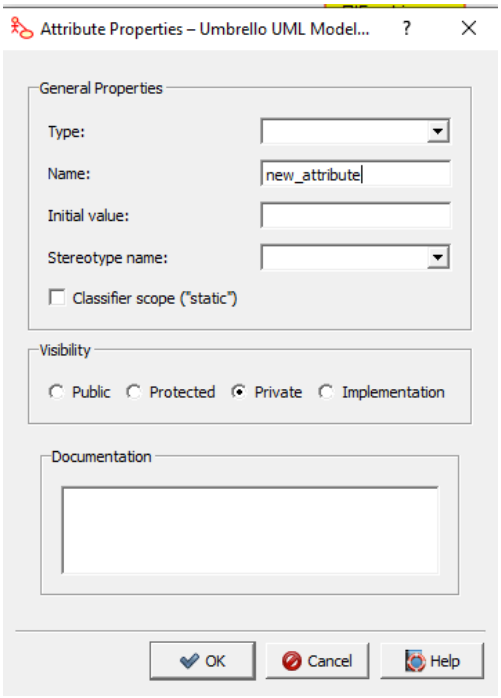

Рис.3.14. – панель властивостей атрибуту

На рис.3.15. зображено діаграму класів після призначення атрибутів для кожного класу.

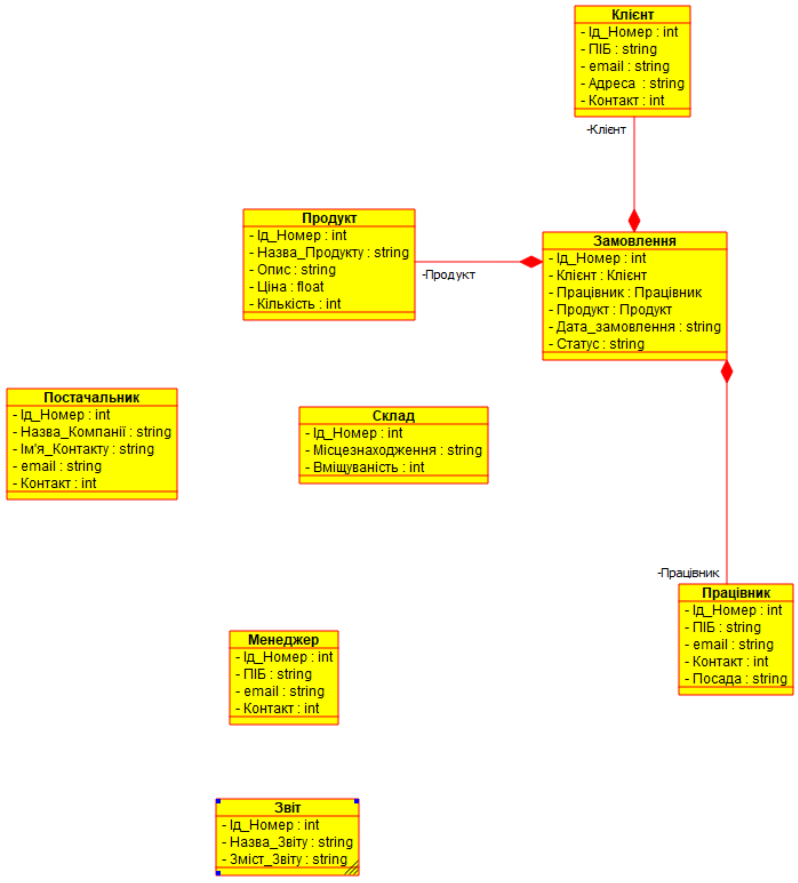

Рис.3.15. – робоче середовище діаграми класів

Наступний крок – це встановлення методів для кожного класу. Для того, щоб це зробити, треба повторити такі самі кроки, як для встановлення атрибутів, але обрати Operation (рис.3.16.).

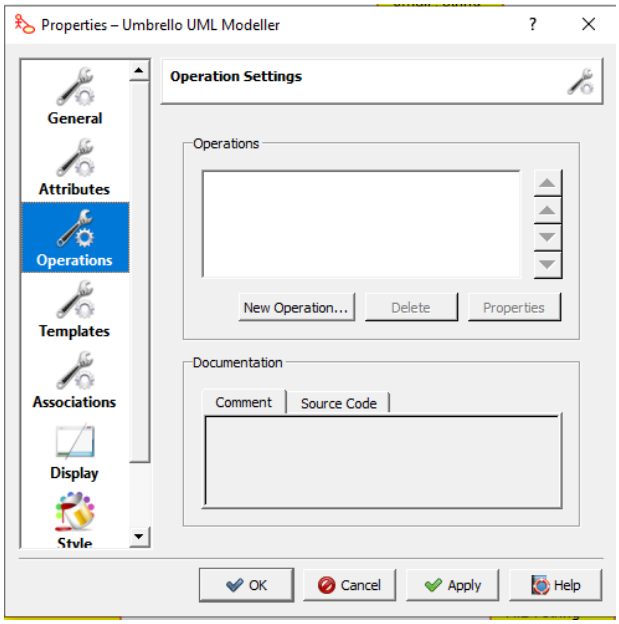

Рис.3.16. – панель методів.

Методи класів відіграють важливу роль у моделюванні об'єктно-орієнтованих систем, включаючи інформаційні системи торговельних підприємств. Методи представляють собою функції або операції, які можуть бути виконані над об'єктами цього класу. Вони визначають поведінку класу, його можливості та взаємодію з іншими об'єктами в системі.

Наприклад, для класу «Клієнт» можна призначити такі методи:

- getІДНомер
- getПІБ
- getEmail
- getАдреса
- getКонтакт

Тож на рис.3.17. зображено робоче середовище після встановлення методів для кожного класу.

