МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Інститут (факультет) *Автоматизації і комп'ютерних систем*
Кафедра *Інформаційних систем*

«До захисту в ЕК»		**«До захисту допущено»**	
Директор інституту(декан факультету)		Завідувач кафедри	
_____	*Андрій Форсюк*	_____	*Сергій Чумаченко*
(підпис)	(ім'я та прізвище)	(підпис)	(ім'я та прізвище)
«___» _____ 2022 р.		«___» _____ 2022 р.	

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

зі спеціальності *122 «Комп'ютерні науки»*
(код та назва спеціальності)
освітньо-професійної програми *«Інформаційні управляючі системи та технології»*

на тему: *«Дослідження та розроблення інформаційно-аналітичної системи ТОВ «СПЕЦПІДЗЕМІНЖБУД»»*

Виконав: здобувач *2* курсу, групи *3*

Андреєв Владислав Миколайович _____
(прізвище, ім'я, по батькові повністю) (підпис)

Керівник *Попик Наталія Володимирівна* _____
(прізвище , ім'я та по батькові повністю) (підпис)

Консультанти *Наталія Попик* _____
(ім'я та прізвище) (підпис)

_____ _____
(ім'я та прізвище) (підпис)

_____ _____
(ім'я та прізвище) (підпис)

Рецензент *Віктор Сідлецький* _____
(ім'я та прізвище) (підпис)

Я як здобувач(ка) Національного університету харчових технологій розумію і підтримую політику університету з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволеної допомоги під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Здобувач_____
(підпис)

Київ - 2022 р.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Інститут (факультет) *Автоматизації і комп'ютерних систем*
Кафедра *Інформаційних систем*
Освітній ступінь *магістр*
Спеціальність *122 «Комп'ютерні науки»*
(код і назва)
Освітньо-професійна програма *«Інформаційні управляючі системи та технології»*
(назва)

ЗАТВЕРДЖУЮ
завідувач кафедри
інформаційних систем
с.н.с. С. М. Чумаченко
"11" листопада **2021 року**

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

АНДРЕЄВ Владислав Миколайович
(прізвище, ім'я, по батькові)

1. Тема роботи *«Дослідження та розроблення інформаційно-аналітичної системи ТОВ «СПЕЦПІДЗЕМІНЖБУД»»*
керівник роботи *Попик Наталія Володимирівна, к.т.н., доцент*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від "*11*" *листопада* 2021 року №*884-кс*
2. Строк подання здобувачем роботи *7 лютого 2022 року*
3. Вихідні дані до роботи *нормативно-правова база діяльності підприємства, інформація про постачальників, дані про товари, дані про розташування складських приміщень, інформація про завідувачів складських приміщень.*
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
1. Дослідження діяльності відділу постачання.
2. Дослідження наявних рішень для вирішення знайдених проблем та вибір наукових методів.
3. Алгоритмізація обраних методів.
4. Автоматизація і тестування методів та аналіз отриманих результатів.
5. Техніко-економічний ефект.
5. Перелік графічного матеріалу
1. Схема бази даних
2. Допоміжні діаграми
3. Системні форми

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 1	*к. т. н., доцент Попик Н. В.*		
Розділ 2	*к. т. н., доцент Попик Н. В.*		
Розділ 3	*к. т. н., доцент Попик Н. В.*		

7. Дата видачі завдання *11 листопада 2021*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів виконання кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	*Функціональне та концептуальне моделювання діяльності компанії ТОВ «СПЕЦПІДЗЕМІНЖБУД» та визначення проблемних задач.*	*22.11.2021*	
2	*Дослідження наукових робіт з рішеннями проблемних задач.*	*13.12.2021*	
4	*Опис та порівняння наукових рішень для розуміння їх використання.*	*12.01.2022*	
5	*Автоматизація обраних наукових рішень.*	*22.01.2022*	
6	*Аналіз отриманих результатів.*	*27.01.2022*	
7	*Розрахунок очікуваного економічного ефекту від впровадження розробки.*	*30.01.2022*	
8	*Оформлення пояснювальної записки.*	*31.01.2022*	
9	*Розробка презентації.*	*03.02.2022*	

Здобувач _____ *Андреєв В. М.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Попик Н. В.*
(підпис) (прізвище та ініціали)

ЗМІСТ

ВСТУП

Автоматизація підприємства включає в себе безліч одиниць компютерної техніки, кількість якої з кожним етапом модернізації стає тільки більшою. Обробка великої кількості даних(BigData), потреба у швидкому та не ресурсозаратному пошуку необхідної інформації, видання та друк звітності, що допомагає у прийнятті найоптимальніших рішень, саме це сприяє росту кількості та попиту на обчислювальну техніку, що підтримується інформаційно-обліковими системами.

У кожної компанії та підприємства на передньому плані завжди стоїть мета - досягнення максимальної ефективності виробництва, саме це надають інформаційні системи.

Створення сучасних інформаційних систем являє собою складну задачу, вирішення якої вимагає застосування спеціальних методик, навичок та інструментів.[1]

Існування такої системи однозначно спрощує роботу всім працівникам хто має до неї доступ, дозволяє автоматизувати процеси які до цього виконувалися вручну, з чого випливає зманшення впливу людського фактору, тобто менша кількість повсякденних помилок які на великих виробництвах коштують дуже дорого, а створенна інформаційна система окуповує себе неймовірно швидко за рахунок ряду факторів.

Не кожне підприємство готове викласти чималі кошти на придбання подібних систем, особливо, коли його фінансові позиції не досить стійкі. В такому випадку доцільно скористатись інформаційною системою, яка частково автоматизує роботу певної категорії фахівців, що обробляють значні об'єми інформації. В цьому разі розробляється локально орієнтована система з конкретно визначеним функціоналом, що немалозначне вона на порядок дешевше за ту яка б створювалась для автоматизації всього підприємства.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття інформаційної системи

Інформаційна система (ІС) - це формальна, соціально-технічна, організаційна система, призначена для збору, обробки, зберігання та розповсюдження інформації. [1] У соціотехнічній перспективі інформаційні системи складаються з чотирьох компонентів: завдання, людей, структури (або ролей) та технології [2]. Інформаційні системи можна визначити як інтеграцію компонентів для збору, зберігання та обробки даних, дані яких використовуються для надання інформації, сприяння знанням, а також цифрових продуктів. [3]

Комп’ютерна інформаційна система - це система, що складається з людей та комп’ютерів, яка обробляє або інтерпретує інформацію. [4] [5] [6] [7] Цей термін також іноді використовують для простого позначення комп'ютерної системи з встановленим програмним забезпеченням.

Інформаційні системи - це академічне дослідження систем із конкретним посиланням на інформацію та додаткові мережі апаратного та програмного забезпечення, які люди та організації використовують для збору, фільтрування, обробки, створення та розповсюдження даних. Акцент робиться на інформаційній системі, що має певні межі, користувачах, процесорах, сховищах, входах, виходах та згаданих мережах зв'язку [8].

У багатьох організаціях відділ або підрозділ, відповідальний за інформаційні системи та обробку даних, відомий як "Інформаційні послуги". [9] [10] [11] [12]

Будь-яка конкретна інформаційна система спрямована на підтримку операцій, управління та прийняття рішень. [13] [14] Інформаційна система - це інформаційно-комунікаційні технології (ІКТ), які використовує організація, а

також спосіб взаємодії людей із цією технологією для підтримки бізнес-процесів [15].

Деякі автори чітко розмежовують інформаційні системи, комп'ютерні системи та бізнес-процеси. Інформаційні системи, як правило, включають компонент ІКТ, але вони не стосуються лише ІКТ, а натомість зосереджуються на кінцевому використанні інформаційних технологій. Інформаційні системи також відрізняються від бізнес-процесів. Інформаційні системи допомагають контролювати ефективність бізнес-процесів. [16]

Альтер [17] [18] аргументує переваги розгляду інформаційної системи як особливого типу робочої системи. Робоча система - це система, в якій люди або машини виконують процеси та дії, використовуючи ресурси для виробництва конкретних продуктів чи послуг для клієнтів. Інформаційна система - це робоча система, діяльність якої спрямована на збір, передачу, зберігання, пошук, маніпулювання та відображення інформації. [19]

Як такі, інформаційні системи взаємопов'язані з системами даних, з одного боку, та системами діяльності, з іншого. [20] Інформаційна система - це форма комунікаційної системи, в якій дані представляють і обробляються як форма соціальної пам'яті. Інформаційну систему також можна вважати напівформальною мовою, яка підтримує прийняття рішень і дій людини.

Інформаційні системи є основним напрямком вивчення організаційної інформатики. [21]

Срібло та ін. (1995) надав два погляди на ІС, що включає програмне забезпечення, апаратне забезпечення, дані, людей та процедури. [22] Чжен надав інший системний погляд на інформаційну систему, який також додає процеси та основні елементи системи, такі як середовище, межі, призначення та взаємодії.

Асоціація обчислювальних машин визначає "Фахівців інформаційних систем [як] зосереджують увагу на інтеграції рішень інформаційних технологій та бізнес-процесів для задоволення інформаційних потреб підприємств та інших підприємств" [23].

Існують різні типи інформаційних систем, наприклад: системи обробки транзакцій, системи підтримки прийняття рішень, системи управління знаннями, системи управління навчанням, системи управління базами даних та офісні інформаційні системи. Критичними для більшості інформаційних систем є інформаційні технології, які, як правило, розроблені для того, щоб люди могли виконувати завдання, для яких людський мозок погано підходить, такі як: обробка великої кількості інформації, виконання складних обчислень та контроль багатьох одночасних процесів.

Інформаційні технології - це дуже важливий та податливий ресурс, доступний керівникам. [24] Багато компаній створили посаду головного інформаційного директора (CIO), який засідає у виконавчому комітеті разом із головним виконавчим директором (CEO), головним фінансовим директором (CFO), головним операційним директором (COO) та головним технічним директором (CTO). Технічний директор може також виконувати функції IT-директора, і навпаки. Головний директор з інформаційної безпеки (CISO) зосереджується на управлінні інформаційною безпекою.

Шість компонентів, які повинні об’єднатися для створення інформаційної системи:

- Апаратне забезпечення: Термін апаратне забезпечення стосується машин та обладнання. У сучасній інформаційній системі до цієї категорії належить сам комп'ютер та все його допоміжне обладнання. До допоміжного обладнання належать пристрої введення та виведення, пристрої зберігання даних та пристрої зв'язку. У докомп’ютерних інформаційних системах апаратне забезпечення може включати книги книги та чорнило.
- Програмне забезпечення: Термін програмне забезпечення означає комп'ютерні програми та посібники (якщо такі є), які їх підтримують. Комп’ютерні програми - це машиночитані інструкції, які спрямовують схеми всередині апаратних частин системи на функціонування таким

чином, щоб отримувати корисну інформацію з даних. Зазвичай програми зберігаються на певному носії вводу / виводу, часто на диску або касеті. "Програмне забезпечення" для докомп'ютерних інформаційних систем включало те, як обладнання було підготовлено до використання (наприклад, заголовки стовпців у книзі книги) та інструкції щодо їх використання (довідник для каталогу карток).

- Дані: Дані - це факти, які використовуються системами для отримання корисної інформації. У сучасних інформаційних системах дані, як правило, зберігаються у машиночитаному вигляді на диску або стрічці, поки комп’ютер не потребує їх. У докомп’ютерних інформаційних системах дані, як правило, зберігаються в зручній для читання формі.
- Процедури: Процедури - це політики, що регулюють роботу інформаційної системи. "Процедури для людей - це те, що програмне забезпечення - для апаратного забезпечення" - загальна аналогія, яка використовується для ілюстрації ролі процедур у системі.
- Люди: Кожна система потребує людей, щоб бути корисною. Часто найбільш ігноруваним елементом системи є люди, ймовірно, той компонент, який найбільше впливає на успіх чи провал інформаційних систем. Сюди входять "не тільки користувачі, але й ті, хто експлуатує та обслуговує комп’ютери, ті, хто підтримує дані, і ті, хто підтримує мережу комп’ютерів" [25].
- Зворотній зв’язок: це інший компонент ІС, який визначає, що ІС може надавати зворотній зв’язок (Хоча цей компонент не потрібний для функціонування).

1.2 Прикладне програмне забезпечення

Прикладне програмне забезпечення (коротше додаток) - це обчислювальне програмне забезпечення, призначене для виконання конкретного завдання, відмінного від того, що стосується роботи самого комп'ютера, [1] як правило, для використання кінцевими користувачами. Прикладами програми є текстовий процесор та медіаплеєр. Прикладне програмне забезпечення колективного іменника відноситься до всіх додатків у сукупності. [2] Інші основні класифікації програмного забезпечення - це системне програмне забезпечення, що стосується роботи комп'ютера, та утиліта ("утиліти").

Програми можуть поставлятися в комплекті з комп'ютером та його системним програмним забезпеченням або публікуватися окремо і можуть кодуватися як власні, відкриті джерела чи проекти. [3] Програми, створені для мобільних платформ, називаються мобільними програмами.

В інформаційних технологіях додаток (додаток), прикладна програма або прикладне програмне забезпечення - це комп’ютерна програма, призначена для допомоги людям у здійсненні тієї чи іншої діяльності. Залежно від діяльності, для якої він був розроблений, програма може маніпулювати текстом, цифрами, аудіо, графікою та комбінацією цих елементів. Деякі пакети програм зосереджені на одному завданні, наприклад, на обробці текстів; інші, що називаються інтегрованим програмним забезпеченням, включають кілька програм.

Написане користувачем програмне забезпечення адаптує системи для задоволення конкретних потреб користувача. Написане користувачем програмне забезпечення включає шаблони електронних таблиць, макроси текстового процесора, наукове моделювання, аудіо, графіку та сценарії анімації. Навіть фільтри електронної пошти є своєрідним програмним забезпеченням для користувачів. Користувачі самі створюють це програмне забезпечення і часто не помічають, наскільки воно важливо.

Однак розмежування між системним програмним забезпеченням, таким як операційна система, та прикладним програмним забезпеченням не є точним і іноді є предметом суперечок [5]. Наприклад, одним із ключових запитань у справі «Сполучені Штати проти Microsoft Corp.» про антимонопольний закон було те, чи є веб-браузер Microsoft Internet Explorer частиною його операційної системи Windows або окремим програмним забезпеченням. Як інший приклад, суперечка щодо імен GNU / Linux, зокрема, зумовлена незгодою щодо взаємозв'язку між ядром Linux та операційними системами, побудованими над цим ядром. У деяких типах вбудованих систем прикладне програмне забезпечення та програмне забезпечення операційної системи можуть не відрізняти користувача, як у випадку програмного забезпечення, що використовується для управління відеомагнітофоном, DVD-програвачем або мікрохвильовою піччю. Наведені вище визначення можуть виключати деякі програми, які можуть існувати на деяких комп’ютерах у великих організаціях.

Слово "додаток", що вживається як прикметник, не обмежується лише значенням "прикладного програмного забезпечення" або що стосується його "[6]. Наприклад, такі поняття, як інтерфейс прикладного програмування (API), сервер додатків, віртуалізація додатків, управління життєвим циклом додатків та портативний додаток стосуються всіх комп’ютерних програм, а не лише прикладного програмного забезпечення.

Деякі програми доступні у версіях для декількох різних платформ; інші працюють лише над одним і тому їх називають, наприклад, географічним додатком для Microsoft Windows, або Android-додатком для навчання, або грою для Linux. Іноді виникає новий і популярний додаток, який працює лише на одній платформі, збільшуючи бажаність цієї платформи. Це називається вбивцею або вбивцею. Наприклад, VisiCalc був першим сучасним програмним забезпеченням для роботи з електронними таблицями для Apple II і допоміг продати нові тоді персональні комп’ютери в офіси. Для Blackberry це було програмне забезпечення для електронної пошти.

В останні роки скорочений термін "додаток" (створений у 1981 році або раніше [7]) став популярним для позначення додатків для мобільних пристроїв, таких як смартфони та планшети, скорочена форма відповідає їх типово меншому обсягу порівняно з програмами на ПК. Ще нещодавно, скорочена версія використовується і для настільного програмного забезпечення.

Існує багато різних та альтернативних способів класифікації прикладного програмного забезпечення.

З юридичної точки зору, прикладне програмне забезпечення в основному класифікується з використанням чорного ящика щодо прав кінцевих кінцевих користувачів або абонентів (з можливим проміжним та багаторівневим рівнями передплати).

Програмні додатки класифікуються також за мовою програмування, на якій пишеться або виконується вихідний код, а також за призначенням та результатами.

За правами власності та користування

Прикладне програмне забезпечення, як правило, виділяють серед двох основних класів: програмне забезпечення із закритим вихідним кодом та програмне забезпечення з відкритим кодом, а також безкоштовне програмне забезпечення або власне програмне забезпечення.

Запатентоване програмне забезпечення підпадає під виключне авторське право, а ліцензія на програмне забезпечення надає обмежені права на використання. Принцип відкрито-закрито стверджує, що програмне забезпечення може бути "відкритим лише для розширення, але не для модифікації". Такі додатки можуть отримувати доповнення лише сторонніми сторонами.

Вільне програмне забезпечення та програмне забезпечення з відкритим кодом слід запускати, розповсюджувати, продавати або розширювати з будь-якою метою, і, будучи відкритим, слід модифікувати або скасовувати таким же чином.

Програмні додатки FOSS, що випускаються за безкоштовною ліцензією, можуть бути безстроковими та також безкоштовно. Можливо, власник, власник або стороння особа, яка виконує будь-які права (авторське право, торгова марка, патент або ius in re aliena), має право додавати винятки, обмеження, затримки чи дати закінчення до ліцензійних умов використання.

Програмне забезпечення публічного домену - це тип FOSS, який є безоплатним і - відкрито чи зарезервовано - може запускатися, розповсюджуватися, модифікуватися, змінюватися, перевидаватися або створюватися у похідних роботах без будь-якого приписування авторських прав і, отже, відкликання. Його можна навіть продати, але без передачі власності, що перебуває у відкритому доступі, іншим окремим суб'єктам. Загальнодоступне програмне забезпечення може бути випущено відповідно до юридичної заяви про (не) ліцензування, яка застосовує ці умови протягом невизначеного періоду (протягом усього життя або назавжди).