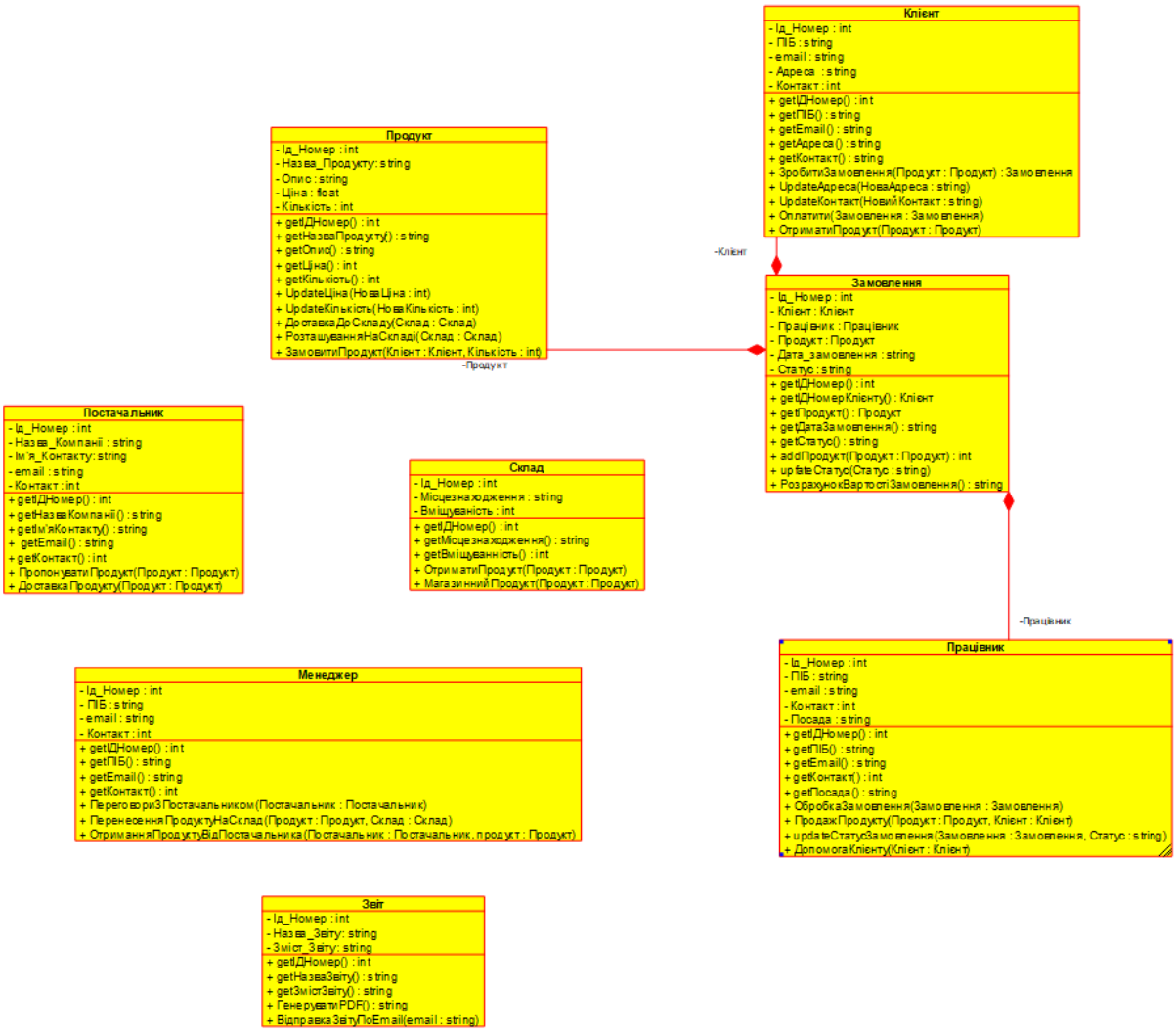

Рис.3.17. – робоче середовище діаграми класів

У діаграмі класів, лінії взаємозв'язку використовуються для показу залежностей та взаємозв'язків між класами. Вони відображають взаємодію та комунікацію між класами в системі. У діаграму класів включено лінії зв’язки з діаграм використання, але є нові:

- Composition (Композиція): Це сильний тип асоціації, де один клас є "власником" або "складовою" іншого класу. Композиція означає, що складовий клас пов'язаний з власником так тісно, що не може існувати окремо від нього.
- Aggregation (Агрегація): Агрегація також представляє зв'язок "частина-ціле" між класами, але вона не є таким сильним, як композиція. У випадку агрегації, класи можуть існувати окремо

один від одного. Власник може містити один або більше екземплярів складового класу.

Результат встановлення зв’язків можна побачити на рис.3.18.

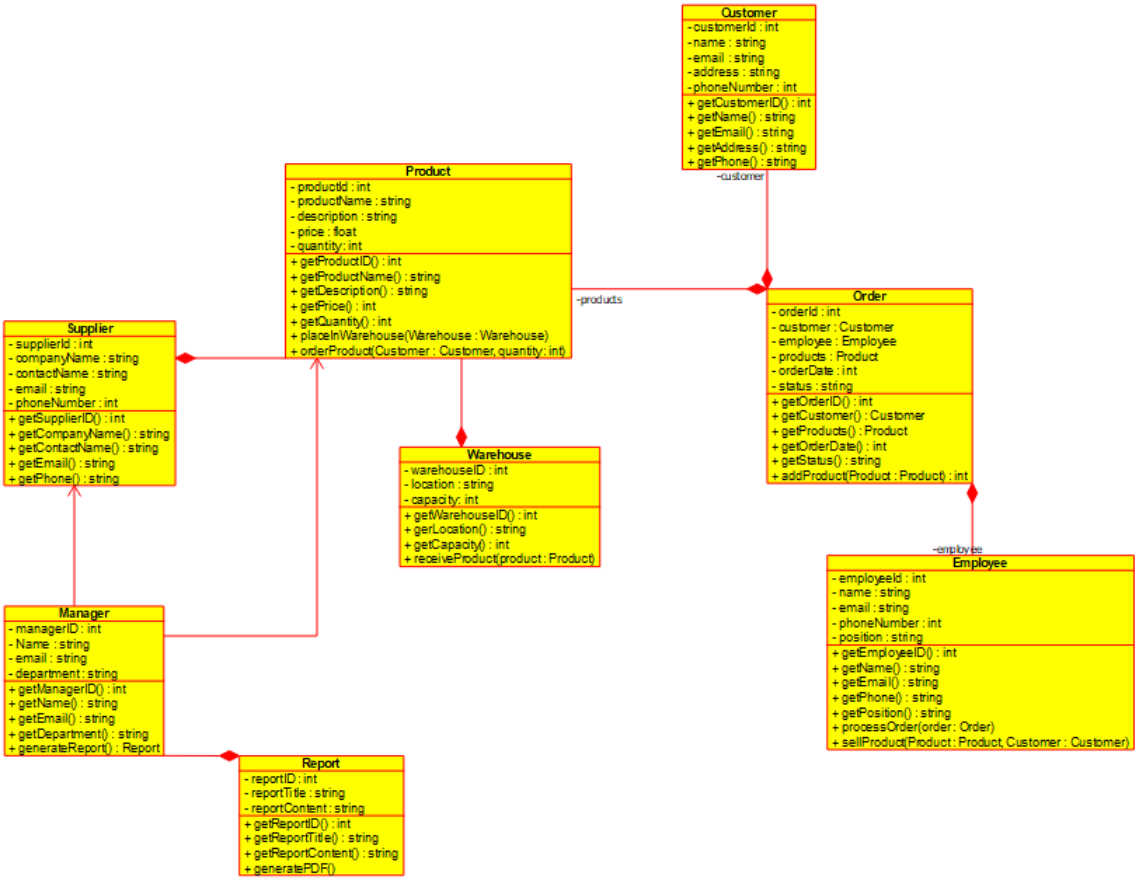

Рис.3.18. – результат встановлення зв’язків.

Модель класів інформаційної системи торгового підприємства допоможе зрозуміти, як система організована та як вона функціонує на рівні об'єктів і їх взаємозв'язків. Це надасть цінний інструмент для подальшого розвитку системи, виявлення можливих покращень та забезпечення більш ефективного управління торговельним підприємством.

3.3. Генерація коду в Python

Інформаційна система розробляється з використанням CASE-системи, що дозволяє автоматизувати процеси проектування та розробки програмного забезпечення. Одним із ключових етапів у цьому процесі є генерація програмного коду, яка дозволяє перетворити проект інформаційної системи на виконуваний код, який може бути виконаний на комп'ютері.

На рис.3.19. зображено як в Umbrello згенерувати код з діаграми класів.

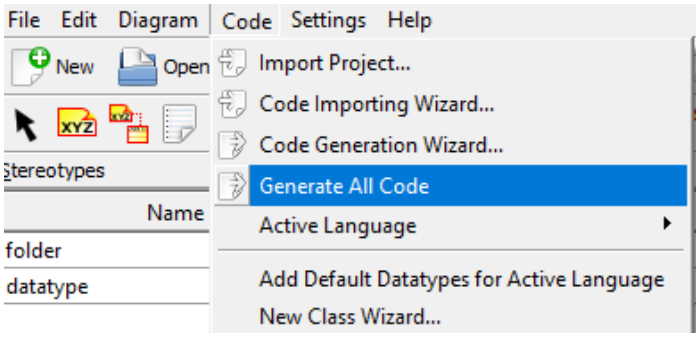

Рис.3.19. – генерація коду

Основна мета генерації коду діаграм класів полягає в перетворенні дизайну системи, який представлений у вигляді діаграм класів, на реалізацію цієї системи у вигляді програмного коду. Цей процес дозволяє зекономити час і зусилля розробника, оскільки він автоматизує процес створення скелету програмного коду на основі вже визначених класів, їх атрибутів та методів.

Результат генерації кода у Python можна побачити у додатку А.3.

Після генерації код класів не є повною програмою, яку можна запустити. Для запуску коду знадобиться додати основну функцію. Після додавання основної функції маємо програму для класу «Клієнт», який можна побачити на рис.3.24.

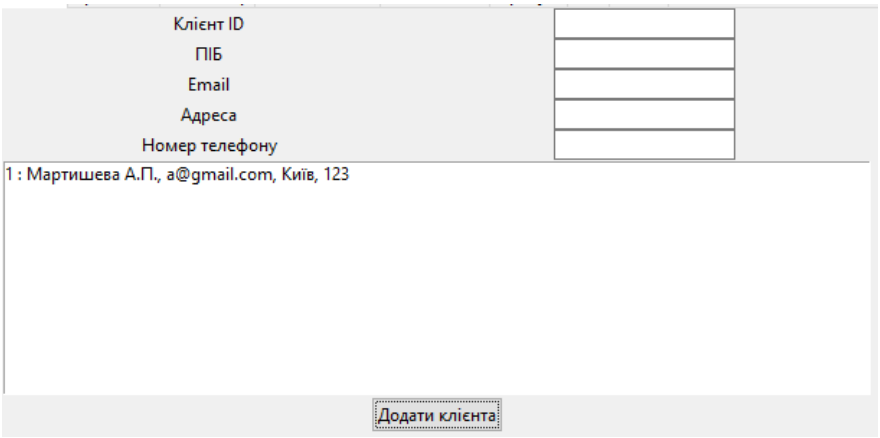

Рис.3.24. – клас «Клієнт» у Python

3.4. Огляд роботи програми

У даному розділі проводиться огляд роботи розробленої програми для інформаційної системи торгівельного підприємства. Огляд роботи програми є важливим етапом у процесі розробки, оскільки дозволяє перевірити

функціональність, надійність та ефективність програмного забезпечення. Після запуску коду відкривається вікно інформаційної системи. В цьому вікні є всі вкладки, що відповідають класам з Umbrello. Побачити інтерфейс можливо на рис.3.25:

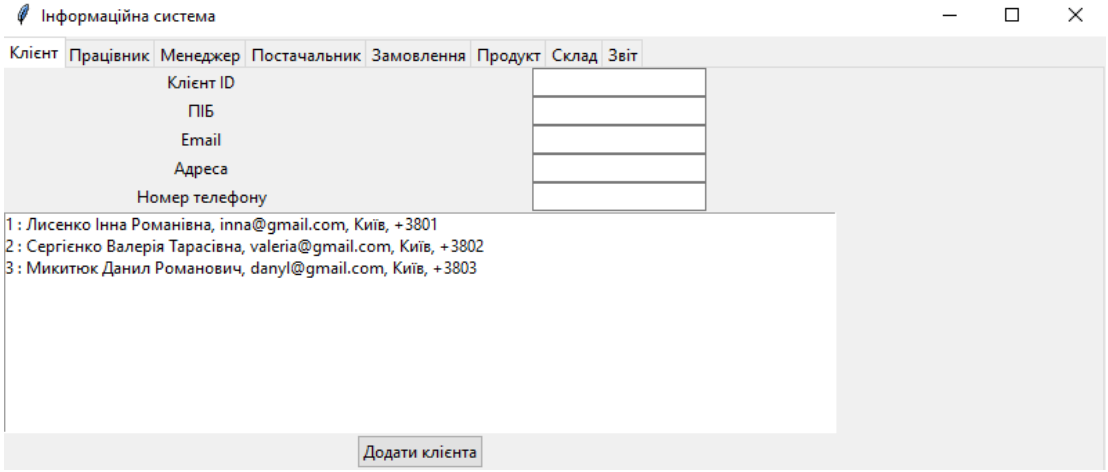

Рис.3.25. – інтерфейс програми.

Перша вкладка – «Клієнт». В ній є поля введення «ID», «ПІБ», «Email», «Адреса», «Номер телефону». Після натискання кнопки «Додати клієнта» додається клієнт до списку.

На рис.3.26. зображені вкладки «Працівник», «Менеджер», «Постачальник». Вони мають схожий зміст, але деякі поля введеня відрізняються та містять іншу інформацію.

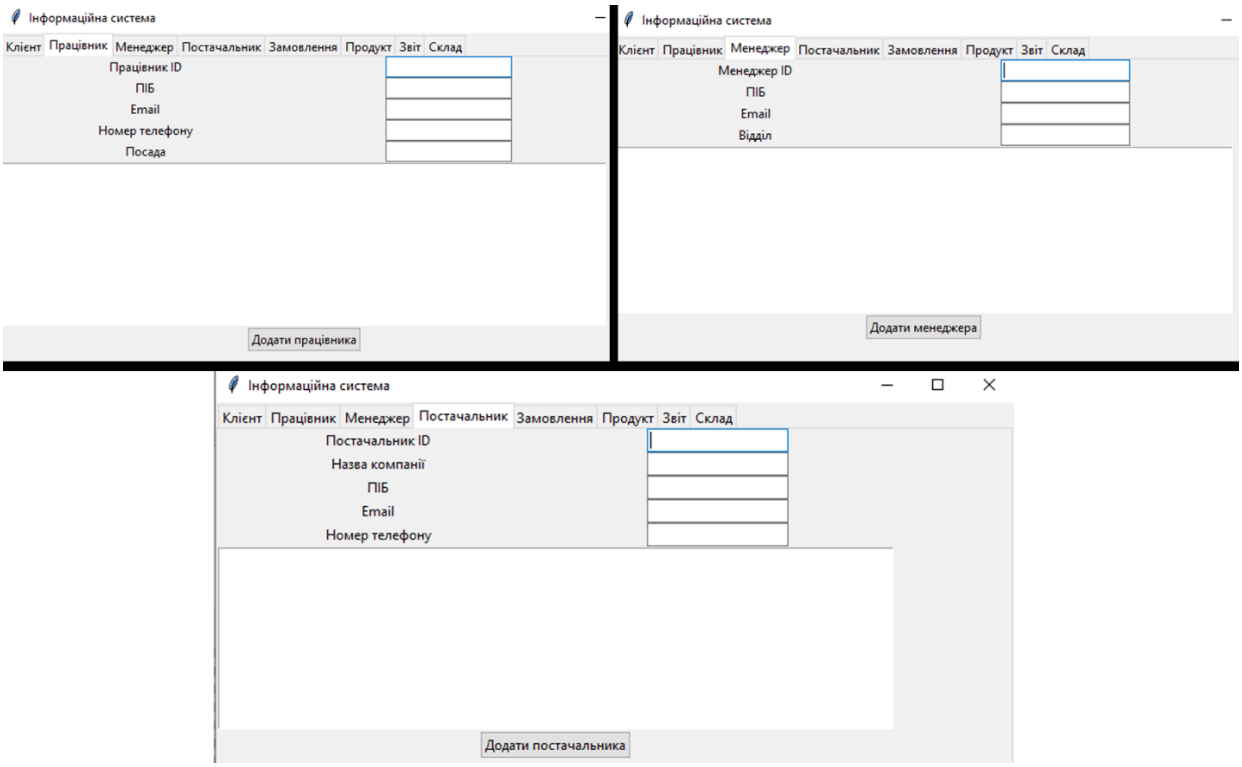

Рис.3.26. - вкладки «Працівник», «Менеджер», «Постачальник»

Ці аспекти програми надають можливість зберігати інформацію як про клієнтів, постачальників, так і про працівників підприємства.

У вкладці «Продукт» працівники підприємства можуть додавати до складу товари. На рис.3.27. зображено як влаштован ввод інформації про продукт.

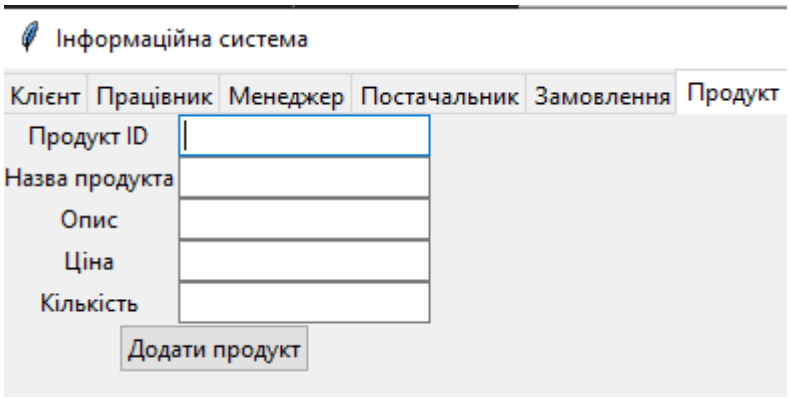

Рис.3.27. – вкладка «Продукт»

Після натискання на кнопку «Додати продукт» товар додається до списку у вікно «Склад». В цьому вікні зберігається інформація про розташування складу та інформація про продукти на складі (рис.3.28.)