За допомогою мови кодування

З моменту розробки та майже універсального прийняття Інтернету, важливим розрізненням, яке з'явилося, було між веб-додатками - написаними за допомогою HTML, JavaScript та іншими власними веб-технологіями, які, як правило, вимагають наявності в Інтернеті та запуску веб-браузера - і більш традиційні рідні програми, написані будь-якими мовами, доступними для конкретного типу комп'ютерів. У комп'ютерному співтоваристві ведуться суперечки щодо веб-додатків, які замінюють рідні програми для багатьох цілей, особливо на мобільних пристроях, таких як смартфони та планшети. Веб-програми дійсно значно зросли в популярності для деяких цілей, але переваги програм роблять їх навряд чи зникнуть найближчим часом, якщо взагалі коли-небудь. Крім того, вони можуть доповнювати і навіть інтегрувати. [8] [9] [10]

За призначенням і результатом

Прикладне програмне забезпечення також можна розглядати як горизонтальне або вертикальне. [11] [12] Горизонтальні програми є більш

популярними та поширеними, оскільки вони є загальним призначенням, наприклад, текстові процесори або бази даних. Вертикальні додатки - це нішеві продукти, призначені для певного виду галузі чи бізнесу або відділу в організації. Інтегровані пакети програмного забезпечення намагатимуться розглянути всі можливі аспекти, наприклад, виробничого або банківського працівника, бухгалтерії чи обслуговування клієнтів.

Існує багато типів прикладного програмного забезпечення: [13]

- Набір програм складається з декількох додатків, що входять до комплекту. Зазвичай вони мають пов’язані функції, функції та користувальницькі інтерфейси і можуть мати можливість взаємодіяти між собою, наприклад відкривати файли один одного. Бізнес-програми часто бувають в люксах, напр. Microsoft Office, LibreOffice та iWork, які об’єднують текстовий процесор, електронну таблицю тощо; але люкси існують для інших цілей, наприклад графіка або музика.
- Корпоративне програмне забезпечення відповідає потребам процесів і потоків даних усієї організації в декількох відділах, часто у великому розподіленому середовищі. Приклади включають системи планування ресурсів підприємства, системи управління взаємовідносинами з клієнтами (CRM) та програмне забезпечення для управління ланцюгами поставок. Департаментське програмне забезпечення - це підвид корпоративного програмного забезпечення, орієнтоване на менші організації чи групи у великій організації. (Приклади включають управління витратами на проїзд та IT-довідкову службу.)
- Корпоративне інфраструктурне програмне забезпечення забезпечує загальні можливості, необхідні для підтримки корпоративних програмних систем. (Приклади включають бази даних, сервери електронної пошти та системи управління мережами та безпекою.)

- Прикладна платформа як послуга (aPaaS) - це служба хмарних обчислень, яка пропонує середовища розробки та розгортання служб додатків.
    - Програмне забезпечення інформаційного працівника дозволяє користувачам створювати та керувати інформацією, часто для окремих проектів у відділі, на відміну від управління підприємством. Приклади включають управління часом, управління ресурсами, аналітичні засоби, інструменти спільної роботи та документації. Текстові процесори, електронні таблиці, клієнти електронної пошти та блогу, система персональної інформації та окремі редактори засобів масової інформації можуть допомогти у виконанні багатьох завдань інформаційного працівника.
    - Програмне забезпечення для доступу до вмісту використовується в основному для доступу до вмісту без редагування, але може включати програмне забезпечення, яке дозволяє редагувати вміст. Таке програмне забезпечення відповідає потребам окремих людей та груп у споживанні цифрових розваг та опублікованого цифрового вмісту. (Приклади включають медіаплеєри, веб-браузери та браузери довідки.)
    - Навчальне програмне забезпечення пов’язане із програмним забезпеченням доступу до вмісту, але має вміст або функції, пристосовані для використання викладачами чи студентами. Наприклад, він може проводити оцінки (тести), відстежувати прогрес матеріалу або включати можливості спільної роботи.
    - Програмне забезпечення для моделювання імітує фізичні або абстрактні системи для дослідницьких, навчальних або розважальних цілей.

- Програмне забезпечення для розробки засобів масової інформації створює друковані та електронні носії для споживання іншими, найчастіше в комерційних або освітніх умовах. Сюди входить програмне забезпечення для графічного мистецтва, програмне забезпечення для настільних видавництв, програмне забезпечення для розробки мультимедіа, редактори HTML, редактори цифрової анімації, цифрові композиції аудіо та відео та багато інших.
- Програмне забезпечення для розробки продуктів використовується при розробці апаратних та програмних продуктів. Це включає автоматизоване проектування (САПР), автоматизоване проектування (CAE), засоби редагування та компіляції комп'ютерної мови, інтегровані середовища розробки та інтерфейси програмістів програм.
- Розважальне програмне забезпечення може стосуватися відеоігор, заставки, програм для відображення кінофільмів або відтворення записаної музики та інших видів розваг, які можна відчути за допомогою використання обчислювального пристрою.

Програми також можна класифікувати за обчислювальними платформами, такими як конкретна операційна система, мережа доставки, наприклад, у хмарних обчисленнях та додатки Web 2.0, або пристрої доставки, такі як мобільні програми для мобільних пристроїв.

Саму операційну систему можна вважати прикладним програмним забезпеченням при виконанні простих завдань обчислення, вимірювання, рендерингу та обробки текстів, які не використовуються для управління апаратним забезпеченням за допомогою інтерфейсу командного рядка або графічного інтерфейсу користувача. Сюди не входить прикладне програмне забезпечення, що входить до складу операційних систем, таких як калькулятор програмного забезпечення або текстовий редактор.

Розробка програмного забезпечення - це процес задуму, уточнення, проектування, програмування, документування, тестування та виправлення помилок, що беруть участь у створенні та обслуговуванні програм, фреймворків чи інших компонентів програмного забезпечення. Розробка програмного забезпечення - це процес написання та підтримання вихідного коду, але в більш широкому розумінні він включає все, що пов’язане від задуму бажаного програмного забезпечення до остаточного прояву програмного забезпечення, іноді у запланованому та структурованому процесі. [1] Тому розробка програмного забезпечення може включати дослідження, нові розробки, створення прототипів, модифікацію, повторне використання, реінжиніринг, технічне обслуговування або будь-яку іншу діяльність, результатом якої є програмні продукти. [2]

Програмне забезпечення може бути розроблено для різних цілей, серед яких три найпоширеніші - для задоволення конкретних потреб конкретного клієнта / бізнесу (у випадку зі спеціальним програмним забезпеченням), для задоволення сприйнятих потреб певного набору потенційних користувачів (у випадку з комерційними та програмне забезпечення з відкритим кодом), або для особистого користування (наприклад, вчений може написати програмне забезпечення для автоматизації буденного завдання). Розробка вбудованого програмного забезпечення, тобто розробка вбудованого програмного забезпечення, такого як використовується для контролю споживчих товарів, вимагає інтеграції процесу розробки з розробкою контрольованого фізичного продукту. Системне програмне забезпечення лежить в основі програм і самого процесу програмування, і часто розробляється окремо.

Потреба в кращому контролі якості процесу розробки програмного забезпечення породила дисципліну програмної інженерії, яка спрямована на застосування системного підходу, наведеного в інженерній парадигмі, до процесу розробки програмного забезпечення.

Існує багато підходів до управління програмними проектами, відомих як моделі життєвого циклу розробки програмного забезпечення, методології,

процеси чи моделі. Модель водоспаду є традиційною версією, на відміну від останніх інновацій у гнучкій розробці програмного забезпечення.

1.3 Загальна характеристика підприємства

Історія підприємства бере початок з 2014р. З того часу підприємство є одним з провідних гравців на ринку робіт з обладнання бурових паль та армованих бетонних опорних стін методом «Стіна в ґрунті».

Організаційно-правовою формою даного підприємства є Товариство з обмеженою відповідальністю, форма власності – приватна.

Галузь – будівельно-фундаменті роботи.

Метою створення Товариства є отримання прибутку від діяльності, в тому числі одержаного в українській та іноземній валюті. Товариство створено для здійснення господарської діяльності у сфері виробничих, торговельних, фінансових та посередницьких операцій, надання відповідних послуг із консультаціями включно та здійснення іншої господарської діяльності.

Товариство досягає своєї мети шляхом здійснення різноманітних видів господарської діяльності.

Товариство може залучатись до інших видів не заборонених законодавством України, які на думку Учасників відповідають меті Товариства та є прибутковими.

Облікова політика товариства забезпечується згідно вимогам Закону України «Про бухгалтерський облік та фінансову звітність в Україні» та наказом «Про організацію бухгалтерського обліку та облікової політики» в 2014 році по ТОВ «Спецпідземінжбуд».

ТОВ «Спецпідземінжбуд» – приватне самоокупне підприємство, що використовує власні обігові кошти та залучає, частково, кредитні ресурси комерційних банків для забезпечення виробничо-фінансової діяльності та модернізації, оновлення основних фондів підприємства.[7]

Виробничі процеси, за допомогою яких відтворюються основні види діяльності, є головними і утворюють базову, основну частину виробництва. Матеріальними об'єктами виробничої структури підприємства є склади, виробничі дільниці, офісні будівлі. На складі накопичується та розподіляється основна частина витратних матеріалів та інструментів. На виробничій

дільниці здійснюється безпосередньо відтворення основних видів виробничої діяльності. У офісних будівлях відбувається основна координація, комунікація між об’єктами та відділами, планування та нормування виробничого процесу.

Технологічний процес і набір сучасного обладнання забезпечує високу і стабільну якість продукту, що дає змогу задовільнити вимоги найвибагливіших замовників, а також дозволяє в цілому механізувати виробничий процес та забезпечити систему безпеки праці.

Виробничі процеси, за допомогою яких відтворюються основні види діяльності, є головними і утворюють базову, основну частину виробництва. Матеріальними об'єктами виробничої структури підприємства є склади, виробничі дільниці, офісні будівлі. На складі накопичується та розподіляється основна частина витратних матеріалів та інструментів. На виробничій дільниці здійснюється безпосередньо відтворення основних видів виробничої діяльності. У офісних будівлях відбувається основна координація, комунікація між об’єктами та відділами, планування та нормування виробничого процесу.

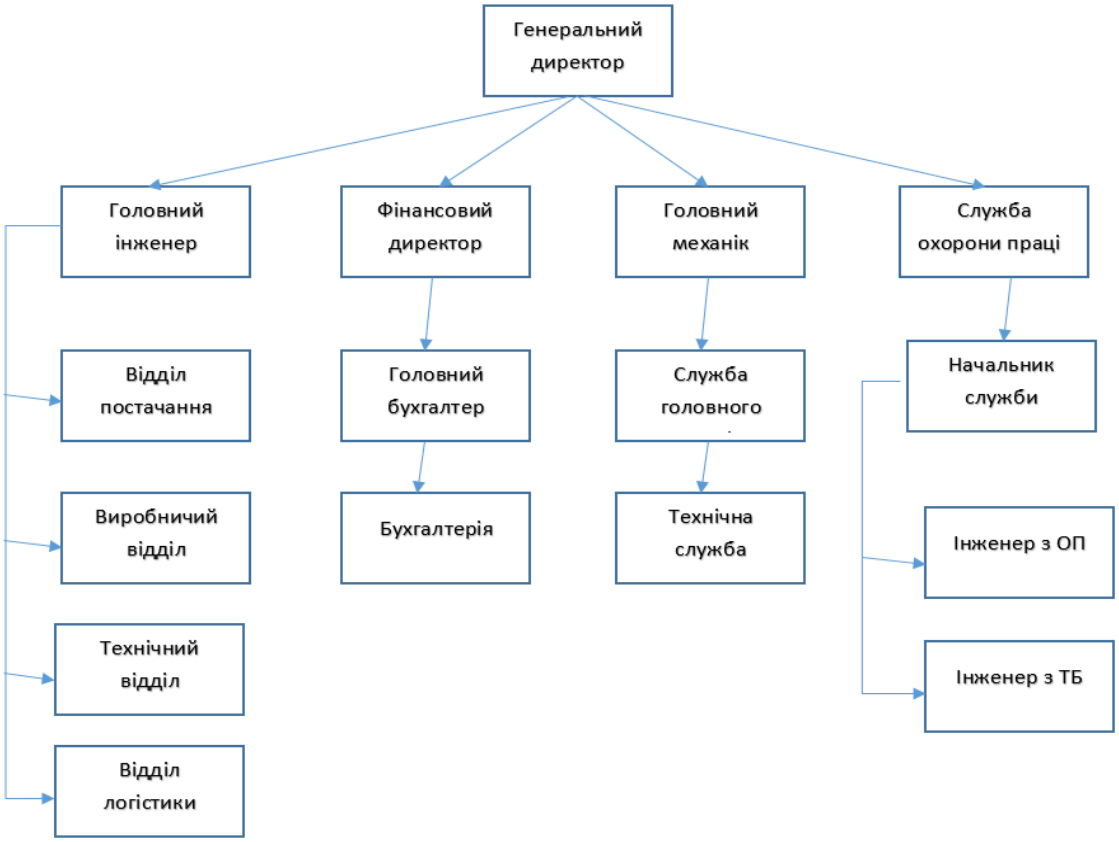

Рисунок 1.1 — Ієрархія ТОВ «Спецпідземінжбуд»

Відділ постачання відіграє одну з фундаментальних ролей в діяльності підприємства, адже без сировини, обладнання і техніки ніякий інший відділ не зможе повноцінно і ефективно функціонувати.

До підпорядкування відділу входять: офісне та складське приміщення.

Рисунок 1.2 — Ієрархія відділу постачання

1.4 Огляд існуючих рішень

1.4.1 1С:Підприємство

Система програм "1С:Підприємство" включає в себе платформу і прикладні рішення, розроблені на її основі, для автоматизації діяльності організацій і приватних осіб. Сама платформа не є програмним продуктом для використання кінцевими користувачами, які зазвичай працюють з одним з багатьох прикладних рішень (конфігурацій), розроблених на даній платформі. Такий підхід дозволяє автоматизувати різні види діяльності, використовуючи єдину технологічну платформу.[9]

Основні можливості програмного продукту:

- Ведення обліку декількох організацій в єдиній інформаційній базі
- Облік «від документу», застосування типових операцій
- Партіонний облік
- Складський облік
- Облік торговельних операцій
- Облік комісійної торгівлі
- Облік операцій з тарою
- Облік банківських і касових операцій
- Облік розрахунків з контрагентами
- Облік основних засобів, нематеріальних і малоцінних активів
- Облік основного й допоміжного виробництва
- Облік непрямих витрат
- Облік напівфабрикатів
- Облік ПДВ
- Облік заробітної плати, кадровий облік
- Підтримка різних схем оподаткування
- Податковий облік з податку на прибуток
- Спрощена система оподаткування
- Завершальні операції періоду

- Стандартні бухгалтерські звіти [10]

1.4.2 УкрСклад

УкрСклад - програмний продукт, що дозволяє вести складський облік по декількох фірмах і складах.

Програма має наступні можливості:

- Виписка і друк наступних документів:
  - Прибуткова накладна
  - Рахунок-форма замовлення
  - Рахунок-фактура
  - Видаткова накладна
- Багатовалютність
- Виписка документів
- Ведення архіву
- Можливість створення групи пов'язаних документів[11]

1.4.3 Storecalc

Вкрай проста складська програма, складський калькулятор. Дозволяє задіяті комп'ютер для невелікої власної справи та організуваті елементарних складський облік. Програма НЕ друкує бухгалтерських документів типу накладних, її призначення - вести облік на складі «для себе». Лише декілька натискань клавіш і віконуються наступні дії:

- Додавання товару на склад
- Віднімання товару зі складу
- Зміна ціни товару
- Пошук товару
- Сортуваннях списку товарів
- Переглядання рухів товару
- Видалення товару зі складу [12]

1.5 Постановка задачі

За результатами аналізу аналогів встановлено, що на даний час не належним чином здійснюється прийом та обік товарів і обладнання на складских приміщеннях, що призводить до не ефективної роботи складу загалом, адже час затрачений на ручне внесення, облік та виконання необхідних видів запитів чи сортування потребують як людських так і значних часових ресурсів.

Виявлені недоліки:

- Немає однієї синхронізованої БД куди вноситься весь товар який є і той що надходить на склад.
- Завжди є загроза помилки через людський фактор.
- Створення звітності займає багато часу, що є не ефективним.

Рішення:

- Створена інформаційно-облікова система має доступ до однієї синхронізованої між усіма складськими приміщеннями БД.
- Ризик виникнення помилки через людський фактор на порядок зменшується через реалізованні обмеження вводу та редагування даних.
- Створення звітності відбуівється буквально в «один клік».

Очікуванні джерела економічного ефекту:

- Економія коштів за рахунок піднання швидкодії процесів прийняття та внесення товару, тобто чим швидше все відбувається тим більше одиниць товарів і техніки склад зможе обробити.
- Зменшення часу на формування звітності, фільтрування та пошуку за певним критерієм, що значно пришвидшить комунікацію між відділами, що в свою чергу в кінцевому результаті прискорить виконання робіт інших частин підприємства.
- Зменшення ризику виникнення помилок через людський фактор за допомогою контрольованого введення та редагування даних зменшить витрати що раніше були направленні на виправлення цих помилок.

Після визначення основних існуючих проблем відділу постчання я виокремив ряд задач які потребують автоматизації за рахунок впровадження інформаційно-облікової системи для складського приміщення.

Основні задачі на автоматизацію системи:

- Внесення нових одиниць товару та техніки, що прибувають на склад.
- Облік.
- Коригувати/видалення одиниць товару та техніки, що знаходяться на складі.
- Доступ до інформації/бази даних складу за допомогою впровадження авторизації та автентифікації користувача / адміністратора.
- Формування та облік накладних і звітів.

РОЗДІЛ 2
ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Вибір мови програмування

C # (вимовляється як музична нота C♯, але написаний зі знаком числа) - це універсальна багатопарадігмальна мова програмування, що включає строгу типізацію, лексичну область видимості, імператив, декларативний, функціональний, універсальний об'єкт -орієнтовані (на основі класів) та компонентно-орієнтовані дисципліни програмування. Він був розроблений Microsoft приблизно в 2000 році в рамках ініціативи .NET, а потім затверджений в якості міжнародного стандарту Ecma (ECMA-334) і ISO (ISO / IEC 23270: 2018). Mono - це назва безкоштовного проекту з відкритим вихідним кодом для розробки компілятора і середовища виконання для мови. C # є одним з мов програмування, розроблених для Common Language Infrastructure (CLI).