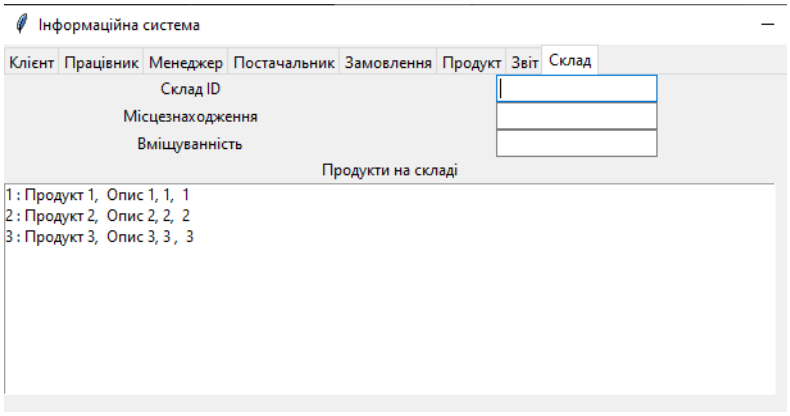

Рис.3.28. – вкладка «склад»

Також для менеджерів є функція заповнення замовлень та звітів. На рис.3.29. зображено вікно «Замовлення».

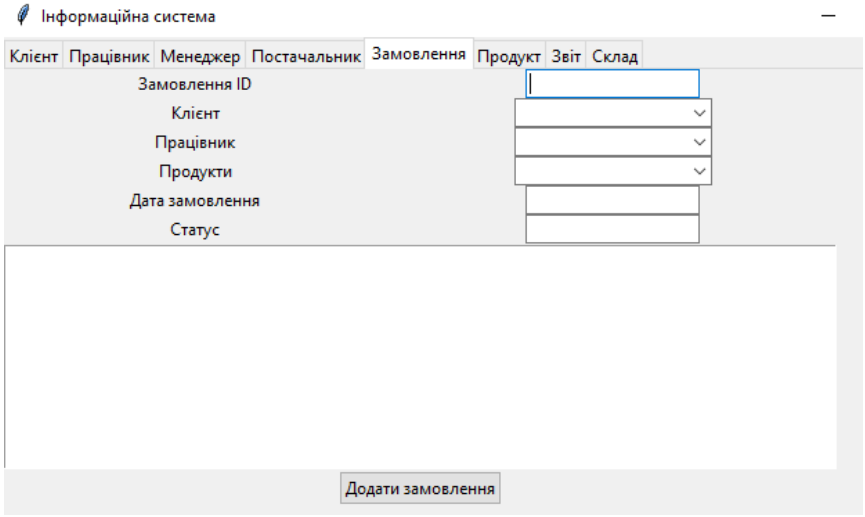

Рис.3.29. – вікно «Замовлення»

Замість поля введення клієнтів, працівників та продуктів програма надає можливість обрати їх із списків відповідних вікон.

На основі проведеного огляду роботи програми можна зробити наступні висновки. Програма, розроблена для інформаційної системи торгівельного підприємства, успішно втілює основні функції, необхідні для ефективного управління бізнес-процесами. Вона відповідає вимогам, поставленим до неї, і забезпечує надійну та ефективну роботу системи. Проте, можливість подальшого вдосконалення та розширення програми може бути розглянута з метою покращення її функціональності та відповідності змінюються потребам торгівельного підприємства.

Висновки до розділу 3

У цьому розділі було зосереджено на розробці програмного забезпечення, використовуючи попередньо створені діаграми використання та класів, а також здійснено генерацію коду в мові програмування Python та проведення огляду програми, яка використовується для цієї мети.

Розділ розпочато з моделювання діаграми використання, що дозволило візуалізувати основних актори та їх взаємодію з інформаційною системою. Це дало чітке уявлення про функціональні можливості системи та взаємодію з її користувачами.

Далі зроблено моделювання діаграми класів, що дозволило детально визначити структуру системи та взаємозв'язки між класами. Це було основою для подальшої програмної реалізації системи, де кожен клас представляв собою модуль з певними функціональними можливостями.

Після цього розпочато генерацію коду в мові програмування Python з використанням відповідної програми. За допомогою цього інструменту було автоматично згенеровано програмний код на основі діаграм та моделей, що використовуються для інформаційної системи торгового підприємства. Це дозволило значно прискорити процес розробки та забезпечити високу точність та надійність реалізації системи.

Висновки

У першому розділі дипломного проекту було проведено аналіз предметної області, пов'язаної з торгівельними підприємствами. Було ретельно вивчено особливості функціонування та організації таких підприємств, а також існуючі проблеми, з якими вони зіштовхуються. Також було проведено аналіз існуючих інформаційних систем, використованих в торговельних підприємствах, з метою виявлення їх переваг та недоліків.

В дослідженні була розглянута тема розробки інформаційної системи для торгівельного підприємства з використанням CASE-системи. Для досягнення цієї мети було проведено аналіз предметної області, включаючи огляд торгівельних підприємств, аналіз поточних проблем і існуючих інформаційних систем. На основі цього аналізу були визначені вимоги до інформаційної системи.

Ці вимоги включають такі аспекти, як ефективне управління запасами, замовлення та постачання товарів, облік фінансових операцій, аналіз продажів та звітність. Також була проведена ідентифікація CASE-системи як потенційного інструменту для моделювання та розробки інформаційної системи.

У другому розділі було розглянуто проектування інформаційної системи для торгового підприємства. Були визначені характеристики завдань, які система повинна виконувати, зокрема автоматизація процесів замовлення, постачання, обліку та аналізу.

Також було проведено вибір програми для моделювання інформаційної системи. При виборі були враховані такі критерії, як функціональні можливості, зручність використання, наявність необхідних інструментів для моделювання діаграм та генерації програмного коду.

Після була змодельована інформаційна система за допомогою CASE-системи Umbrello. В неї входило моделювання діаграм, включаючи діаграми

використання та діаграми класів, які допомагають уявити структуру та взаємозв'язки між компонентами системи.

У результаті роботи над другим розділом було встановлено основні характеристики та вимоги до інформаційної системи для торгового підприємства. Було здійснено вибір програми для моделювання системи.

У результаті аналізу ідентифіковано CASE-систему як інструмент для моделювання і розробки інформаційної системи. Було виконано проектування інформаційної системи, включаючи характеристику завдань та вибір програми для моделювання.

У третьому розділі було здійснено моделювання діаграм використання та класів інформаційної системи. За допомогою відповідної програми було згенеровано код на мові програмування Python і розроблено програму.

Висновки з дослідження дають змогу стверджувати, що розробка інформаційної системи за допомогою CASE-системи є важливим і перспективним напрямом. Вона дозволяє забезпечити ефективну автоматизацію торгівельного підприємства, підвищити продуктивність роботи, поліпшити управління та прийняття рішень. Результати дослідження можуть бути корисними для організацій, що прагнуть впровадити сучасну інформаційну систему у свою діяльність.

Список використаних джерел

1. Поняття XML/EDI. Wayback Machine. URL: https://web.archive.org/web/20010219110800/http://citforum.univ.kiev.ua/internet/articles/xmledi.shtml
2. Учасники проектів Вікімедіа. Інформаційна система – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Інформаційна_система
3. Учасники проектів Вікімедіа. Umbrello – вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Umbrello
4. Career cornerstone center: careers in science, technology, engineering, math and medicine. Career Cornerstone Center: Careers in Science, Technology, Engineering, Math and Medicine. URL: https://www.careercornerstone.org/infosys/infosys.htm
5. Учасники проектів Вікімедіа. Unified modeling language – вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Unified_Modeling_Language
6. Avison D. E. Information systems development: methodologies, techniques, and tools. 3-тє вид. London : McGraw-Hill, 2003. 592 с.
7. Booch G. Unified modeling language user guid. Pearson Technology Group Canada, 2017. 504 с.
8. Bruegge B., Dutoit A. H. Object-Oriented software engineering using UML, patterns, and java. Pearson Education, Limited, 2009. 800 с.
9. CASE tools : index. Wayback Machine. URL: https://web.archive.org/web/20100305133706/http://case-tools.org/
10. Database systems design implementation and management. Cengage Learning, Inc, 2014.
11. FFIEC IT examination handbook infobase - computer-aided software engineering. FFIEC IT Examination Handbook InfoBase - Home.

URL: https://ithandbook.ffiec.gov/it-booklets/development-and-acquisition/development-procedures/software-development-techniques/computer-aided-software-engineering.aspx

12. Fowler M. UML distilled: a brief guide to the standard object modeling language. Pearson Education, Limited, 2018.
13. Han J. Data mining: Concepts and techniques. 3rd ed. Burlington, MA: Elsevier, 2011. 703 c.
14. Heizer J., Render B. Operations management: sustainability and supply chain management. Pearson Education, Limited, 2013.
15. Kelly R. R. Introduction to information systems: supporting and transforming business. Hoboken, NJ : John Wiley & Sons, Inc., 2015.
16. Lankhorst M. Enterprise architecture at work: modelling, communication and analysis. Springer, 2018. 360 c.
17. Laudon J. P., Laudon K. C. Management information systems: managing the digital firm. Pearson Higher Education & Professional Group, 2017. 672 p.
18. Larman C. Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development. Dorling Kindersley Pvt Ltd, 2008. 736 c.
19. Lazebna N. English-language basis of python programming language. Research bulletin series philological sciences. 2021. T. 1, № 193. C. 371–376. URL: https://doi.org/10.36550/2522-4077-2021-1-193-371-376
20. Novokshonov A. K. Performance analysis of arithmetic algorithms implemented in C++ and Python programming languages. Problems in programming. 2016. № 2-3. C. 026–031. URL: https://doi.org/10.15407/pp2016.02-03.026
21. Mohapatra S. Business process reengineering: automation decision points in process reengineering. Springer, 2012. 274 c.
22. Pearlson K. Managing and using information systems: a strategic approach. 3-тє вид. Hoboken, NJ : Wiley, 2006. 369 c.

23. Slyke C. V., Belanger F. Information systems for business: an experiential approach. Wiley & Sons, Incorporated, John, 2013.
24. Tan P.-N. Introduction to data mining. ADDISON WESLEY PUBLI, 2006. 769 c.
25. The umbrello UML modeller open source project on open hub. Open Hub, the open source network. URL: https://www.openhub.net/p/umbrello
26. The unified modeling language / ред.: M. Schader, A. Korthaus. Heidelberg : Physica-Verlag HD, 1998. URL: https://doi.org/10.1007/978-3-642-48673-9
27. Turner P., Cadle J., Paul D. Business analysis techniques: 99 essential tools for success. BCS Learning & Development Limited, 2014. 356 c.
28. UML 2002 (2002 Dresden, Germany). UML 2002- the unified modeling language: model engineering, concepts, and tools : 5th international conference, dresden, germany, september 30-october 4, 2002 : proceedings. Berlin : Springer, 2002. 447 c.
29. Vliet H. V. Software engineering: principles and practice. Wiley & Sons, Incorporated, John, 2014.

Додатки

Додаток А.3.

Результат генерації коду в програмі Umbrello

```
# coding=System
from Order import *

class Customer(object):

  """


  :version:
  :author:
  """

  """ ATTRIBUTES



  customerId  (private)



  name  (private)
```

email (private)

address (private)

phoneNumber (private)

"""

def getCustomerID(self):

  """

  @return int :

  @author

  """

  pass

def getName(self):

  """

```
  @return string :
  @author
  """
  pass

def getEmail(self):
  """



  @return string :
  @author
  """
  pass

def getAddress(self):
  """



  @return string :
  @author
  """
  pass
```

```
def getPhone(self):
  """



  @return string :
  @author
  """
  pass
```