C # був розроблений Андерсом Хейлсбергом, а його командою розробників в даний час керує Мадс Торгерсен. Остання версія - 8.0, випущена в 2019 році разом з Visual Studio 2019 версії 16.3.

Під час розробки .NET Framework бібліотеки класів спочатку були написані з використанням компілятора системи керованого коду під назвою «Simple Managed C» (SMC). У січні 1999 року Андерс Хейлсберг сформував команду для створення нової мови під назвою Cool, який позначав «C-подібна об'єктно-орієнтована мова». Microsoft розглядала збереження назви «Cool» в якості остаточної назви мови, але вирішила не робити цього з причин, пов'язаних з товарними знаками. До того часу, коли проект .NET був публічно оголошено на конференції професійних розробників в липні 2000 року, мову був перейменований в C #, а бібліотеки класів і середовище виконання ASP.NET були портований на C #.

Хейлсберг - головний дизайнер і головний архітектор C # в Microsoft, раніше займався проектуванням Turbo Pascal, Embarcadero Delphi (раніше CodeGear Delphi, Inprise Delphi і Borland Delphi) і Visual J ++. У своїх інтерв'ю і технічних статтях він заявив, що недоліки в більшості основних мов програмування (наприклад, C ++, Java, Delphi і Smalltalk) призвели до основ Common Language Runtime (CLR), що, в свою чергу, призвело до розробки сама мова C #.

З часу випуску C # 2.0 в листопаді 2005 року мови C # і Java розвивалися за все більш і більш розбіжним траєкторіях, ставши двома зовсім різними мовами. Одним з перших основних відхилень стало додавання узагальнень до обох мов з абсолютно різними реалізаціями. C # використовує reification для надання «першокласних» універсальних об'єктів, які можна використовувати як будь-який інший клас, з генерацією коду, що виконується під час завантаження класу. Крім того, в C #, додано кілька основних функцій для програмування функціонального стилю, кульмінацією яких є розширення LINQ, випущені в C # 3.0, і підтримуюча його структура лямбда-виразів, методів розширення і анонімних типів. Ці функції дозволяють програмістам C # використовувати методи функціонального програмування, такі як замикання, коли це вигідно для їх застосування. Розширення LINQ і функціональний імпорт допомагають розробникам скоротити обсяг стандартного коду, який включається в загальні завдання, такі як запити до бази даних, аналіз файлу XML або пошук в структурі даних, зміщуючи акцент на реальну логіку програми, щоб поліпшити читаність і ремонтопридатність.

По своєму дизайну C # є мовою програмування, який безпосередньо відображає основну інфраструктуру спільної мови (CLI). Більшість його внутрішніх типів відповідають типам значень, реалізованих середовищем CLI. Однак специфікація мови не містить вимог до генерації коду компілятора, тобто не вказує, що компілятор C # має орієнтуватися на середовище виконання спільної мови, або генерувати загальний проміжний мова (CIL), або

генерувати будь-який інший конкретний формат. Теоретично, компілятор C # може генерувати машинний код, як традиційні компілятори C ++ та Fortran.

C # підтримує строго типізовані оголошення неявних змінних з ключовим словом var, а також неявно типізовані масиви з ключовим словом new [], за яким слід ініціалізатор колекції.

C # підтримує суворий логічний тип даних, bool. Оператори, які приймають умови, такі як while і if, вимагають вирази типу, що реалізує оператор true, такого як логічний тип. Хоча C ++ також має логічний тип, він може вільно перетворюватися в цілі і з цілих чисел, а вирази, наприклад, якщо (a) вимагають тільки, щоб a був перетворений в bool, дозволяючи a бути int або покажчиком. C # забороняє цей підхід «цілочисельне значення істина або брехня» на тій підставі, що примус програмістів використовувати вирази, які повертають саме bool, може запобігти певні типи помилок програмування, таких як if (a = b) (використання присвоювання = замість рівності ==).

C # більш безпечний для типів, ніж C ++. Єдиними неявними перетвореннями за замовчуванням є ті, які вважаються безпечними, наприклад, розширення цілих чисел. Це застосовується під час компіляції, під час JIT і, в деяких випадках, під час виконання. Не відбувається неявного перетворення між логічними значеннями і цілими числами, а також між членами перерахування і цілими числами (за винятком літерала 0, який може бути неявно перетворений в будь-який перераховується тип). Будь кероване перетворення повинне бути чітко позначено як явне або неявне, на відміну від конструкторів копіювання C ++ та операторів перетворення, які за замовчуванням неявні.

C # має явну підтримку коваріації і контраваріантності в універсальних типах, на відміну від C ++, який має деяку ступінь підтримки контраваріантності просто завдяки семантиці повертаються типів віртуальних методи.

Учасники перерахування розміщуються у своїй області видимості.

Мова C # не допускає глобальних змінних або функцій. Всі методи і члени повинні бути оголошені всередині класів. Статичні члени відкритих класів можуть замінювати глобальні змінні і функції.

Локальні змінні не можуть приховувати змінні вміщує блоку, на відміну від C та C ++.

C # має підтримку строго типізованих покажчиків на функції через ключове слово делегат. Як і псевдо-C ++ фреймворк Qt, в C # є семантика, зокрема навколишнє події стилю публікації-підписки, хоча C # використовує для цього делегати.

Керована пам'ять не може бути звільнена; замість цього він автоматично збирається. Збірка сміття вирішує проблему витоків пам'яті, позбавляючи програміста від відповідальності за звільнення пам'яті, яка більше не потрібна.

На відміну від C ++, C # не підтримує множинне успадкування, хоча клас може реалізовувати будь-яку кількість інтерфейсів. Це було дизайнерське рішення провідного архітектора мови, щоб уникнути ускладнення і спростити архітектурні вимоги у всьому CLI. При реалізації декількох інтерфейсів, які містять метод з однаковою сигнатурою, і. тобто два методу з одним і тим же ім'ям, що приймають параметри одного й того ж типу в одному і тому ж порядку, C # дозволяє реалізовувати кожен метод в залежності від того, через який інтерфейс викликається цей метод, або, як Java, дозволяє реалізувати метод один раз і мати один виклик за викликом через будь-який з інтерфейсів класу.

Однак, на відміну від Java, C # підтримує перевантаження операторів. Тільки найбільш часто перевантажені оператори в C ++ можуть бути перевантажені в C #.

C # має можливість використовувати LINQ через .NET Framework. Розробник може запросити будь-який об'єкт IEnumerable <T>, документи XML, набір даних ADO.NET і бази даних SQL. [60] Використання LINQ в C # дає такі переваги, як підтримка Intellisense, потужні можливості фільтрації, безпека типів з можливістю перевірки помилок компіляції і узгодженість

даних для запиту з різних джерел. Є кілька різних мовних структур, які можна використовувати з C # з LINQ, і вони є виразами запитів, лямбда-виразами, анонімними типами, неявно типізованими змінними, методами розширення і ініціалізаторами об'єктів

У серпні 2001 року корпорації Microsoft, Hewlett-Packard і Intel Corporation виступили співавторами в уявленні специфікації для C #, а також інфраструктури спільної мови (CLI) в організацію по стандартизації Ecma International. У грудні 2001 року ECMA випустила специфікацію мови ECMA-334 C #. C # став стандартом ISO у 2003 році (ISO / IEC 23270: 2003 Інформаційні технології. Мови програмування - C #). ECMA раніше прийняла еквівалентні специфікації як друге видання C # в грудні 2002 року.

У червні 2005 року ECMA схвалив видання 3 специфікації C # і оновило ECMA-334. Доповнення включали в себе часткові класи, анонімні методи, нульові типи і універсальні шаблони (щось схоже на шаблони C ++).

У липні 2005 року ECMA представила в ICO / МЕК JTC 1 за допомогою прискореного процесу останнього стандарти та відповідні ТЗ. Цей процес зазвичай займає 6-9 місяців.

Визначення мови C # і CLI стандартизовані у відповідності зі стандартами ISO і Ecma, які забезпечують розумну та недискримінаційну ліцензійну захист від патентних претензій.

Microsoft погодилася не пред'являти позов розробникам програмного забезпечення з відкритим вихідним кодом за порушення патентів в некомерційних проектах в частині структури, охоплюваній OSP. Microsoft також погодилася не застосовувати патенти на продукти Novell щодо платять клієнтів Novell, за винятком списку продуктів, в яких явно не згадується C#.NET або реалізація Novell .NET (The Mono Project). Тим не менш, Novell стверджує, що Mono не порушує жодних патентів Microsoft. Microsoft також уклала конкретну угоду про відмову у захисту патентних прав, пов'язаних з плагіном браузера Moonlight, який залежить від Mono, якщо він отриманий через Novell

Microsoft очолює розробку еталонного компілятор C # з відкритим вихідним кодом і набору інструментів, що раніше носили кодова назва "Roslyn". Компілятор, який повністю написаний на керованому коді (C #), був відкритий, а функціональність відображена як API. Це дозволяє розробникам створювати інструменти рефакторінгу та діагностики.

Інші компілятори C # (деякі з яких включають реалізацію інфраструктури спільної мови і бібліотек класів .NET):

• Проект Mono надає компілятор C # з відкритим вихідним кодом, повну реалізацію загальномовної інфраструктури з відкритим вихідним кодом, включаючи необхідні бібліотеки інфраструктури, як вони зазначені в специфікації ECMA, і майже повну реалізацію пропрієтарних бібліотек класів Microsoft .NET до .NET 3.5. Починаючи з Mono 2.6, планів по впровадженню WPF не існує; WF планується до більш пізнього випуску; і є лише часткові реалізації LINQ to SQL і WCF.

• Проект DotGNU (в даний час припинений) також надав компілятор C # з відкритим вихідним кодом, майже повну реалізацію інфраструктури спільної мови, включаючи необхідні бібліотеки інфраструктури, як вони зазначені в специфікації ECMA, і підмножина деяких залишилися пропрієтарних класів Microsoft .NET. бібліотеки .NET 2.0 (не документовані або не включені в специфікацію ECMA, але включені в стандартний дистрибутив Microsoft .NET Framework).• Загальна мовна інфраструктура загального ресурсу Microsoft під кодовою назвою «Rotor» забезпечує реалізацію спільного джерела середовища CLR і компілятора C #, ліцензованого тільки для освітніх і дослідницьких цілей, а також підмножина необхідних бібліотек інфраструктури Common Language Infrastructure в специфікації ECMA (вгору в C # 2.0 і підтримується тільки в Windows XP).

C# була обрана основною мовою розробки даного проекту з огляду на усе вищеописане, а саме — на функціонал, який підтримує дана мова, операційні системи, які підтримуються та простота вивчення.

Загалом — функціонал C# найкраще підходить під дану розробку, за рахунок простої реалізації основних принципів ООП, простоти розробки форм та зрозумілого інтерфейсу.

2.2 Вибір СКБД

Microsoft SQL Server - це реляційна система управління базами даних, розроблена Microsoft. Як сервер баз даних, це програмний продукт з основною функцією зберігання та отримання даних, як вимагають інші програмні програми, - який може працювати як на одному комп'ютері, так і на іншому комп'ютері через мережу (включаючи Інтернет). Корпорація Майкрософт продає щонайменше дюжину різних видань Microsoft SQL Server, спрямованих на різну аудиторію та для робочих навантажень, починаючи від невеликих однокомпонентних додатків і закінчуючи великими додатками, орієнтованими на Інтернет, з багатьма одночасними користувачами.

Історія Microsoft SQL Server починається з першого продукту Microsoft SQL Server - SQL Server 1.0, 16-розрядного сервера для операційної системи OS / 2 в 1989 році - і поширюється на поточний день.

- MS SQL Server для OS / 2 розпочався як проект портування Sybase SQL Server на OS / 2 в 1989 році від Sybase, Ashton-Tate та Microsoft.
- SQL Server 4.2 для NT вийшов у 1993 році, що означає вхід до Windows NT.
- SQL Server 6.0 вийшов у 1995 році, що означає закінчення співпраці з Sybase; Sybase продовжуватиме розробляти власний варіант SQL Server, Sybase Adaptive Server Enterprise, незалежно від Microsoft.
- SQL Server 7.0 вийшов у 1998 році, що означає перетворення вихідного коду з C на C ++.
- SQL Server 2005, випущений у 2005 році, закінчує повну переробку старого коду Sybase у код Microsoft.
- SQL Server 2012, випущений в 2012 році, додає стовпчасте сховище в пам’яті aka xVelocity, яке розпочало нову епоху в онлайн-аналітиці
- SQL Server 2017, випущений у 2017 році, додає підтримку Linux для таких платформ Linux: Red Hat Enterprise Linux, SUSE Linux Enterprise Server, Ubuntu & Docker Engine. [4]

Випуск SQL Server Express - це масштабоване безкоштовне видання SQL Server, що включає основний механізм баз даних. Незважаючи на відсутність обмежень щодо кількості баз даних або користувачів, які підтримуються, воно обмежується використанням одного процесора, 1 ГБ пам'яті та 10 ГБ файлів баз даних (4 ГБ файлів баз даних до SQL Server Express 2008 R2). [13] Він призначений як заміна MSDE. Два додаткові видання надають набір функцій, що не в оригіналі Express Edition. Перший - це SQL Server Express з інструментами, до складу якого входить SQL Server Management Studio Basic. SQL Server Express з розширеними послугами додає можливості повнотекстового пошуку та служби звітування. [14]

Зберігання даних - це база даних, яка являє собою сукупність таблиць із набраними стовпцями. SQL Server підтримує різні типи даних, включаючи примітивні типи, такі як Integer, Float, Decimal, Char (включаючи рядки символів), Varchar (рядки символів змінної довжини), двійкові (для неструктурованих BLOB-даних), Text (для текстових даних) та ін. . В округленні плаваючих значень до цілих чисел використовується або Симетричне арифметичне округлення, або Симетричне округлення вниз (виправлення) залежно від аргументів: SELECT Round (2.5, 0) дає 3.

Microsoft SQL Server також дозволяє визначати та використовувати визначені користувачем складені типи (UDT). Це також робить статистику сервера доступною як віртуальні таблиці та подання (так звані Dynamic Management Views або DMV). На додаток до таблиць, база даних може також містити інші об'єкти, включаючи подання, збережені процедури, індекси та обмеження, а також журнал транзакцій. База даних SQL Server може містити максимум 231 об’єкт і може охоплювати декілька файлів на рівні ОС з максимальним розміром файлу 260 байт (1 екзабайт). [9] Дані в базі даних зберігаються у первинних файлах даних із розширенням .mdf. Вторинні файли даних, що ідентифікуються із розширенням .ndf, використовуються для того, щоб дані однієї бази даних могли поширюватися в декількох файлах, а також

необов'язково в декількох файлових системах. Файли журналів ідентифікуються із розширенням .ldf. [9]

Місце для зберігання, виділене базі даних, розділене на сторінки з послідовною нумерацією, кожна розміром 8 КБ. Сторінка є основною одиницею вводу-виводу для операцій SQL Server. Сторінка позначена 96-байтним заголовком, в якому зберігаються метадані про сторінку, включаючи номер сторінки, тип сторінки, вільний простір на сторінці та ідентифікатор об’єкта, який їй належить. Тип сторінки визначає дані, що містяться на сторінці. Ці дані включають: дані, що зберігаються в базі даних, індекс, карту розподілу, яка містить інформацію про те, як сторінки розподіляються по таблицях та індексах; і карта змін, яка містить інформацію про зміни, внесені на інші сторінки з моменту останнього резервного копіювання чи реєстрації, або містить великі типи даних, такі як зображення або текст. Незважаючи на те, що сторінка є основною одиницею операції вводу-виводу, простір насправді управляється з точки зору, який складається з 8 сторінок. Об'єкт бази даних може або охоплювати всі 8 сторінок в екстенті ("однаковий екстент"), або спільно використовувати екстент із ще 7 об'єктами ("змішаний екстент"). Рядок у таблиці бази даних не може охоплювати більше однієї сторінки, тому його розмір обмежений 8 КБ. Однак, якщо дані перевищують 8 КБ і рядок містить дані varchar або varbinary, дані в цих стовпцях переміщуються на нову сторінку (або, можливо, послідовність сторінок, що називається одиницею розподілу) і замінюються вказівником на дані. [23]

Для фізичного зберігання таблиці її рядки поділяються на ряд розділів (з номерами від 1 до n). Розмір розділу визначається користувачем; за замовчуванням всі рядки знаходяться в одному розділі. Таблиця розбита на кілька розділів для розповсюдження бази даних по кластеру комп'ютерів. Рядки в кожному розділі зберігаються або в B-дереві, або в купі. Якщо таблиця має пов’язаний кластерний індекс, що дозволяє швидко отримувати рядки, рядки зберігаються в порядку відповідно до їхніх значень індексу, а індекс дає B-дерево. Дані знаходяться в листовому вузлі листя, а інші вузли зберігають

значення індексу для листкових даних, доступних з відповідних вузлів. Якщо індекс не кластеризований, рядки не сортуються відповідно до ключів індексу. Індексований вигляд має ту саму структуру сховища, що і індексована таблиця. Таблиця без кластерного індексу зберігається в невпорядкованій структурі купи. Однак таблиця може мати некластеризовані індекси, що дозволяють швидко отримувати рядки. У деяких ситуаціях структура купи має переваги щодо продуктивності перед кластерною структурою. І купи, і B-дерева можуть охоплювати кілька одиниць розподілу. [24]

Основним режимом отримання даних з бази даних SQL Server є запит щодо них. Запит виражається за допомогою варіанту SQL, який називається T-SQL, діалект Microsoft SQL Server, який ділиться із Sybase SQL Server через його спадщину. Запит декларативно вказує, що потрібно отримати. Він обробляється процесором запитів, який визначає послідовність кроків, необхідних для отримання запитуваних даних. Послідовність дій, необхідних для виконання запиту, називається планом запитів. Може бути кілька способів обробити один і той же запит. Наприклад, для запиту, який містить оператор join і оператор select, виконання об’єднання в обох таблицях, а потім виконання select у результатах дасть той самий результат, що і вибір із кожної таблиці, а потім виконання об’єднання, але результат у різному виконанні плани. У такому випадку SQL Server вибирає план, який, як очікується, дасть результати в найкоротші терміни. Це називається оптимізацією запитів і виконується самим процесором запитів. [9]

SQL Server включає оптимізатор запитів на основі витрат, який намагається оптимізувати витрати з точки зору ресурсів, необхідних для виконання запиту. Враховуючи запит, тоді оптимізатор запитів розглядає схему бази даних, статистику бази даних та завантаження системи на той момент. Потім він вирішує, яку послідовність отримати для доступу до таблиць, згаданих у запиті, яку послідовність для виконання операцій та який спосіб доступу використовувати для доступу до таблиць. Наприклад, якщо таблиця має пов’язаний індекс, чи слід використовувати індекс, чи ні: якщо

індекс знаходиться у стовпці, який не є унікальним для більшості стовпців (низька „вибірковість”), можливо, не варто використовувати індекс для доступу до даних. Нарешті, він вирішує, виконувати запит одночасно чи ні. Хоча одночасне виконання є більш дорогим з точки зору загального часу процесора, оскільки виконання насправді розділене на різні процесори, може означати, що воно буде виконуватися швидше. Після створення плану запиту для запиту він тимчасово кешований. Для подальших викликів того самого запиту використовується кешований план. Невикористані плани через деякий час відкидаються. [9] [26]

SQL Server також дозволяє визначати збережені процедури. Збережені процедури - це параметризовані запити T-SQL, які зберігаються на самому сервері (а не видаються клієнтською програмою, як це відбувається у загальних запитах). Збережені процедури можуть приймати значення, надіслані клієнтом, як вхідні параметри, а результати повертати як вихідні параметри. Вони можуть викликати визначені функції та інші збережені процедури, включаючи ту саму збережену процедуру (до встановленої кількості разів). До них можна вибірково надати доступ. На відміну від інших запитів, збережені процедури мають пов'язане ім'я, яке використовується під час виконання для вирішення фактичних запитів. Крім того, що код не потрібно надсилати щоразу від клієнта (оскільки до нього можна отримати доступ по імені), це зменшує мережевий трафік і дещо покращує продуктивність. Плани виконання збережених процедур також кешуються за необхідності.