Додаток Б

Лістинг програми

```
import tkinter as tk
from tkinter import ttk, messagebox
import csv


class CustomerTab(ttk.Frame):
    def __init__(self, parent, order_tab):
        super().__init__(parent)
        self.parent = parent
        self.order_tab = order_tab
        self.create_widgets()

    def create_widgets(self):
        self.customer_id_label = ttk.Label(self, text="Клієнт ID")
        self.customer_id_entry = ttk.Entry(self)
        self.customer_name_label = ttk.Label(self, text="ПІБ")
        self.customer_name_entry = ttk.Entry(self)
        self.customer_email_label = ttk.Label(self, text="Email")
        self.customer_email_entry = ttk.Entry(self)
        self.customer_address_label = ttk.Label(self, text="Адреса")
        self.customer_address_entry = ttk.Entry(self)
        self.customer_phone_label = ttk.Label(self, text="Номер телефону")
        self.customer_phone_entry = ttk.Entry(self)

        self.customer_id_label.grid(row=0, column=0)
        self.customer_id_entry.grid(row=0, column=1)
        self.customer_name_label.grid(row=1, column=0)
        self.customer_name_entry.grid(row=1, column=1)
        self.customer_email_label.grid(row=2, column=0)
        self.customer_email_entry.grid(row=2, column=1)
        self.customer_address_label.grid(row=3, column=0)
        self.customer_address_entry.grid(row=3, column=1)
        self.customer_phone_label.grid(row=4, column=0)
        self.customer_phone_entry.grid(row=4, column=1)

        self.customer_list = tk.Listbox(self, height=10, width=100)
        self.customer_list.grid(row=6, column=0, columnspan=2, sticky="nsew")

        self.add_customer_btn = ttk.Button(self, text="Додати клієнта",
command=self.add_customer)
        self.add_customer_btn.grid(row=7, column=0, columnspan=2)

        # Завантаження наявних даних клієнтів
        self.load_customer_data()

    def add_customer(self):
        customer_id = self.customer_id_entry.get()
        customer_name = self.customer_name_entry.get()
        customer_email = self.customer_email_entry.get()
        customer_address = self.customer_address_entry.get()
        customer_phone = self.customer_phone_entry.get()

        if customer_id and customer_name and customer_email and
customer_address and customer_phone:
            customer_info = f'{customer_id} : {customer_name},
{customer_email}, {customer_address}, {customer_phone}'
            self.customer_list.insert(tk.END, customer_info)
```

```
            self.customer_id_entry.delete(0, tk.END)
            self.customer_name_entry.delete(0, tk.END)
            self.customer_email_entry.delete(0, tk.END)
            self.customer_address_entry.delete(0, tk.END)
            self.customer_phone_entry.delete(0, tk.END)
            self.order_tab.update_customer_combobox(customer_info)
            self.save_customer_data()  # Збереження оновлених даних клієнтів
        else:
            messagebox.showerror("Неповні дані", "Будь ласка, заповніть усі
поля")

    def load_customer_data(self):
        try:
            with open('customer_data.csv', newline='') as file:
                reader = csv.reader(file)
                for row in reader:
                    self.customer_list.insert(tk.END, row[0])
        except FileNotFoundError:
            messagebox.showinfo("Файл не знайдено", "Дані клієнтів не
знайдено")

    def save_customer_data(self):
        with open('customer_data.csv', 'w', newline='') as file:
            writer = csv.writer(file)
            for i in range(self.customer_list.size()):
                writer.writerow([self.customer_list.get(i)])


class EmployeeTab(ttk.Frame):
    def __init__(self, parent, order_tab):
        super().__init__(parent)
        self.parent = parent
        self.order_tab = order_tab
        self.create_widgets()

    def create_widgets(self):
        self.employee_id_label = ttk.Label(self, text="Працівник ID")
        self.employee_id_entry = ttk.Entry(self)
        self.employee_name_label = ttk.Label(self, text="ПІБ")
        self.employee_name_entry = ttk.Entry(self)
        self.employee_email_label = ttk.Label(self, text="Email")
        self.employee_email_entry = ttk.Entry(self)
        self.employee_phone_label = ttk.Label(self, text="Номер телефону")
        self.employee_phone_entry = ttk.Entry(self)
        self.employee_position_label = ttk.Label(self, text="Посада")
        self.employee_position_entry = ttk.Entry(self)

        self.employee_id_label.grid(row=0, column=0)
        self.employee_id_entry.grid(row=0, column=1)
        self.employee_name_label.grid(row=1, column=0)
        self.employee_name_entry.grid(row=1, column=1)
        self.employee_email_label.grid(row=2, column=0)
        self.employee_email_entry.grid(row=2, column=1)
        self.employee_phone_label.grid(row=3, column=0)
        self.employee_phone_entry.grid(row=3, column=1)
        self.employee_position_label.grid(row=4, column=0)
        self.employee_position_entry.grid(row=4, column=1)

        self.employee_list = tk.Listbox(self, height=10, width=100)
        self.employee_list.grid(row=6, column=0, columnspan=2, sticky="nsew")

        self.add_employee_btn = ttk.Button(self, text="Додати працівника",
command=self.add_employee)
```

```
        self.add_employee_btn.grid(row=7, column=0, columnspan=2)

        # Завантаження наявних даних про співробітників
        self.load_employee_data()

    def add_employee(self):
        employee_id = self.employee_id_entry.get()
        employee_name = self.employee_name_entry.get()
        employee_email = self.employee_email_entry.get()
        employee_phone = self.employee_phone_entry.get()
        employee_position = self.employee_position_entry.get()

        if employee_id and employee_name and employee_email and
employee_phone and employee_position:
            employee_info = f'{employee_id} : {employee_name},
{employee_email}, {employee_phone}, {employee_position}'
            self.employee_list.insert(tk.END, employee_info)
            self.employee_id_entry.delete(0, tk.END)
            self.employee_name_entry.delete(0, tk.END)
            self.employee_email_entry.delete(0, tk.END)
            self.employee_phone_entry.delete(0, tk.END)
            self.employee_position_entry.delete(0, tk.END)
            self.order_tab.update_employee_combobox(employee_info)
            self.save_employee_data()  # Збереження оновленних даних
співробітників
        else:
            messagebox.showerror("Неповні дані", "Будь ласка, заповніть усі
поля")

    def load_employee_data(self):
        try:
            with open('employee_data.csv', newline='') as file:
                reader = csv.reader(file)
                for row in reader:
                    self.employee_list.insert(tk.END, row[0])
        except FileNotFoundError:
            messagebox.showinfo("Файл не знайдено", "Даних про співробітників
не знайдено.")

    def save_employee_data(self):
        with open('employee_data.csv', 'w', newline='') as file:
            writer = csv.writer(file)
            for i in range(self.employee_list.size()):
                writer.writerow([self.employee_list.get(i)])


class ManagerTab(ttk.Frame):
    def __init__(self, parent, report_tab, app_instance):
        super().__init__(parent)
        self.report_tab = report_tab
        self.parent = parent
        self.app_instance = app_instance
        self.create_widgets()

    def create_widgets(self):
        self.manager_id_label = ttk.Label(self, text="Менеджер ID")
        self.manager_id_entry = ttk.Entry(self)
        self.manager_name_label = ttk.Label(self, text="ПІБ")
        self.manager_name_entry = ttk.Entry(self)
        self.manager_email_label = ttk.Label(self, text="Email")
        self.manager_email_entry = ttk.Entry(self)
        self.manager_department_label = ttk.Label(self, text="Відділ")
        self.manager_department_entry = ttk.Entry(self)
```

```
        self.manager_id_label.grid(row=0, column=0)
        self.manager_id_entry.grid(row=0, column=1)
        self.manager_name_label.grid(row=1, column=0)
        self.manager_name_entry.grid(row=1, column=1)
        self.manager_email_label.grid(row=2, column=0)
        self.manager_email_entry.grid(row=2, column=1)
        self.manager_department_label.grid(row=3, column=0)
        self.manager_department_entry.grid(row=3, column=1)

        self.manager_list = tk.Listbox(self, height=10, width=100)
        self.manager_list.grid(row=6, column=0, columnspan=2, sticky="nsew")

        self.add_manager_btn = ttk.Button(self, text="Додати менеджера",
command=self.add_manager)
        self.add_manager_btn.grid(row=7, column=0, columnspan=2)

        self.add_supplier_btn = ttk.Button(self, text="Додати постачальника",
command=self.app_instance.switch_to_supplier_tab)
        self.add_supplier_btn.grid(row=8, column=0, columnspan=2)  # Adjust
the grid position as needed

        self.add_product_btn = ttk.Button(self, text="Додати продукт",

command=self.app_instance.switch_to_product_tab)
        self.add_product_btn.grid(row=9, column=0, columnspan=2)  # Adjust
the grid position as needed

        # Завантажити існуючі дані менеджерів
        self.load_manager_data()


    def add_manager(self):
        manager_id = self.manager_id_entry.get()
        manager_name = self.manager_name_entry.get()
        manager_email = self.manager_email_entry.get()
        manager_department = self.manager_department_entry.get()

        if manager_id and manager_name and manager_email and
manager_department:
            manager_info = f'{manager_id} : {manager_name}, {manager_email},
{manager_department}'
            self.manager_list.insert(tk.END, manager_info)
            self.manager_id_entry.delete(0, tk.END)
            self.manager_name_entry.delete(0, tk.END)