T-SQL (Transact-SQL) - це запатентоване розширення процедурної мови від Microsoft для SQL Server. Він надає інструкції REPL (Read-Eval-Print-Loop), які розширюють стандартний набір команд SQL для маніпуляцій даними (DML) та інструкцій з визначення даних (DDL), включаючи налаштування SQL Server, захист та управління статистикою баз даних.

Він надає ключові слова для операцій, які можна виконувати на SQL Server, включаючи створення та зміну схем бази даних, введення та

редагування даних у базі даних, а також моніторинг та управління самим сервером. Клієнтські програми, які споживають дані або керують сервером, використовуватимуть функціональність SQL Server, надсилаючи запити та оператори T-SQL, які потім обробляються сервером і результати (або помилки) повертаються до клієнтської програми. Для цього він виставляє таблиці лише для читання, з яких можна читати статистику сервера. Функціональність управління забезпечується за допомогою визначених системою збережених процедур, які можна викликати із запитів T-SQL для виконання операції управління. Також можна створити пов'язані сервери за допомогою T-SQL. Пов’язані сервери дозволяють одним запитом обробляти операції, що виконуються на декількох серверах. [28]

Через лекгість у використанні, вбудовану інтеграцію з C# Visual Studio та наявність інтуїтивно зрозумілого графічного інтерфейсу була обрана системою управління баз даних для даного проекту.

2.3 Вибір середовища розробки

Microsoft Visual Studio - це інтегроване середовище розробки (IDE) від Microsoft. Він використовується для розробки комп'ютерних програм, а також веб-сайтів, веб-додатків, веб-служб та мобільних додатків. Visual Studio використовує платформи Microsoft для розробки програмного забезпечення, такі як API Windows, Windows Forms, Foundation Presentation Foundation, Windows Store та Microsoft Silverlight. Він може створювати як власний код, так і керований код.

Visual Studio включає редактор коду, що підтримує IntelliSense (компонент доповнення коду), а також рефакторинг коду. Інтегрований налагоджувач працює як налагоджувач на рівні джерела, так і налагоджувач на рівні машини. Інші вбудовані інструменти включають кодовий профілер, конструктор для побудови програм GUI, веб-дизайнер, дизайнер класів та дизайнер схем бази даних. Він приймає плагіни, які покращують функціональність майже на кожному рівні - включаючи додавання підтримки для систем керування джерелами (наприклад, Subversion і Git) та додавання нових наборів інструментів, таких як редактори та візуальні дизайнери для мов, що задаються доменом або набори інструментів для інших аспектів розробки програмного забезпечення життєвий цикл (як клієнт Azure DevOps: Team Explorer).

Visual Studio підтримує 36 різних мов програмування та дозволяє редактору коду та налагоджувачу підтримувати (в різній мірі) майже будь-яку мову програмування за умови існування послуги, що залежить від мови. Вбудовані мови включають C, C ++, C ++ / CLI, Visual Basic .NET, C #, F #, JavaScript, TypeScript, XML, XSLT, HTML та CSS. Підтримка інших мов, таких як Python, Ruby, Node.js та M серед інших, доступна через плагіни. Java (і J #) підтримувалися в минулому.

Найбільш основне видання Visual Studio, спільноти, доступне безкоштовно. Гасло для видання Visual Studio Community - "Безкоштовне

повнофункціональне IDE для студентів, відкритих джерел та індивідуальних розробників".

На даний момент підтримується версія Visual Studio - 2019 рік.

Visual Studio не підтримує жодної мови програмування, рішення чи інструменту внутрішньо; натомість це дозволяє підключати функціональність, кодовану як VSPackage. Після встановлення функціональність доступна як Сервіс. IDE надає три послуги: SVsSolution, яка надає можливість перераховувати проекти та рішення; SVsUIShell, який забезпечує функцію вікон та інтерфейсу користувача (включаючи вкладки, панелі інструментів та вікна інструментів); та SVsShell, яка займається реєстрацією VSPackages. Крім того, IDE також відповідає за координацію та забезпечення зв'язку між службами. Всі редактори, дизайнери, типи проектів та інші інструменти реалізовані як VSPackages. Visual Studio використовує COM для доступу до VSPackages. SDK Visual Studio також включає в себе керовану рамку пакетів (MPF), яка є набором керованих обгорток навколо COM-інтерфейсів, які дозволяють писати пакети будь-якою мовою, сумісною з CLI. Однак MPF не забезпечує всю функціональність, що піддається інтерфейсам Visual Studio COM. Потім послуги можуть бути використані для створення інших пакетів, які додають функціональність Visual Studio IDE.

Підтримка мов програмування додається за допомогою спеціального VSPackage, який називається Language Service. Мовна служба визначає різні інтерфейси, які може реалізувати реалізація VSPackage, щоб додати підтримку різних функціональних можливостей. Функції, які можна додати таким чином, включають забарвлення синтаксису, завершення оператора, відповідність дужок, підказки інформації про параметри, списки членів та маркери помилок для складання фону. Якщо інтерфейс буде реалізований, функціонал буде доступний для мови. Мовні послуги реалізуються на мовній основі. Реалізації можуть повторно використовувати код з аналізатора або компілятора для мови. Мовні сервіси можуть бути реалізовані як у рідному коді, так і керованому коді. Для нативного коду можуть бути використані або рідні

інтерфейси COM, або Babel Framework (частина Visual Studio SDK). Для керованого коду MPF включає обгортки для написання сервісів керованої мови.

Visual Studio не включає вбудовану підтримку управління джерелами, але вона визначає два альтернативних способи інтеграції систем управління джерелами з IDE. VSPackage управління джерелом може забезпечити власний індивідуальний інтерфейс користувача. Навпаки, плагін управління джерелом за допомогою MSSCCI (Microsoft Source Code Control Interface) надає набір функцій, які використовуються для реалізації різних функцій управління джерелом, зі стандартним інтерфейсом користувача Visual Studio. MSSCCI вперше використовувався для інтеграції Visual SourceSafe з Visual Studio 6.0, але згодом був відкритий через SDK Visual Studio. Visual Studio .NET 2002 використовував MSSCCI 1.1, а Visual Studio .NET 2003 використовували MSSCCI 1.2. Visual Studio 2005, 2008 та 2010 використовує MSSCCI версії 1.3, яка додає підтримку перейменування та видалення розповсюдження, а також асинхронного відкриття.

Visual Studio підтримує запуск декількох екземплярів середовища (кожен зі своїм набором VSPackages). Екземпляри використовують різні вулики реєстру (див. Визначення MSDN терміна "вулик реєстру" у сенсі, що використовується тут) для зберігання конфігураційного стану та диференціюються їх AppId (ідентифікатор програми). Екземпляри запускаються специфічним для AppId .exe, який вибирає AppId, встановлює кореневий вулик та запускає IDE. VSP-пакети, зареєстровані для одного AppId, інтегруються з іншими VSP-пакетами для цього AppId. Різні випуски продуктів Visual Studio створені за допомогою різних додатків. Продукти випуску Visual Studio Express встановлюються разом із власними AppIds, але продукти Standard, Professional та Team Suite мають однаковий AppId. Отже, видання Express можна встановити поряд з іншими виданнями, на відміну від інших видань, які оновлюють ту саму установку. Професійне видання включає в себе набір VSPackages у стандартному виданні, а командний набір

включає суперсет VSPackages в обох інших виданнях. Система AppId використовується Visual Studio Shell в Visual Studio 2008.

Дане середовище розробки було обрано по декількох критеріях:

- Підтримка мови C#
- Інтуїтивно зрозумілий інтерфейс
- Можливості графічного редактору, підходящі до задач

Так як усі критерії задовільнено — середовищем розробки було обрано саме Visual Studio 2019.

РОЗДІЛ 3
ПРОЕКТУВАННЯ І РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Моделювання бази даних

У обчислювальній техніці база даних — це організована сукупність даних, які зберігаються та доступні в електронному вигляді. Невеликі бази даних можна зберігати у файловій системі, тоді як великі бази даних розміщуються в комп’ютерних кластерах або хмарних сховищах. Проектування баз даних охоплює формальні методи та практичні міркування, включаючи моделювання даних, ефективне представлення та зберігання даних, мови запитів, безпеку та конфіденційність конфіденційних даних, а також проблеми розподілених обчислень, включаючи підтримку одночасного доступу та відмовостійкості.

Система управління базами даних (СУБД) — це програмне забезпечення, яке взаємодіє з кінцевими користувачами, додатками та самою базою даних для збору й аналізу даних. Програмне забезпечення СУБД додатково охоплює основні засоби, надані для адміністрування бази даних. Загальну суму бази даних, СУБД і пов’язаних програм можна назвати системою баз даних. Часто термін "база даних" також використовується вільно для позначення будь-якої з СУБД, системи баз даних або програми, пов'язаної з базою даних.

Комп’ютерні науковці можуть класифікувати системи керування базами даних відповідно до моделей баз даних, які вони підтримують. Реляційні бази даних стали домінуючими в 1980-х роках. Ці дані моделюють у вигляді рядків і стовпців у серії таблиць, і переважна більшість використовує SQL для запису та запиту даних. У 2000-х роках стали популярними нереляційні бази даних, які спільно називали NoSQL, оскільки вони

Модель бази даних — це тип моделі даних, що визначає логічну структуру бази даних. Вона в основному визначає, яким чином дані можна зберігати, організовувати та маніпулювати. Найпопулярнішим прикладом моделі бази даних є реляційна модель, яка використовує формат на основі таблиці.

Дана система управління базою даних може надавати одну або кілька моделей. Оптимальна структура залежить від природної організації даних програми та від вимог до програми, які включають швидкість транзакції (швидкість), надійність, ремонтопридатність, масштабованість та вартість. Більшість систем керування базами даних побудовано навколо однієї конкретної моделі даних, хоча продукти можуть запропонувати підтримку кількох моделей.

Різні фізичні моделі даних можуть реалізувати будь-яку задану логічну модель. Більшість програмного забезпечення баз даних запропонує користувачеві певний рівень контролю при налаштуванні фізичної реалізації, оскільки вибір, який робиться, має значний вплив на продуктивність.

Модель — це не просто спосіб структурування даних: вона також визначає набір операцій, які можна виконати над даними.[1] Реляційна модель, наприклад, визначає такі операції, як вибір (проект) і приєднання. Хоча ці операції можуть не бути явними в певній мові запитів, вони забезпечують основу, на якій будується мова запитів.

Реляційна модель була введена Е.Ф. Коддом у 1970 році[2] як спосіб зробити системи управління базами даних більш незалежними від будь-якої конкретної програми. Це математична модель, визначена в термінах логіки предикатів і теорії множин, і її реалізації використовувалися мейнфреймами, системами середнього рівня та мікрокомп'ютерами.

Продукти, які зазвичай називають реляційними базами даних, насправді реалізують модель, яка є лише наближенням до математичної моделі, визначеної Коддом. Три ключові терміни широко використовуються в моделях реляційних баз даних: відносини, атрибути та домени. Відношення –

це таблиця зі стовпцями та рядками. Названі стовпці відношення називаються атрибутами, а домен — це набір значень, які атрибутам дозволено приймати.

Основною структурою даних реляційної моделі є таблиця, де інформація про певну сутність (скажімо, працівника) представлена в рядках (також званих кортежами) і стовпцях. Таким чином, «відношення» в «реляційній базі даних» відноситься до різних таблиць у базі даних; відношення — це набір кортежів. Стовпці перераховують різні атрибути сутності (наприклад, ім'я працівника, адресу чи номер телефону), а рядок — це фактичний екземпляр сутності (конкретного працівника), який представлений відношенням. В результаті кожен кортеж таблиці співробітників представляє різні атрибути окремого співробітника.

Усі відносини (і, таким чином, таблиці) у реляційній базі даних повинні відповідати деяким основним правилам, щоб кваліфікуватися як відносини. По-перше, порядок стовпців у таблиці не має значення. По-друге, в таблиці не може бути однакових кортежів або рядків. І по-третє, кожен кортеж міститиме одне значення для кожного зі своїх атрибутів.

Реляційна база даних містить кілька таблиць, кожна з яких подібна до тієї в моделі бази даних «плоска». Однією з сильних сторін реляційної моделі є те, що, в принципі, будь-яке значення, що зустрічається в двох різних записах (що належать до однієї таблиці або до різних таблиць), передбачає зв'язок між цими двома записами. Тим не менш, щоб застосувати явні обмеження цілісності, зв'язки між записами в таблицях також можуть бути визначені явно, шляхом ідентифікації або неідентифікації батьківсько-дочірніх зв'язків, що характеризуються призначенням потужності (1:1, (0)1:M, M:M). Таблиці також можуть мати призначений один атрибут або набір атрибутів, які можуть діяти як «ключ», який можна використовувати для унікальної ідентифікації кожного кортежу в таблиці.

Ключ, який можна використовувати для однозначної ідентифікації рядка в таблиці, називається первинним ключем. Ключі зазвичай використовуються для об'єднання або об'єднання даних з двох або більше таблиць. Наприклад,

таблиця Employee може містити стовпець з назвою Location, який містить значення, яке відповідає ключу таблиці Location. Ключі також мають вирішальне значення при створенні індексів, які полегшують швидке отримання даних з великих таблиць. Будь-який стовпець може бути ключем, або кілька стовпців можна згрупувати разом у складений ключ. Необов'язково визначати всі ключі заздалегідь; стовпець можна використовувати як ключ, навіть якщо він спочатку не передбачався.

Ключ, який має зовнішнє, реальне значення (наприклад, ім'я людини, ISBN книги або серійний номер автомобіля), іноді називають «природним» ключем. Якщо жоден природний ключ не підходить (згадайте багатьох людей на ім'я Браун), можна призначити довільний або сурогатний ключ (наприклад, надавши співробітникам ідентифікаційні номери). На практиці більшість баз даних мають як згенеровані, так і природні ключі, оскільки згенеровані ключі можуть використовуватися внутрішньо для створення зв'язків між рядками, які не можуть розірватися, тоді як природні ключі можна використовувати менш надійно для пошуку та для інтеграції з іншими базами даних. (Наприклад, записи в двох незалежно розроблених базах даних можуть бути зіставлені за номером соціального страхування, за винятком випадків, коли номери соціального страхування неправильні, відсутні або змінені.)

Найпоширенішою мовою запитів, яка використовується з реляційною моделлю, є мова структурованих запитів (SQL).

В додатку А на рисунку 1 зображена модель бази даних.

3.2 Діаграма класів системи

У програмній інженерії діаграма класів в Уніфікованій мові моделювання (UML) - це тип статичної структурної діаграми, що описує структуру системи, показуючи класи системи, їх атрибути, операції (або методи) та взаємозв'язки між об'єктами.

Діаграма класів є основним будівельним елементом об’єктно-орієнтованого моделювання. Він використовується для загального концептуального моделювання структури програми та для детального моделювання переведення моделей у програмовий код. Діаграми класів також можуть бути використані для моделювання даних. Класи на діаграмі класів представляють як основні елементи, взаємодії в програмі, так і класи, що програмуються.

На схемі класи представлені вікнами, які містять три відділення:

- У верхньому відділенні міститься назва класу. Надруковано жирним шрифтом і відцентровано, а перша літера написана великими літерами.
- Середній відсік містить атрибути класу. Вони вирівняні за лівим краєм, а перша буква мала.
- У нижньому відділенні містяться операції, які може виконувати клас. Вони також вирівняні за лівим краєм, а перша буква - мала.

При проектуванні системи ряд класів ідентифікується та згруповується у схему класів, яка допомагає визначити статичні відносини між ними. При детальному моделюванні класи концептуального проекту часто поділяються на ряд підкласів.

Залежність - це семантичний зв’язок між залежними та незалежними елементами моделі. Він існує між двома елементами, якщо зміни у визначенні одного елемента (сервера або цілі) можуть спричинити зміни для іншого (клієнта або джерела). Ця асоціація є односпрямованою. Залежність відображається у вигляді штрихової лінії з відкритою стрілкою, яка вказує від клієнта до постачальника.

Для подальшого опису поведінки систем ці діаграми класів можуть бути доповнені діаграмою стану або машиною стану UML.

Асоціація представляє родину посилань. Двійкова асоціація (з двома кінцями) зазвичай представляється у вигляді рядка. Асоціація може пов’язувати будь-яку кількість класів. Асоціація з трьома ланками називається потрійною асоціацією. Асоціацію можна назвати, а кінці асоціації можна прикрасити іменами ролей, показниками власності, кратністю, видимістю та іншими властивостями.

Існує чотири різні типи асоціацій: двонаправлена, односпрямована, агрегаційна (включає агрегацію композиції) та рефлексивна. Двонаправлені та односпрямовані асоціації є найбільш поширеними.

Наприклад, клас польоту асоціюється з класом літака двонаправлено. Асоціація представляє статичне відношення, яке ділиться між об'єктами двох класів.

Агрегація є варіантом взаємозв'язку "має"; агрегація є більш конкретною, ніж асоціація. Це асоціація, яка представляє частково цілі або часткові стосунки. Як показано на зображенні, професор "має" клас для викладання. Як тип асоціації, агрегація може бути названа та мати ті самі прикраси, що і асоціація. Однак агрегація не може включати більше двох класів; це має бути бінарна асоціація. Крім того, навряд чи існує різниця між агрегаціями та асоціаціями під час реалізації, і діаграма може взагалі пропустити відносини агрегування. [7]

Агрегація може відбуватися, коли клас є колекцією або контейнером інших класів, але вміщені класи не мають сильної залежності життєвого циклу від контейнера. Вміст контейнера все ще існує, коли контейнер знищений.

В UML він графічно представлений у вигляді порожнистої форми ромба на вміщуючому класі одним рядком, що зв’язує його із вміщеним класом. Сукупність - це семантично розширений об’єкт, який у багатьох операціях

трактується як одиниця, хоча фізично він складається з декількох менших об’єктів.

Приклад: Бібліотека та студенти. Тут студент може існувати без бібліотеки, зв'язок між студентом і бібліотекою є агрегацією.

Це вказує на те, що один із двох пов’язаних класів (підклас) вважається спеціалізованою формою іншого (супер тип), а суперклас - узагальненням підкласу. На практиці це означає, що будь-який екземпляр підтипу є також екземпляром суперкласу. Зразкове дерево узагальнень цієї форми зустрічається в біологічній класифікації: людина - це підклас маймуни, який є підкласом ссавців тощо. Зв’язок найлегше зрозуміти за допомогою фрази „А - це В” (людина - це ссавець, ссавець - тварина).

Графічне представлення UML узагальнення - це форма порожнистого трикутника на кінці суперкласу рядка (або дерева рядків), що зв’язує його з одним або кількома підтипами.

Відносини узагальнення також відомі як спадщина або відносини "є".

Суперклас (базовий клас) у відносинах узагальнення також відомий як "батьківський", суперклас, базовий клас або базовий тип.

Підтип у відносинах спеціалізації також відомий як "дочірній", підклас, похідний клас, похідний тип, клас успадкування або тип успадкування.

Зверніть увагу, що ці стосунки нічим не схожі на біологічні стосунки батьків та дітей: використання цих термінів надзвичайно поширене, але може ввести в оману.

А - це тип В

Наприклад, "дуб - це тип дерева", "автомобіль - це тип транспортного засобу"

Узагальнення може бути показано лише на діаграмах класів та на діаграмах використання.

При моделюванні UML взаємозв'язок реалізації - це взаємозв'язок між двома елементами моделі, в яких один елемент моделі (клієнт) реалізує

(реалізує або виконує) поведінку, яку вказує інший елемент моделі (постачальник).

Графічне представлення UML реалізації - це порожниста форма трикутника на кінці інтерфейсу штрихової лінії (або дерева рядків), яка з'єднує її з одним або кількома реалізаторами. Проста головка стрілки використовується на кінці інтерфейсу штрихової лінії, що з'єднує її з користувачами. У діаграмах компонентів використовується графічна умова «м'яч і сокет» (реалізатори виставляють кульку або льодяник, тоді як користувачі показують сокет). Реалізації можна показати лише на діаграмах класів або компонентів. Реалізація - це взаємозв'язок між класами, інтерфейсами, компонентами та пакетами, що з'єднує елемент клієнта з елементом постачальника. Зв'язок реалізації між класами / компонентами та інтерфейсами показує, що клас / компонент реалізує операції, пропоновані інтерфейсом.

Залежність - це слабша форма зв'язку, яка вказує на те, що один клас залежить від іншого, оскільки він використовує його в певний момент часу. Один клас залежить від іншого, якщо незалежний клас є змінною параметра або локальною змінною методу залежного класу. Це відрізняється від асоціації, де атрибут залежного класу є екземпляром незалежного класу. Іноді відносини між двома класами дуже слабкі. Вони взагалі не реалізовані зі змінними-членами. Швидше вони можуть бути реалізовані як аргументи функції-члена.

інший. Ці відносини зазвичай описуються як "А має В" (у матері-кота є кошенята, у кошенят - мати-кішка).

Представлення UML асоціації - це лінія, що з'єднує два пов'язані класи. На кожному кінці рядка є додаткові позначення. Наприклад, ми можемо вказати, використовуючи наконечник стрілки, що загострений кінець видно з хвоста стрілки. Ми можемо вказати власність шляхом розміщення кульки, ролі, яку відіграють елементи цього кінця, вказавши ім'я ролі та множинність

екземплярів цієї сутності (діапазон кількості об'єктів, які беруть участь в асоціації з точки зору іншого кінця).

Класи сутності моделюють довгоживучу інформацію, якою обробляє система, а іноді і поведінку, пов'язану з цією інформацією. Їх не слід ідентифікувати як таблиці баз даних чи інших сховищ даних.

Вони намальовані як кола з короткою лінією, прикріпленою до нижньої частини кола. Як варіант, їх можна намалювати як звичайні класи із позначенням стереотипу «сутність» над назвою класу.

В додатку Б на рисунку 2 зображена діаграма класів програмного продукту.

3.3 Діаграма варіантів використання системи

Діаграма використання найпростіша - це представлення взаємодії користувача із системою, яка показує взаємозв'язок між користувачем та різними випадками використання, в яких користувач бере участь. Діаграма випадків використання може ідентифікувати різні типи користувачів системи та різні випадки використання, і часто вона супроводжується також іншими типами діаграм. Варіанти використання представлені кругами або еліпсами.

Незважаючи на те, що сам випадок використання може детально вивчити кожну можливість, діаграма прикладів використання може допомогти забезпечити огляд системи на більш високому рівні. Раніше вже було сказано, що "схеми використання - це принципи вашої системи".

Через їх спрощений характер, схеми використання можуть бути хорошим інструментом комунікації для зацікавлених сторін. Креслення намагаються імітувати реальний світ і дають зацікавленій стороні уявлення про те, як буде розроблена система. Сіау та Лі провели дослідження, щоб визначити, чи взагалі існувала дійсна ситуація для схем використання або вони були непотрібними. Було виявлено, що діаграми випадків використання передають намір системи більш спрощеним чином зацікавленим сторонам і що вони "інтерпретуються більш повно, ніж діаграми класів".

Метою діаграми використання є відображення динамічного аспекту системи. Додаткові схеми та документація можуть бути використані для забезпечення повного функціонального та технічного уявлення про систему. Вони забезпечують спрощене та графічне представлення того, що система насправді повинна робити.

Елементи:

- рамки системи (англ. system border) - прямоугольник із назвою у верхніх частинах та еліпсами (прецедентами) всередині. Часто може бути опущено без корисної інформації про полезну інформацію,

- актор (англ. актор) - стилізований людський персонаж, обозначаючий набір ролей користувача (розуміється в широкому змісті: людина, внешняя сутність, клас, інша система), взаємодіючого з деякою сутністю (системною, підсистемою, класом). Актори не можуть бути пов'язані між собою з іншим (за вимкнення відносин щодо обробки / дослідження),
- прецедент - еліпс із надписом, що обозначає виконувану систематичну дію (може включати можливі варіанти), що призводить до спостерігаємих акторами результатів. Надпис може бути ім'ям або описом (з точки зору актора) того, "що" робить система (а не "як"). Ім прецедента зв’язано з неперервним (атомарним) сценарієм - конкретною послідовністю дій, ілюстріруючою повіддю. Під час сценарію актори обмінюються із систематичними повідомленнями. Сценарій може бути приведений на діаграмі прецедентів у відео UML-коментарі. З одним прецедентом може бути пов'язано кілька різних сценаріїв

В додатку Б на рисунку 3 зображено діаграму варіантів використання, яка описує можливі дії користувача в системі.

3.4 Діаграма компонентів

В уніфікованій мові моделювання (UML) діаграма компонентів показує, як компоненти з’єднані разом, щоб утворити більші компоненти або програмні системи. Вони використовуються для ілюстрації структури як завгодно складних систем.

Діаграма компонентів дозволяє перевірити, що необхідна функціональність системи є прийнятною. Ці діаграми також використовуються як інструмент комунікації між розробником і зацікавленими сторонами системи. Програмісти та розробники використовують діаграми, щоб формалізувати дорожню карту впровадження, що дозволить краще приймати рішення щодо призначення завдання або необхідного покращення навичок. Системні адміністратори можуть використовувати діаграми компонентів для планування наперед, використовуючи уявлення про логічні програмні компоненти та їх взаємозв’язки в системі.[1]

Діаграма компонентів розширює інформацію, наведену в елементі позначення компонента. Один із способів ілюстрації наданих і необхідних інтерфейсів зазначеним компонентом — це прямокутний відсік, приєднаний до компонентного елемента.[2] Іншим прийнятним способом представлення інтерфейсів є використання графічної конвенції «м’яч і гнізда». Надана залежність від компонента до інтерфейсу ілюструється суцільною лінією до компонента, що використовує інтерфейс із «льодяника» або кульки, позначеного назвою інтерфейсу. Необхідна залежність використання від компонента до інтерфейсу ілюструється півколом або сокетом, позначеним назвою інтерфейсу, прикріпленим суцільною лінією до компонента, якому потрібен цей інтерфейс. Успадковані інтерфейси можуть бути показані за допомогою льодяника, перед міткою імені символом каретки. Щоб проілюструвати залежності між ними, використовуйте суцільну лінію із звичайним наконечником стрілки, що з’єднує розетку з льодяником.[3]

В додатку Б на рисунку 4 зображено діаграму компонентів.

3.5 Діаграма розгортання системи

Діаграма розгортання в уніфікованій мові моделювання моделює фізичне розгортання артефактів на вузлах.[1] Для опису веб-сайту, наприклад, діаграма розгортання показує, які апаратні компоненти («вузли») існують (наприклад, веб-сервер, сервер додатків і сервер баз даних), які програмні компоненти («артефакти») працюють на кожен вузол (наприклад, веб-додаток, база даних), а також спосіб з'єднання різних частин (наприклад, JDBC, REST, RMI).

Вузли відображаються у вигляді квадратів, а артефакти, виділені кожному вузлу, відображаються у вигляді прямокутників у межах. Вузли можуть мати підвузли, які виглядають як вкладені блоки. Один вузол на схемі розгортання може концептуально представляти кілька фізичних вузлів, наприклад кластер серверів баз даних.

Існує два типи вузлів:

Вузол пристрою

Вузол середовища виконання

Вузли пристроїв — це фізичні обчислювальні ресурси з обробною пам’яттю та службами для виконання програмного забезпечення, наприклад типових комп’ютерів або мобільних телефонів. Вузол середовища виконання (EEN) — це програмний обчислювальний ресурс, який працює всередині зовнішнього вузла і який сам надає послугу для розміщення та виконання інших виконуваних програмних елементів.

В додатку Б на рисунку 5 зображено діаграму розгортання системи.

3.6 Розробка графічного інтерфейсу

В складі розробленої системи є 11 віконних форм, а саме:

- форма авторизації (Додаток В рис. 6);
- головне меню (Додаток В рис. 7, рис. 8, рис. 9);
- форма постачальників (Додаток В рис. 10);
- форма товарів (Додаток В рис. 11);
- форма складів (Додаток В рис. 12);
- форма накладних (Додаток В рис. 13);
- форма прийому товару (Додаток В рис. 14);
- форма товарів за рік (Додаток В рис. 15);
- форма поставок постачальника (Додаток В рис. 16);
- форма переліку товарів на складах (Додаток В рис. 17);
- форма товарів за місяць (Додаток В рис. 18).

3.7 Тестування розробленої системи

Для тестування розробленого програмного продукту було обрано методику чорного ящика. Тестування чорного ящика — це метод тестування програмного забезпечення, який перевіряє функціональність програми, не заглядаючи в її внутрішні структури чи роботи. Цей метод тестування можна застосувати практично на кожному рівні тестування програмного забезпечення: одиничному, інтеграційному, системному та приймальному. Його іноді називають тестуванням на основі специфікації.

Спеціальні знання коду програми, внутрішньої структури та знання програмування загалом не потрібні. Тестер знає, що програмне забезпечення має робити, але не знає, як воно це робить. Наприклад, тестувальник знає, що конкретні вхідні дані повертають певний незмінний вихід, але не знає, як програмне забезпечення виробляє вихідні дані в першу чергу.

Таблиця 1 — Тестування додатку

№	Тест-кейс	Очікуваний результат	Отриманий результат
1	Невірні данні при авторизації	При введенні невірних даних система повідомляє користувача про те, що такого користувача не існує, або дані введені невірно	При введенні невірних даних система повідомляє користувача про те, що такого користувача не існує, або дані введені невірно
2	Пусті поля при авторизації	При спробі авторизації з пустими полями система повідомляє користувача про те,	При спробі авторизації з пустими полями система повідомляє користувача про те,

		що ці поля необхідно заповнити.	що ці поля необхідно заповнити.
3	Неспівпадаючі паролі при реєстрації	При введенні неспівпадаючих паролів система повідомляє користувача про те, що паролі не співпадають.	При введенні неспівпадаючих паролів система повідомляє користувача про те, що паролі не співпадають.
4	Пусті поля при реєстрації	При спробі реєстрації з пустими полями система повідомляє користувача про те, що ці поля необхідно заповнити.	При спробі реєстрації з пустими полями система повідомляє користувача про те, що ці поля необхідно заповнити.
5	Створення звіту з пустими полями	При спробі створення звіту з пустими полями система повідомляє користувача про те, що ці поля необхідно заповнити.	При спробі створення звіту з пустими полями система повідомляє користувача про те, що ці поля необхідно заповнити.
6	Створення пустого звіту	При спробі створення звіту з пустими полями система повідомляє користувача про те, що ці поля необхідно заповнити.	При спробі створення звіту з пустими полями система повідомляє користувача про те, що ці поля необхідно заповнити.

7	Спроба додати данні до бази даних з пустими полями	При спробі користувача додати данні до бази даних з пустими полями система повідомляє користувача, що подібні дії не можуть бути виконані.	При спробі користувача додати данні до бази даних з пустими полями система повідомляє користувача, що подібні дії не можуть бути виконані.

Результати тестування показують, що система повністю функціональна і готова до використання в реальних умовах.

3.8 Аналіз результатів і новизни системи

Для аналізу результатів і новизни системи можна провести порівняльний аналіз власної реалізації з відомими конкурентами.

Таблиця 2 — Порівняльна характеристика рішень

№	Критерій оцінки	Системи			
		1С	УкрСкад	Storecalc	Власна
1	Простота встановлення системи	-	-	-	+
2	Інтуїтивна зрозумілість графічного інтерфейсу користувача	-	-	+	+
3	Варіативність створення виробничих задач	+	-	-	+
4	Наявність ролей користувачів для обмеження доступу певних груп до закритої для них інформації	+	-	-	+

5	Збереження звітності	+	-	-	+
6	Зручний формат зберігання інформації	-	-	-	+

Як видно з порівняльної таблиці, власне рішення вирішує велику кількість основних проблем, які заважають нормальній роботі спеціаліста в обраній тематиці, що і можна вважати водночас і новизною системи і результатом.

3.9. Розрахунок очікуваного техніко-економічного ефекту від впровадження розробки.

Поточні витрати на експлуатацію.

Балансова вартість ПК розраховується за такою формулою:

Ц (РС) = Цр * (1+ k) = 22 000 * (1+0.3) = 28 600 грн

ЦР – ринкова вартість ПК, орієнтовно складає 22000 грн., k – коефіцієнт, що враховує витрати на установку і налагодження ПК.

Амортизаційні відрахування використання ПК обчислюються за такою формулою:

Z(A) = Ц (РС) / На = 28 600 / 5 = 5720 ₴

На – норма амортизаційних відрахувань

Витрати на електроенергію:

Z (E) = P(PC) * T1(PC) * C(e) * A = 0.040 * 388.4 * 1.9 * 0.9 = 26.56 грн

P(PC) – потужність ПК

A – коефіцієнт інтенсивного використання

Z(R) – витрати на поточний ремонт і технічне обслуговування ПК визначаються як 6% від балансової вартості ПК, ЦПК:

Z(R) = 28600 * 0.06 = 1716 грн

Z(M) – непрямі витрати, пов'язані з експлуатацією ПК, визначаються як 5% від балансової вартості ПК ЦПК:

Z(M) = 28600 * 0.05 = 1430 грн

Таким чином поточні витрати на експлуатацію визначаються:

V1' = Z(O) + Z(A) + Z(E) + Z(R) + Z(M) = 3360 + 5720 + 26.56 + 1716 + 1430 = 12252 грн

Загальні витрати на розробку програмного забезпечення становлять:

V = V1' + V' = 250000 + 12252 = 262252 грн

Оскільки потреб у закупівлі ПК немає, то ці витрати дорівнюють 0.

V2=0

Витрати на навчання персоналу.

В середньому навчання персоналу триватиме 2 дні, тому:

V4 = 3200 грн

Загальна вартість розробки і впровадження.

Загальна вартість розробки і впровадження системи V∑ , вираховується за:

V∑ = V1 + V2 + V3 + V4 = 2252 + 0 + 0 + 3200 = 5452 грн.