            self.manager_email_entry.delete(0, tk.END)
            self.manager_department_entry.delete(0, tk.END)
            self.report_tab.update_manager_combobox(manager_info)
            self.save_manager_data()  # Збереження оновлених даних менеджерів
        else:
            messagebox.showerror("Неповні дані", "Будь ласка, заповніть усі
поля")

    def load_manager_data(self):
        try:
            with open('manager_data.csv', newline='') as file:
                reader = csv.reader(file)
                for row in reader:
                    self.manager_list.insert(tk.END, row[0])
        except FileNotFoundError:
            messagebox.showinfo("Файл не знайдено", "Відсутні дані
менеджера.")
```

```
    def save_manager_data(self):
        with open('manager_data.csv', 'w', newline='') as file:
            writer = csv.writer(file)
            for i in range(self.manager_list.size()):
                writer.writerow([self.manager_list.get(i)])


class SupplierTab(ttk.Frame):
    def __init__(self, parent, product_tab):
        super().__init__(parent)
        self.product_tab = product_tab
        self.parent = parent
        self.create_widgets()

    def create_widgets(self):
        self.supplier_id_label = ttk.Label(self, text="Постачальник ID")
        self.supplier_id_entry = ttk.Entry(self)
        self.company_name_label = ttk.Label(self, text="Назва компанії")
        self.company_name_entry = ttk.Entry(self)
        self.contact_name_label = ttk.Label(self, text="ПІБ")
        self.contact_name_entry = ttk.Entry(self)
        self.email_label = ttk.Label(self, text="Email")
        self.email_entry = ttk.Entry(self)
        self.phone_number_label = ttk.Label(self, text="Номер телефону")
        self.phone_number_entry = ttk.Entry(self)

        self.supplier_id_label.grid(row=0, column=0)
        self.supplier_id_entry.grid(row=0, column=1)
        self.company name label.grid(row=1, column=0)
        self.company_name_entry.grid(row=1, column=1)
        self.contact_name_label.grid(row=2, column=0)
        self.contact_name_entry.grid(row=2, column=1)
        self.email_label.grid(row=3, column=0)
        self.email_entry.grid(row=3, column=1)
        self.phone_number_label.grid(row=4, column=0)
        self.phone_number_entry.grid(row=4, column=1)

        self.supplier_list = tk.Listbox(self, height=10, width=100)
        self.supplier_list.grid(row=6, column=0, columnspan=2, sticky="nsew")

        self.add_supplier_btn = ttk.Button(self, text="Додати постачальника",
command=self.add_supplier)
        self.add_supplier_btn.grid(row=7, column=0, columnspan=2)

        # Завантажити існуючі дані постачальника
        self.load_supplier_data()

    def add_supplier(self):
        supplier_id = self.supplier_id_entry.get()
        company_name = self.company_name_entry.get()
        contact_name = self.contact_name_entry.get()
        email = self.email_entry.get()
        phone_number = self.phone_number_entry.get()

        if supplier_id and company_name and contact_name and email and
phone_number:
            supplier_info = f'{supplier_id} : {company_name}, {contact_name},
{email}, {phone_number}'
            self.supplier_list.insert(tk.END, supplier_info)
            self.supplier_id_entry.delete(0, tk.END)
            self.company_name_entry.delete(0, tk.END)
            self.contact_name_entry.delete(0, tk.END)
            self.email_entry.delete(0, tk.END)
```

```
            self.phone_number_entry.delete(0, tk.END)
            self.product_tab.update_supplier_combobox(supplier_info)
            self.save_supplier_data()  # Збереження оновлених даних
постачальника
        else:
            messagebox.showerror("Неповні дані", "Будь ласка, заповніть усі
поля")

    def load_supplier_data(self):
        try:
            with open('supplier_data.csv', newline='') as file:
                reader = csv.reader(file)
                for row in reader:
                    self.supplier_list.insert(tk.END, row[0])
        except FileNotFoundError:
            messagebox.showinfo("Файл не знайдено", "Відсутні дані про
постачальника.")

    def save_supplier_data(self):
        with open('supplier_data.csv', 'w', newline='') as file:
            writer = csv.writer(file)
            for i in range(self.supplier_list.size()):
                writer.writerow([self.supplier_list.get(i)])


class OrderTab(ttk.Frame):
    def __init__(self, parent):
        super().__init__(parent)
        self.parent = parent
        self.create_widgets()

    def create_widgets(self):
        self.order_id_label = ttk.Label(self, text="Замовлення ID")
        self.order_id_entry = ttk.Entry(self)
        self.order_customer_label = ttk.Label(self, text="Клієнт")
        self.order_customer_entry = ttk.Combobox(self)
        self.order_employee_label = ttk.Label(self, text="Працівник")
        self.order_employee_entry = ttk.Combobox(self)
        self.order_products_label = ttk.Label(self, text="Продукти")
        self.order_products_entry = ttk.Combobox(self)
        self.order_date_label = ttk.Label(self, text="Дата замовлення")
        self.order_date_entry = ttk.Entry(self)
        self.order_status_label = ttk.Label(self, text="Статус")
        self.order_status_entry = ttk.Entry(self)

        self.order_id_label.grid(row=0, column=0)
        self.order_id_entry.grid(row=0, column=1)
        self.order_customer_label.grid(row=1, column=0)
        self.order_customer_entry.grid(row=1, column=1)
        self.order_employee_label.grid(row=2, column=0)
        self.order_employee_entry.grid(row=2, column=1)
        self.order_products_label.grid(row=3, column=0)
        self.order_products_entry.grid(row=3, column=1)
        self.order_date_label.grid(row=4, column=0)
        self.order_date_entry.grid(row=4, column=1)
        self.order_status_label.grid(row=5, column=0)
        self.order_status_entry.grid(row=5, column=1)

        self.order_list = tk.Listbox(self, height=10, width=100)
        self.order_list.grid(row=6, column=0, columnspan=2, sticky="nsew")

        self.add_order_btn = ttk.Button(self, text="Додати замовлення",
command=self.add_order)
```

```
        self.add_order_btn.grid(row=7, column=0, columnspan=2)

        # Завантажити наявні дані замовлення
        self.load_order_data()

    def update_customer_combobox(self, customer_info):
        customers = list(self.order_customer_entry['values'])
        customers.append(customer_info)
        self.order_customer_entry['values'] = tuple(customers)

    def update_product_combobox(self, product_info):
        products = list(self.order_products_entry['values'])
        products.append(product_info)
        self.order_products_entry['values'] = tuple(products)

    def update_employee_combobox(self, employee_info):
        employees = list(self.order_employee_entry['values'])
        employees.append(employee_info)
        self.order_employee_entry['values'] = employees

    def add_order(self):
        order_id = self.order_id_entry.get()
        customer = self.order_customer_entry.get()
        employee = self.order_employee_entry.get()
        products = self.order_products_entry.get()
        order_date = self.order_date_entry.get()
        status = self.order_status_entry.get()

        if order id and customer and employee and products and order date and
status:
            order_info = f'{order_id} : {customer}, {employee}, {products},
{order_date}, {status}'
            self.order_list.insert(tk.END, order_info)
            self.order_id_entry.delete(0, tk.END)
            self.order_customer_entry.delete(0, tk.END)
            self.order_employee_entry.delete(0, tk.END)
            self.order_products_entry.delete(0, tk.END)
            self.order_date_entry.delete(0, tk.END)
            self.order_status_entry.delete(0, tk.END)
            self.save_order_data()  # Зберегти оновлені дані замовлення
        else:
            messagebox.showerror("Неповні дані", "Будь ласка, заповніть усі
поля")

    def load_order_data(self):
        try:
            with open('order_data.csv', newline='') as file:
                reader = csv.reader(file)
                for row in reader:
                    self.order_list.insert(tk.END, row[0])
        except FileNotFoundError:
            messagebox.showinfo("Файл не знайдено", "Немає даних про
замовлення.")

    def save_order_data(self):
        with open('order_data.csv', 'w', newline='') as file:
            writer = csv.writer(file)
            for i in range(self.order_list.size()):
                writer.writerow([self.order_list.get(i)])


class ProductTab(ttk.Frame):
    def __init__(self, parent, order_tab, warehouse_tab):
```

```
        super().__init__(parent)
        self.parent = parent
        self.order_tab = order_tab
        self.warehouse_tab = warehouse_tab
        self.create_widgets()

    def create_widgets(self):
        self.product_id_label = ttk.Label(self, text="Продукт ID")
        self.product_id_entry = ttk.Entry(self)
        self.product_name_label = ttk.Label(self, text="Назва продукту")
        self.product_name_entry = ttk.Entry(self)
        self.product_price_label = ttk.Label(self, text="Ціна")
        self.product_price_entry = ttk.Entry(self)
        self.product_description_label = ttk.Label(self, text="Опис")
        self.product_description_entry = ttk.Entry(self)
        self.product_quantity_label = ttk.Label(self, text="Кількість")
        self.product_quantity_entry = ttk.Entry(self)
        self.product_supplier_label = ttk.Label(self, text="Постачальник")
        self.product_supplier_entry = ttk.Combobox(self)

        self.product_id_label.grid(row=0, column=0)
        self.product_id_entry.grid(row=0, column=1)
        self.product_name_label.grid(row=1, column=0)