Оскільки норма амортизаційних втрат для комп'ютерних систем НА = 5, то для обрахування річного економічного ефекту слід брати до розгляду величину

V(Y) = 5452 / 5 = 1090.4грн

Річний прибуток P(Y) складатиме 13800 грн на рік. Коефіцієнт економічної ефективності тоді: K(E) = P(Y)/V(Y) = 1.6

Тоді термін окупності розробки визначається за формулою:

T(O) = 1/K(E) = 0.625

Тобто термін окупності складатиме приблизно 6 місяців.

Річний прибуток P(Y) складатиме 20 000 ₴ на рік. Коефіцієнт економічної ефективності тоді: K(E) = P(Y)/V(Y) = 4.3

Тоді термін окупності розробки визначається за формулою:

T(O) = 1/K(E) = 0.23

Тобто термін окупності складатиме приблизно 2.5 місяця.

ВИСНОВКИ

В даній дипломній роботі досліджено роботу відділу постачання компанії «Спецпідземінжбуд».

Даний проект розроблявся для обліку та контролю кількості замовленого товару на складі ТОВ «Спецпідземіндбул», який виконано у середовищі Microsoft Visual Studio 2013 при використанні Visual C # та СУБД MS SQL Server 2016. Дана підсистема повинна значно полегшити роботу завідувача складом та начальника відділу постачання при проведенні обліку і контролю товару та обладнання, що надійшли на склад, а саме зручний спосіб внесення даних, редагування і вилучення даних та виконання пошуку та фільтрації потрібних записів за допомогою запитів.

Виконання даної роботи покращило навички та вміння в розробці та проектуванні іноформаційно-облікових систем, що здатні полегшити та значною мірою атоматизувати процеси роботи на складських приміщеннях.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Piccoli, Gabriele; Pigni, Federico (July 2018). Information systems for managers: with cases (Edition 4.0 ed.). Prospect Press. p. 28. ISBN 978-1-943153-50-3. Retrieved 25 November 2018.
2. O'Hara, Margaret; Watson, Richard; Cavan, Bruce (1999). "Managing the three levels of change". Information Systems Management. 16 (3): 64. doi:10.1201/1078/43197.16.3.19990601/31317.9. Retrieved 25 November 2018.
3. "Information Systems". 2020-11-12.
4. "information system". BusinessDictionary.com.
5. "Information Systems". Principia Cybernetica Web.
6. Vladimir Zwass (2016-02-10). "Information system". Britannica.
7. D'Atri A., De Marco M., Casalino N. (2008). "Interdisciplinary Aspects of Information Systems Studies", Physica-Verlag, Springer, Germany, pp. 1–416, doi:10.1007/978-3-7908-2010-2 ISBN 978-3-7908-2009-6
8. "Information Technology vs Information Systems: What's The Difference?". CityU of Seattle. 2020-01-16. Retrieved 2021-11-13.
9. Jessup, Leonard M.; Joseph S. Valacich (2008). Information Systems Today (3rd ed.). Pearson Publishing. Glossary p. 416
10. "What is Information Systems or Information Services (IS)?". Definition from Techopedia. Retrieved 6 March 2021.
11. "What is IS (information system or information services)?". WhatIs.com. Retrieved 6 March 2021.
12. "Information Services". Directory. Australian Government. 2 June 2017. Retrieved 6 March 2021.
13. "Information Services". Ramsey County. 12 September 2015. Retrieved 6 March 2021.

14. Bulgacs, Simon (2013). "The first phase of creating a standardised international innovative technological implementation framework/Software application". International Journal of Business and Systems Research. 7 (3): 250. doi:10.1504/IJBSR.2013.055312. Retrieved 2015-11-02.
15. "SEI Report, "Glossary"". Archived from the original on September 3, 2007. Retrieved 2013-04-02.
16. Kroenke, D M. (2008). Experiencing MIS. Prentice-Hall, Upper Saddle River, NJ
17. O'Brien, J A. (2003). Introduction to information systems: essentials for the e-business enterprise. McGraw-Hill, Boston, MA
18. Alter, S. (2003) "18 Reasons Why IT-Reliant Work Systems Should Replace 'The IT Artifact' as the Core Subject Matter of the IS Field," Communications of the Association for Information Systems, 12(23), Oct., pp. 365–394, http://aisel.aisnet.org/cais/vol12/iss1/23/
19. Alter, S (2013). "Work System Theory: Overview of Core Concepts, Extensions, and Challenges for the Future". Journal of the Association for Information Systems. 14 (2): 72–121. doi:10.17705/1jais.00323.
20. Alter, S. (2006) The Work System Method: Connecting People, Processes, and IT for Business Results. Works System Press, CA
21. Bacon, C. James; Fitzgerald, Brian (2001-04-01). "A systemic framework for the field of information systems". ACM SIGMIS Database: The DATABASE for Advances in Information Systems. 32 (2): 46–67. doi:10.1145/506732.506738. ISSN 0095-0033. S2CID 15687595.
22. Beynon-Davies P. (2009). Business Information Systems. Palgrave, Basingstoke
23. Marc S. Silver, M. Lynne Markus, Cynthia Mathis Beath (Sep 1995). "The Information Technology Interactive Model: A Foundation for the MBA Core Course". MIS Quarterly: 361–390.

24. The Joint Task Force for Computing Curricula 2005. Computing Curricula 2005: The Overview Report (pdf) Archived 2014-10-21 at the Wayback Machine
25. Rockart et al. (1996) Eight imperatives for the new IT organization Sloan Management review.
26. Kroenke, D. M. (2015). MIS Essentials. Pearson Education
27. Laudon, K.C. and Laudon, J.P. Management Information Systems, Macmillan, 1988.
28. Rainer, R. Kelly Jr, and Casey G. Cegielski. Introduction to Information System: Support and Transforming Business Fourth Edition. New Jersey: John Wiley and Sons, Inc., 2012. Print.
29. Neumann, Gustaf; Sobernig, Stefan; Aram, Michael (February 2014). "Evolutionary Business Information Systems". Business and Information Systems Engineering. 6 (1): 33–36. doi:10.1007/s12599-013-0305-1. S2CID 15979292.
30. Aram, Michael; Neumann, Gustaf (2015-07-01). "Multilayered analysis of co-development of business information systems" (PDF). Journal of Internet Services and Applications. 6 (1). doi:10.1186/s13174-015-0030-8. S2CID 16502371.
31. Using MIS. Kroenke. 2009. ISBN 978-0-13-713029-0.
32. Börje Langefors (1973). Theoretical Analysis of Information Systems. Auerbach. ISBN 978-0-87769-151-8.
33. Computer Studies. Frederick Nyawaya. 2008. ISBN 978-9966-781-24-6.
34. "Computer and Logic Essentials – Units of study – Swinburne University of Technology – Melbourne, Australia".
35. "Building IT Systems – RMIT University".
36. "Systems Development – Units of study – Swinburne University of Technology – Melbourne, Australia".
37. Kelly, Sue; Gibson, Nicola; Holland, Christopher; Light, Ben (July 1999). "Focus Issue on Legacy Information Systems and Business Process

Engineering: a Business Perspective of Legacy Information Systems". Communications of the AIS. 2 (7): 1–27.

38. Archibald, J.A. (May 1975). "Computer Science education for majors of other disciplines". AFIPS Joint Computer Conferences: 903–906. Computer science spreads out over several related disciplines, and shares with these disciplines certain sub-disciplines that traditionally have been located exclusively in the more conventional disciplines
39. Denning, Peter (July 1999). "COMPUTER SCIENCE: THE DISCIPLINE". Encyclopaedia of Computer Science (2000 Edition). The Domain of Computer Science: Even though computer science addresses both human-made and natural information processes, the main effort in the discipline has been directed toward human-made processes, especially information processing systems and machines
40. Coy, Wolfgang (June 2004). "Between the disciplines". ACM SIGCSE Bulletin. 36 (2): 7–10. doi:10.1145/1024338.1024340. ISSN 0097-8418. S2CID 10389644. Computer science may be in the core of these processes. The actual question is not to ignore disciplinary boundaries with its methodological differences but to open the disciplines for collaborative work. We must learn to build bridges, not to start in the gap between disciplines
41. Hoganson, Ken (December 2001). "Alternative curriculum models for integrating computer science and information systems analysis, recommendations, pitfalls, opportunities, accreditations, and trends". Journal of Computing Sciences in Colleges. 17 (2): 313–325. ISSN 1937-4771. ... Information Systems grew out of the need to bridge the gap between business management and computer science ...
42. Davis, Timothy; Geist, Robert; Matzko, Sarah; Westall, James (March 2004). τ΄εχνη: A First Step. Technical Symposium on Computer Science Education. pp. 125–129. ISBN 978-1-58113-798-9. In 1999, Clemson University established a (graduate) degree program that bridges the arts and the

sciences... All students in the program are required to complete graduate level work in both the arts and computer science

43. Hoganson, Ken (December 2001). "Alternative curriculum models for integrating computer science and information systems analysis, recommendations, pitfalls, opportunities, accreditations, and trends". Journal of Computing Sciences in Colleges. 17 (2): 313–325. ISSN 1937-4771. The field of information systems as a separate discipline is relatively new and is undergoing continuous change as technology evolves and the field matures
44. Khazanchi, Deepak; Bjorn Erik Munkvold (Summer 2000). "Is information system a science? an inquiry into the nature of the information systems discipline". ACM SIGMIS Database. 31 (3): 24–42. doi:10.1145/381823.381834. ISSN 0095-0033. S2CID 52847480. From this we have concluded that IS is a science, i.e., a scientific discipline in contrast to purportedly non-scientific fields
45. Denning, Peter (June 2007). "Ubiquity a new interview with Peter Denning on the great principles of computing". 2007 (June): 1. People from other fields are saying they have discovered information processes in their deepest structures and that collaboration with computing is essential to them.
46. "Computer science is the study of information" New Jersey Institute of Technology, Gutenberg Information Technologies Archived September 15, 2008, at the Wayback Machine
47. "Computer science is the study of computation." Computer Science Department, College of Saint Benedict Archived 2007-02-03 at the Wayback Machine, Saint John's University

ДОДАТКИ

Додаток А. Модель бази даних відділу постачання ТОВ «Спецпідземінжбуд»

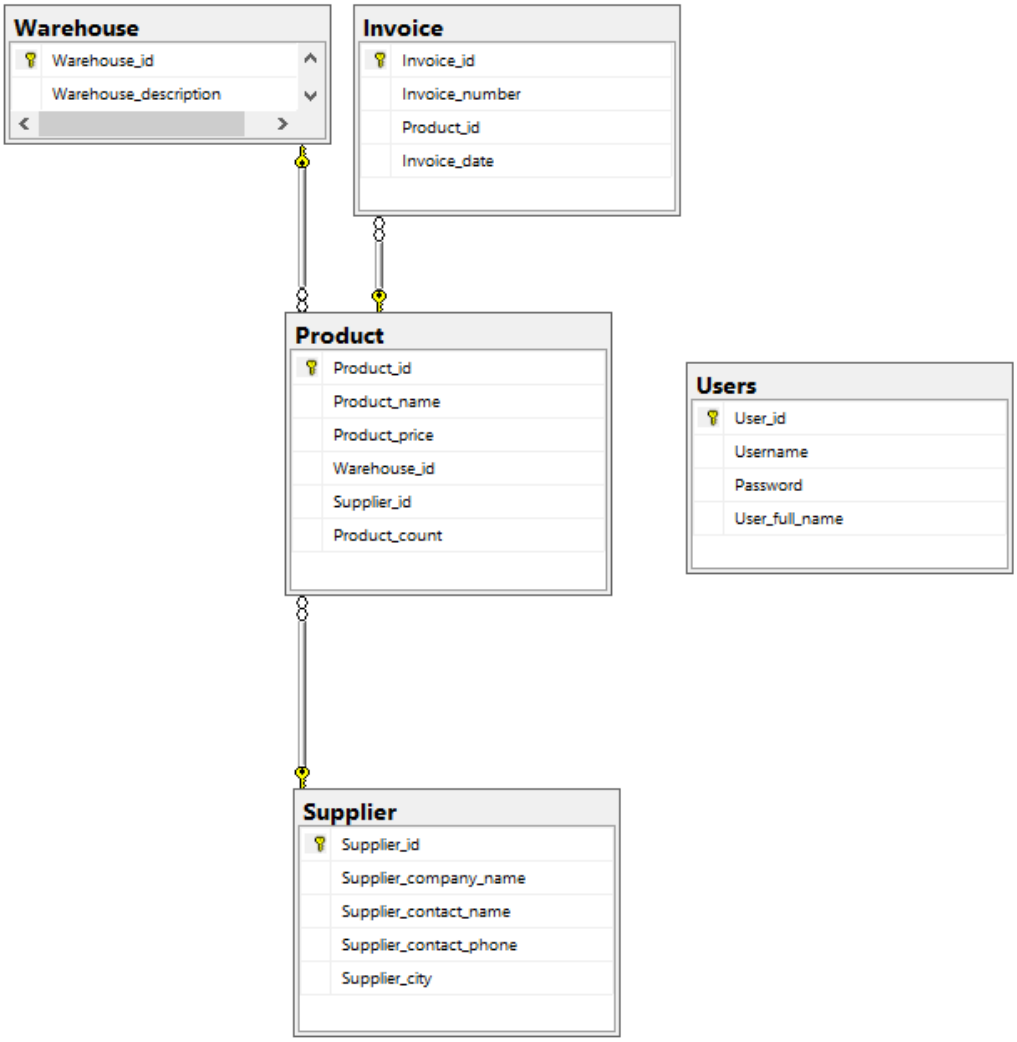

Рисунок 1 — Модель бази даних

Додаток Б. Допоміжні діаграми

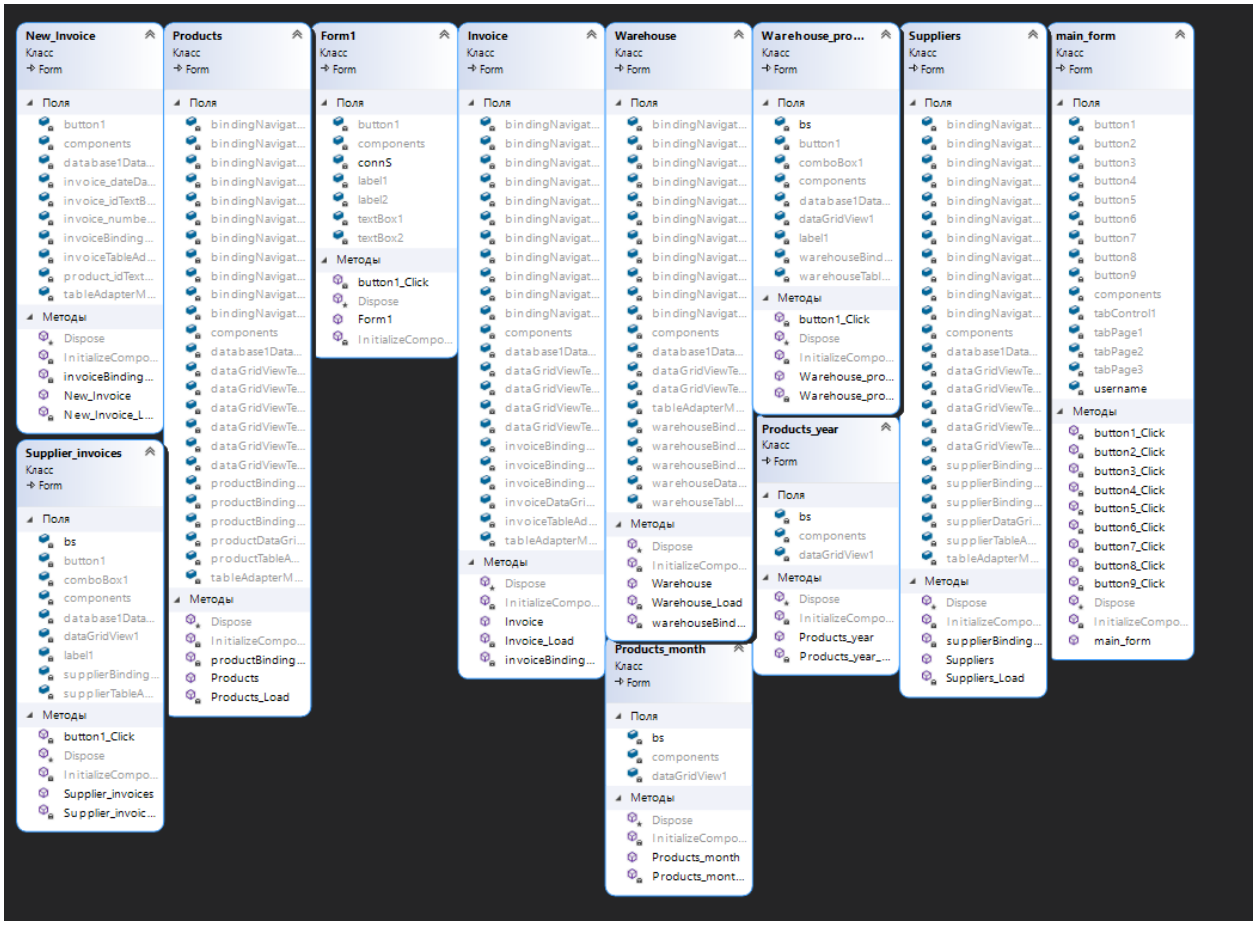

Рисунок 2 — Діаграма класів програмного продукту

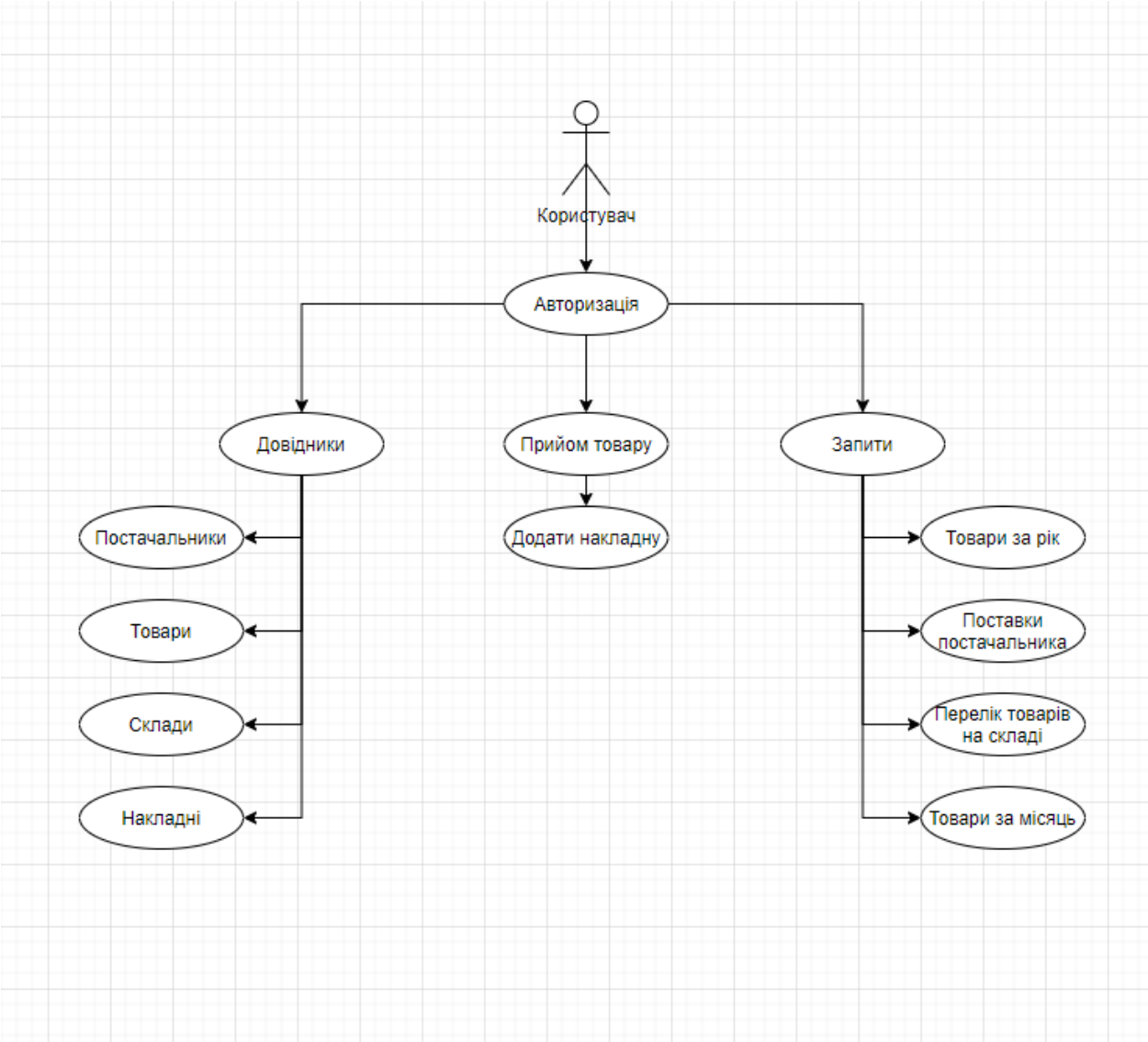

Рисунок 3 — Діаграма варіантів використання

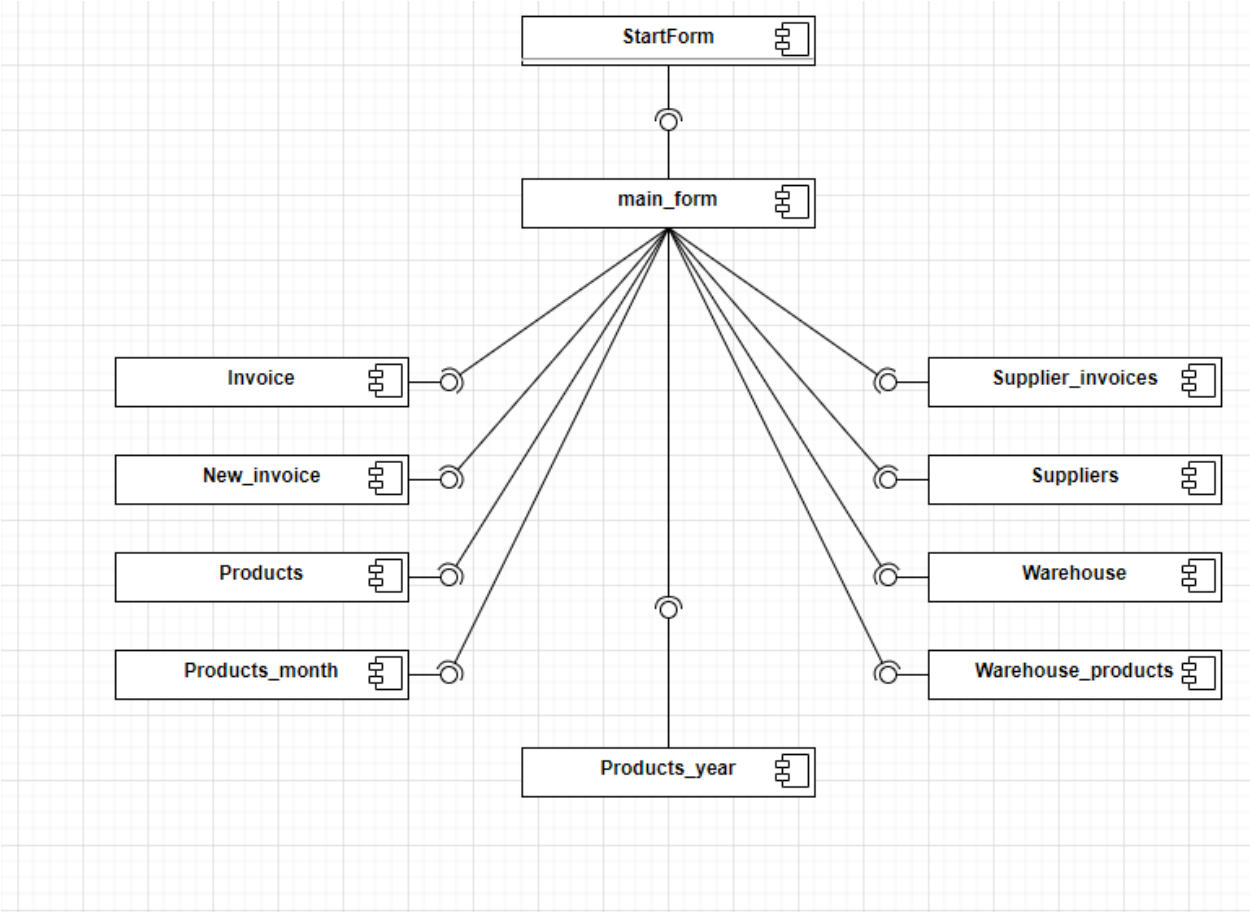

Рисунок 4 — Діаграма компонентів

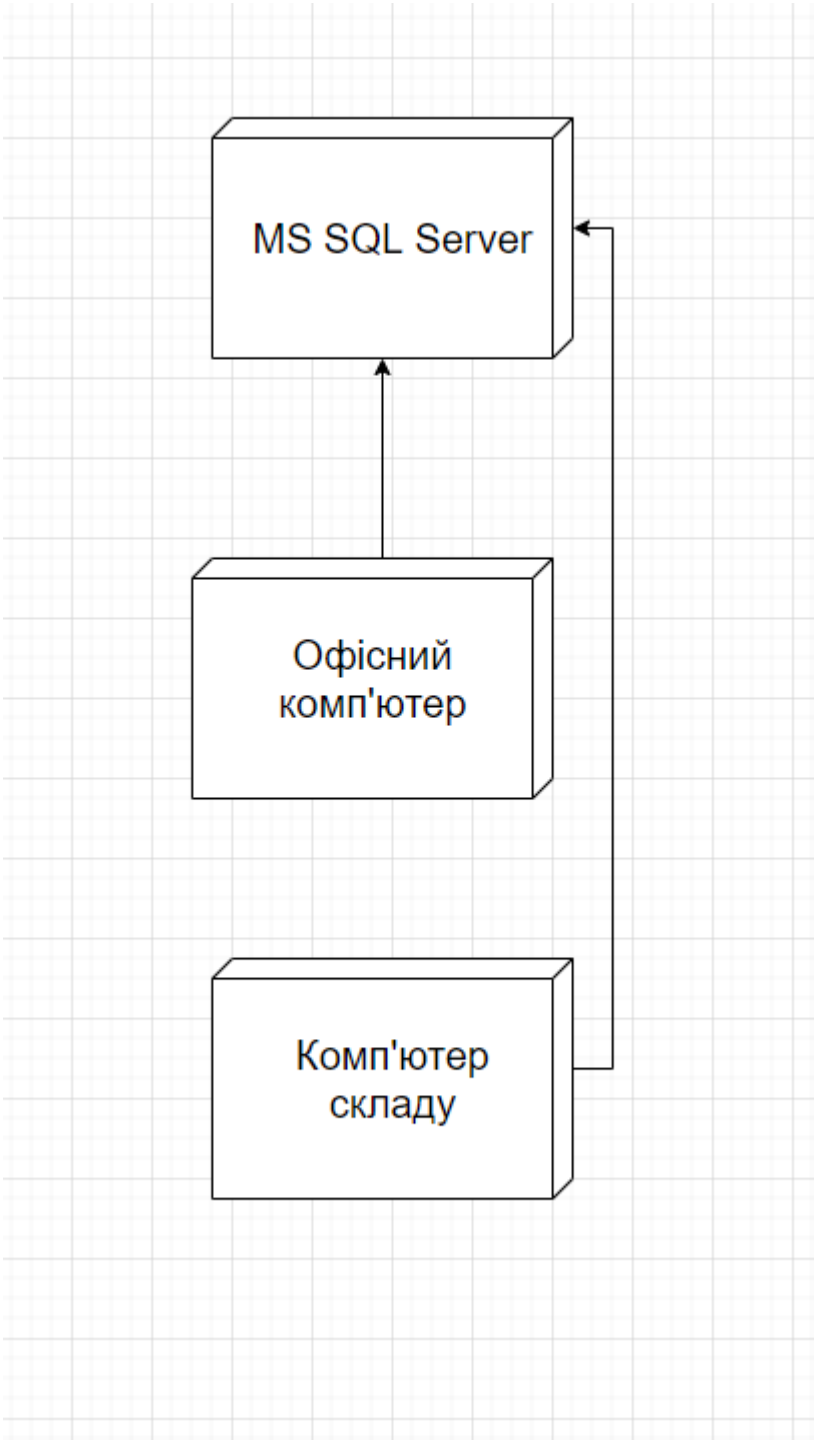

Рисунок 5 — Діаграма розгортання

Додаток В. Системні форми

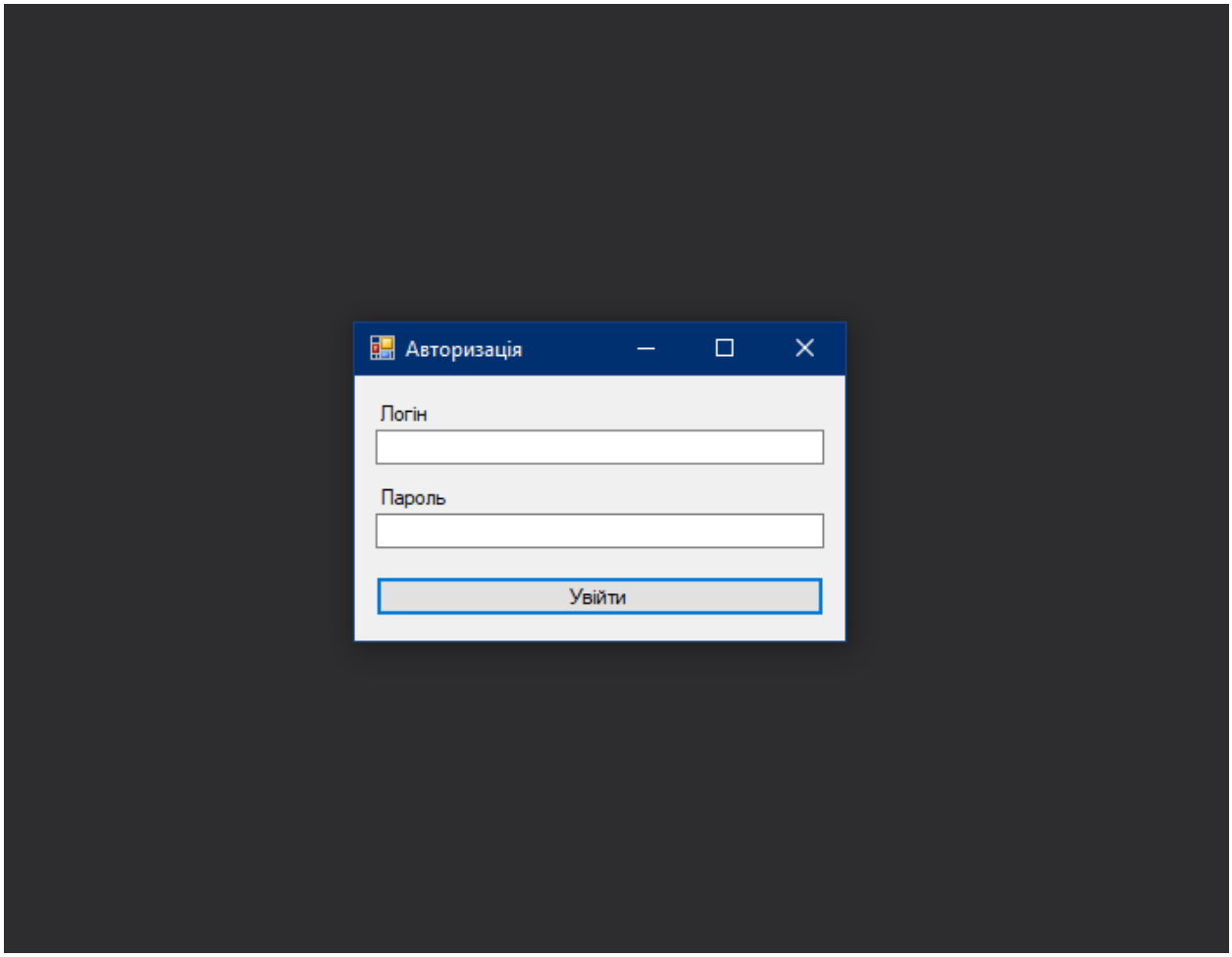

Рисунок 6 — Форма авторизації

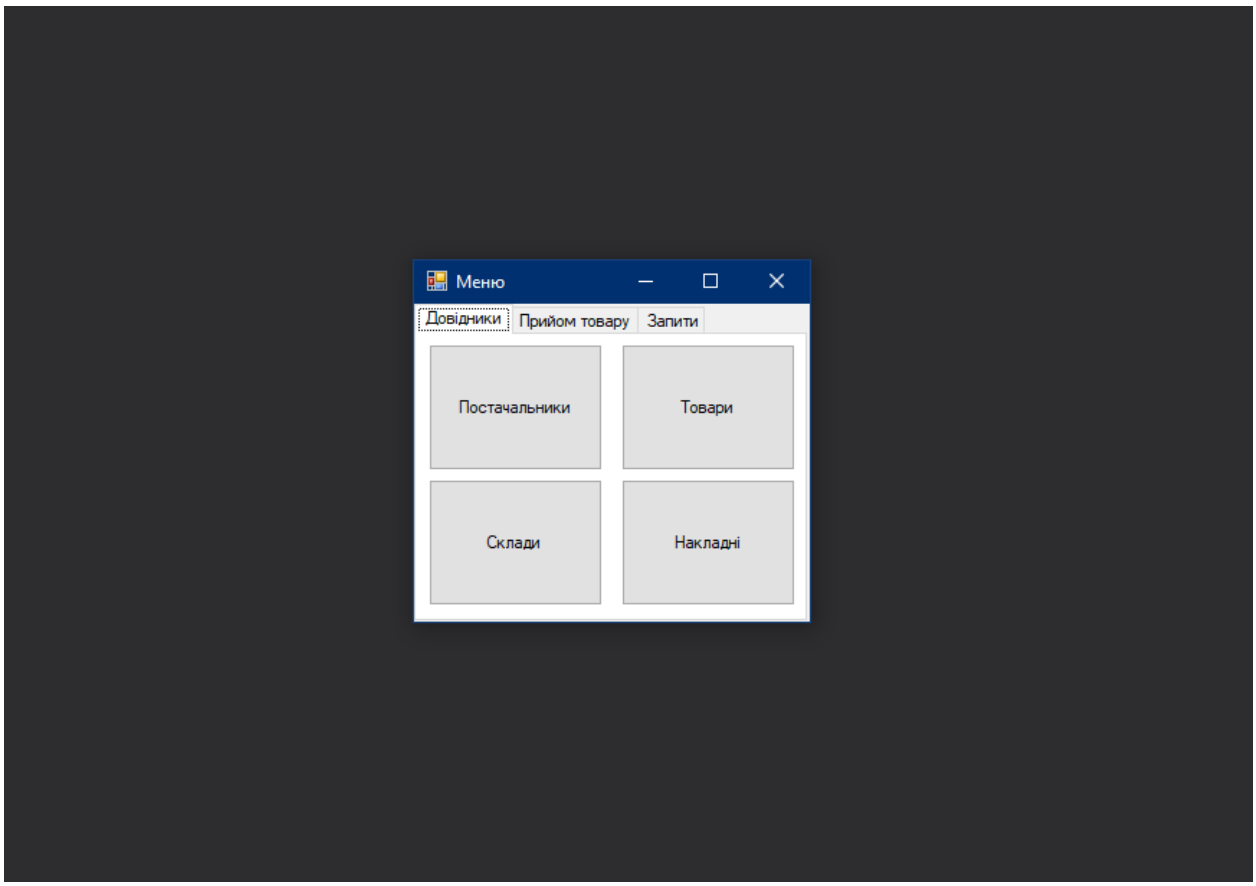

Рисунок 7 — Форма головного меню (1-ше вікно)