        self.product_name_entry.grid(row=1, column=1)
        self.product_price_label.grid(row=2, column=0)
        self.product_price_entry.grid(row=2, column=1)
        self.product_description_label.grid(row=3, column=0)
        self.product_description_entry.grid(row=3, column=1)
        self.product_quantity_label.grid(row=4, column=0)
        self.product_quantity_entry.grid(row=4, column=1)
        self.product_supplier_label.grid(row=5, column=0)
        self.product_supplier_entry.grid(row=5, column=1)

        self.product_list = tk.Listbox(self, height=10, width=100)
        self.product_list.grid(row=7, column=0, columnspan=2, sticky="nsew")

        self.add_product_btn = ttk.Button(self, text="Додати продукт",
command=self.add_product)
        self.add_product_btn.grid(row=8, column=0, columnspan=2)

        # Завантажити наявні дані про продукт
        self.load_product_data()


    def update_supplier_combobox(self, supplier_info):
        supplier = list(self.product_supplier_entry['values'])
        supplier.append(supplier_info)
        self.product_supplier_entry['values'] = tuple(supplier)
    def add_product(self):
        product_id = self.product_id_entry.get()
        product_name = self.product_name_entry.get()
        product_price = self.product_price_entry.get()
        product_description = self.product_description_entry.get()
        product_quantity = self.product_quantity_entry.get()
        supplier = self.product_supplier_entry.get()

        if product_id and product_name and product_price:
            product_info = f'{product_id} : {product_name}, {product_price},
{product_description}, {product_quantity}, {supplier}'
            self.product_list.insert(tk.END, product_info)
            self.product_id_entry.delete(0, tk.END)
            self.product_name_entry.delete(0, tk.END)
            self.product_price_entry.delete(0, tk.END)
```

```
            self.product_description_entry.delete(0, tk.END)
            self.product_quantity_entry.delete(0, tk.END)
            self.product_supplier_entry.delete(0, tk.END)
            self.order_tab.update_product_combobox(product_info)
            self.warehouse_tab.update_product_list(product_info)
            self.save_product_data()  # Збережіть оновлені дані продукту
        else:
            messagebox.showerror("Неповні дані", "Будь ласка, заповніть усі
поля")

    def load_product_data(self):
        try:
            with open('product_data.csv', newline='') as file:
                reader = csv.reader(file)
                for row in reader:
                    self.product_list.insert(tk.END, row[0])
        except FileNotFoundError:
            messagebox.showinfo("Файл не знайдено", "Немає даних про
продукт.")

    def save_product_data(self):
        with open('product_data.csv', 'w', newline='') as file:
            writer = csv.writer(file)
            for i in range(self.product_list.size()):
                writer.writerow([self.product_list.get(i)])


class WarehouseTab(ttk.Frame):
    def   init  (self, parent):
        super().__init__(parent)
        self.parent = parent
        self.create_widgets()

    def create_widgets(self):
        # Мітки для додаткової інформації
        location_label = ttk.Label(self, text="Місцезнаходження: Київ")
        location_label.grid(row=0, column=0, sticky="w")

        warehouse_id_label = ttk.Label(self, text="Склад ID: 1")
        warehouse_id_label.grid(row=0, column=1, sticky="w")

        capacity_label = ttk.Label(self, text="Вміщуванність: 1000")
        capacity_label.grid(row=0, column=2, sticky="w")

        self.product_list = tk.Listbox(self, height=10, width=100)
        self.product_list.grid(row=1, column=0, columnspan=3, sticky="nsew")

        # Завантажити наявні дані про продукт
        self.load_product_data()
    def update_product_list(self, product_info):
        self.product_list.insert(tk.END, product_info)

    def load_product_data(self):
        try:
            with open('product_data.csv', newline='') as file:
                reader = csv.reader(file)
                for row in reader:
                    self.product_list.insert(tk.END, row[0])
        except FileNotFoundError:
            messagebox.showinfo("File Not Found", "No product data found.")

class ReportTab(ttk.Frame):
    def __init__(self, parent):
```

```
        super().__init__(parent)
        self.parent = parent
        self.create_widgets()

    def create_widgets(self):
        self.report_id_label = ttk.Label(self, text="Звіт ID")
        self.report_id_entry = ttk.Entry(self)
        self.report_name_label = ttk.Label(self, text="Назва звіту")
        self.report_name_entry = ttk.Entry(self)
        self.report_manager_label = ttk.Label(self, text="Менеджер")
        self.report_manager_entry = ttk.Combobox(self)
        self.report_content_label = ttk.Label(self, text="Зміст звіту")
        self.report_content_entry = tk.Text(self, height=10, width=100)

        self.report_id_label.grid(row=0, column=0)
        self.report_id_entry.grid(row=0, column=1)
        self.report_name_label.grid(row=1, column=0)
        self.report_name_entry.grid(row=1, column=1)
        self.report_manager_label.grid(row=2, column=0)
        self.report_manager_entry.grid(row=2, column=1)
        self.report_content_label.grid(row=3, column=0)
        self.report_content_entry.grid(row=3, column=1)

        self.save_report_btn = ttk.Button(self, text="Зберегти звіт",
command=self.save_report)
        self.save_report_btn.grid(row=4, column=0, columnspan=2)

    def update_manager_combobox(self, manager_info):
        manager = list(self.report manager entry['values'])
        manager.append(manager_info)
        self.report_manager_entry['values'] = tuple(manager)

    def save_report(self):
        report_id = self.report_id_entry.get()
        report_name = self.report_name_entry.get()
        report_manager = self.report_manager_entry.get()
        report_content = self.report_content_entry.get("1.0", tk.END)

        if report_id and report_name and report_content and report_manager:
            file_name = f"report_{report_id}.txt"
            with open(file_name, "w") as file:
                file.write(f"Report ID: {report_id}\n")
                file.write(f"Report Name: {report_name}\n\n")
                file.write(f"Report Manager: {report_manager}\n\n")
                file.write(report_content)

            messagebox.showinfo("Звіт збережено", f"Report '{report_name}'
було збережено як '{file_name}'")
            self.clear_fields()
        else:
            messagebox.showerror("Неповні дані", "Будь ласка, заповніть усі
поля")

    def clear_fields(self):
        self.report_id_entry.delete(0, tk.END)
        self.report_name_entry.delete(0, tk.END)
        self.report_content_entry.delete("1.0", tk.END)


class Application(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Інформаційна система")
```

```
        self.geometry("800x600")

        self.tab_control = ttk.Notebook(self)
        self.tab_control.pack(fill="both", expand=True)

        # Існуючі вкладки
        self.order_tab = OrderTab(self.tab_control)
        self.customer_tab = CustomerTab(self.tab_control, self.order_tab)
        self.warehouse_tab = WarehouseTab(self.tab_control)
        self.product_tab = ProductTab(self.tab_control, self.order_tab,
self.warehouse_tab)
        self.employee_tab = EmployeeTab(self.tab_control, self.order_tab)
        self.manager_tab = ManagerTab(self.tab_control, self.order_tab, self)
        self.supplier_tab = SupplierTab(self.tab_control, self.product_tab)
        self.report_tab = ReportTab(self.tab_control)

        self.tab_control.add(self.customer_tab, text="Клієнт")
        self.tab_control.add(self.employee_tab, text="Працівник")
        self.tab_control.add(self.manager_tab, text="Менеджер")
        self.tab_control.add(self.supplier_tab, text="Постачальник")
        self.tab_control.add(self.order_tab, text="Замовлення")
        self.tab_control.add(self.product_tab, text="Продукт")
        self.tab_control.add(self.warehouse_tab, text="Склад")
        self.tab_control.add(self.report_tab, text="Звіт")

        self.populate_customer_combobox()
        self.populate_employee_combobox()
        self.populate_product_combobox()
        self.populate manager combobox()
        self.populate_supplier_combobox()

    def populate_customer_combobox(self):
        customers = [self.customer_tab.customer_list.get(idx) for idx in
range(self.customer_tab.customer_list.size())]
        self.order_tab.order_customer_entry['values'] = customers

    def populate_employee_combobox(self):
        employees = [self.employee_tab.employee_list.get(idx) for idx in
range(self.employee_tab.employee_list.size())]
        self.order_tab.order_employee_entry['values'] = employees

    def populate_product_combobox(self):
        products = [self.product_tab.product_list.get(idx) for idx in
range(self.product_tab.product_list.size())]
        self.order_tab.order_products_entry['values'] = products

    def populate_manager_combobox(self):
        manager = [self.manager_tab.manager_list.get(idx) for idx in
range(self.manager_tab.manager_list.size())]
        self.report_tab.report_manager_entry['values'] = manager

    def populate_supplier_combobox(self):
        supplier = [self.supplier_tab.supplier_list.get(idx) for idx in
range(self.supplier_tab.supplier_list.size())]
        self.product_tab.product_supplier_entry['values'] = supplier

    def switch_to_supplier_tab(self):
        self.tab_control.select(self.supplier_tab)

    def switch_to_product_tab(self):
            self.tab_control.select(self.product_tab)

if __name__ == "__main__":
```

```
    app = Application()
    app.mainloop()
```