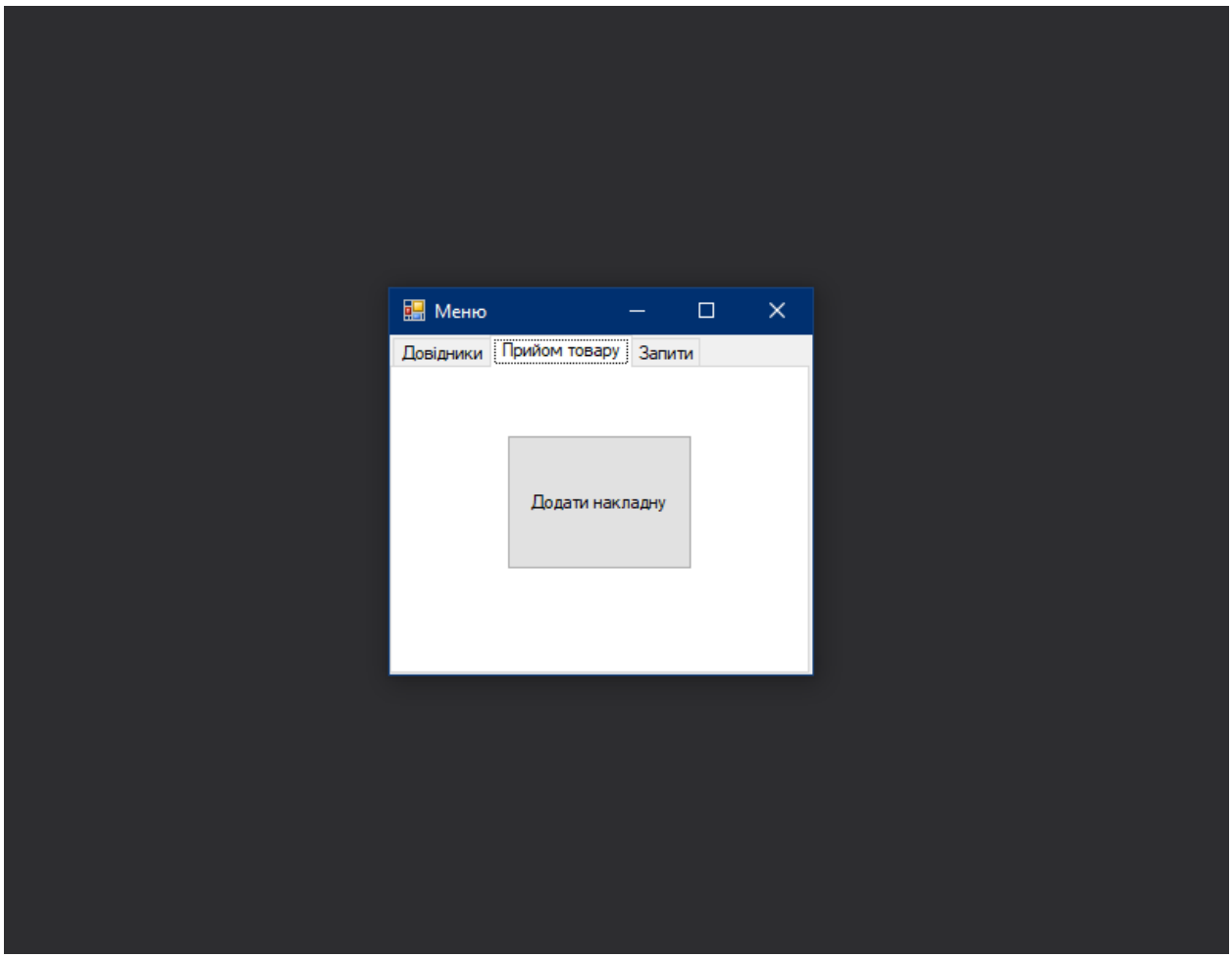

Рисунок 8 — Форма головного меню (2-ге вікно)

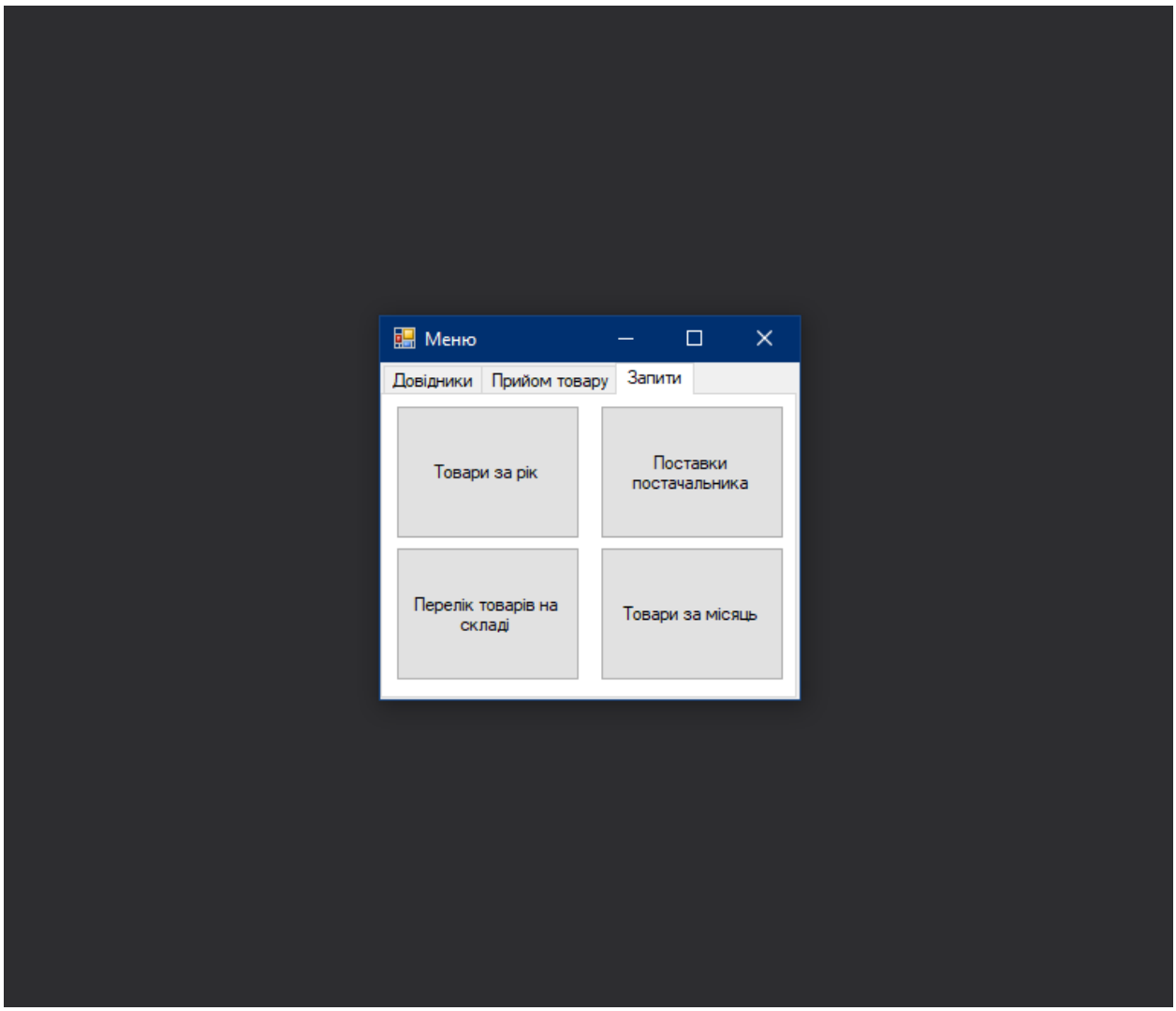

Рисунок 9 — Форма головного меню (3-є вікно)

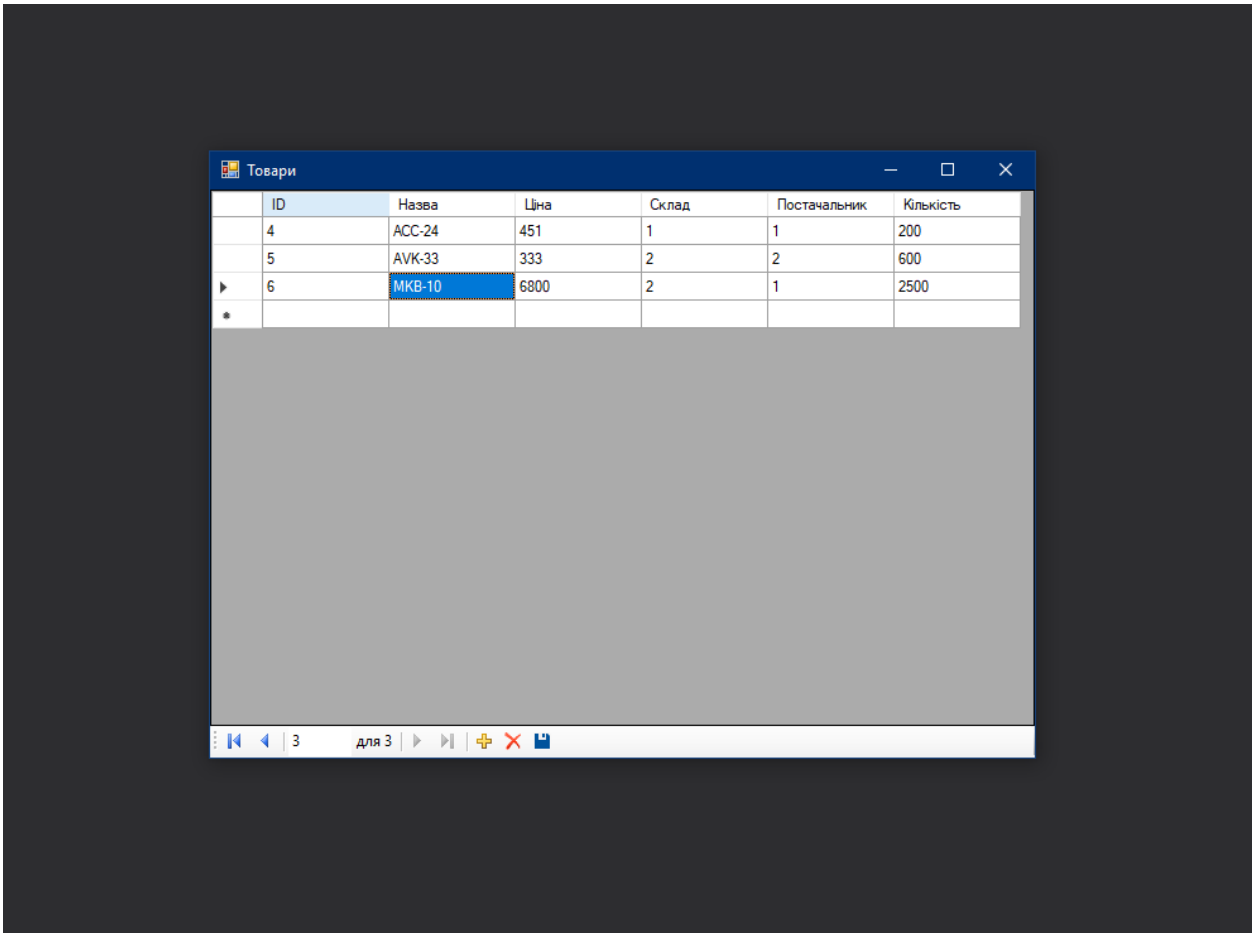

Рисунок 10 — Форма товарів

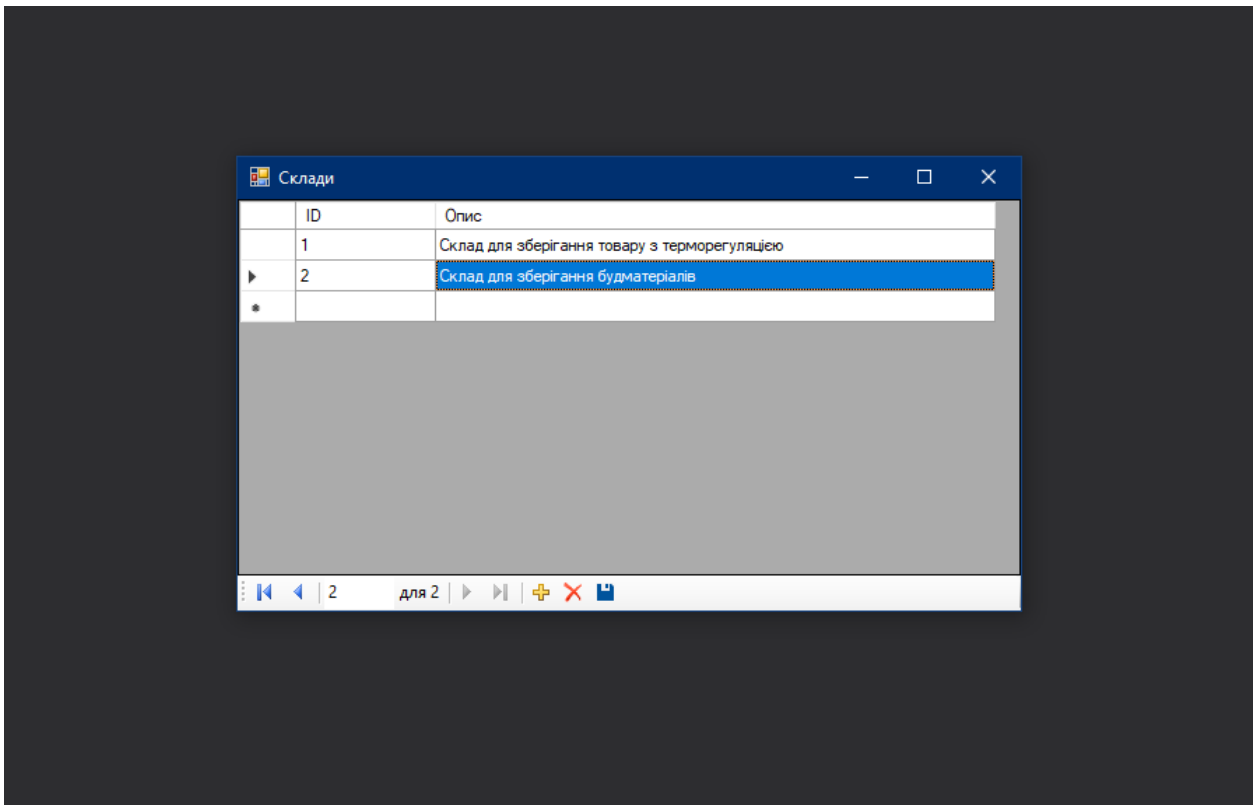

Рисунок 11 — Форма складів

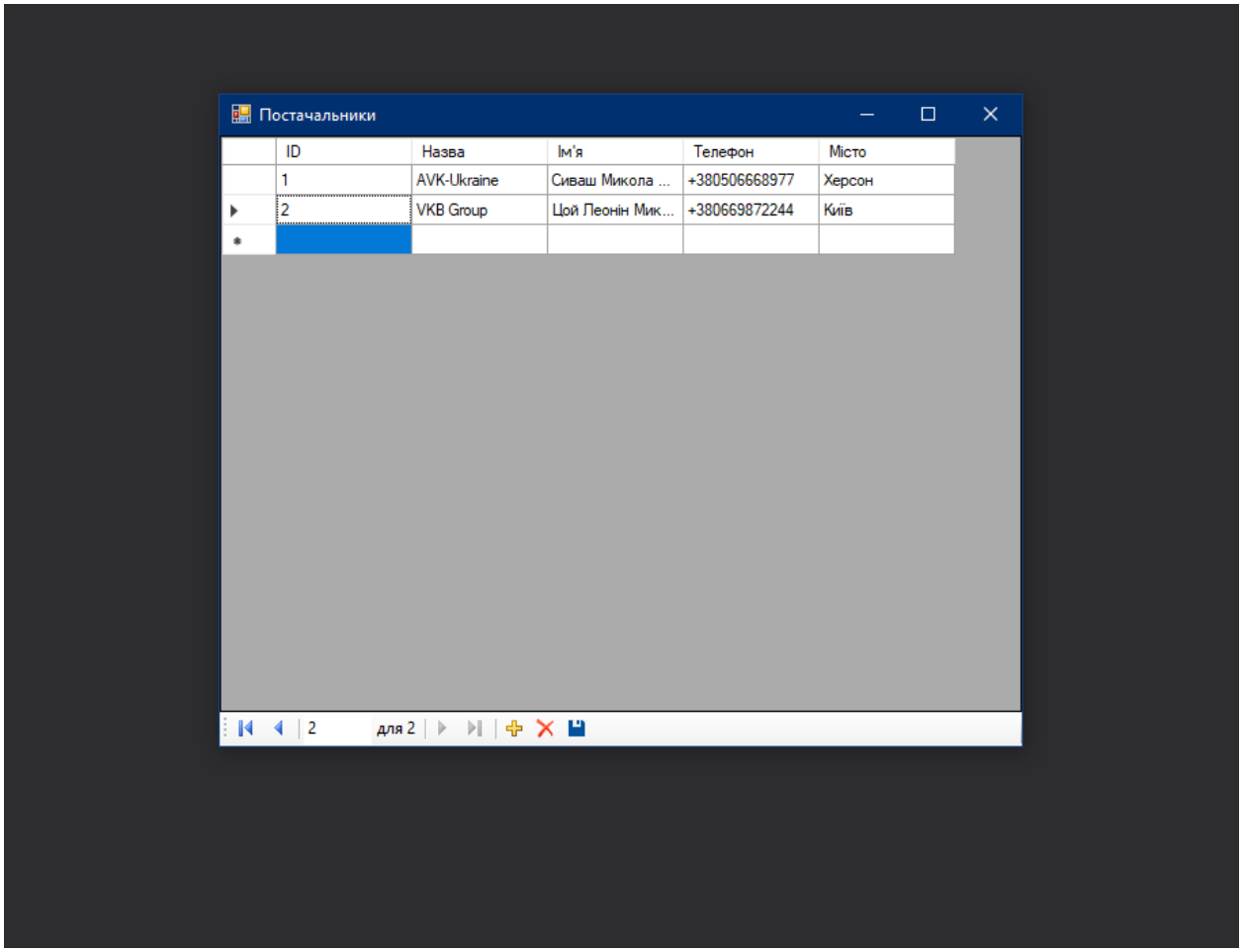

Рисунок 12 — Форма постачальників

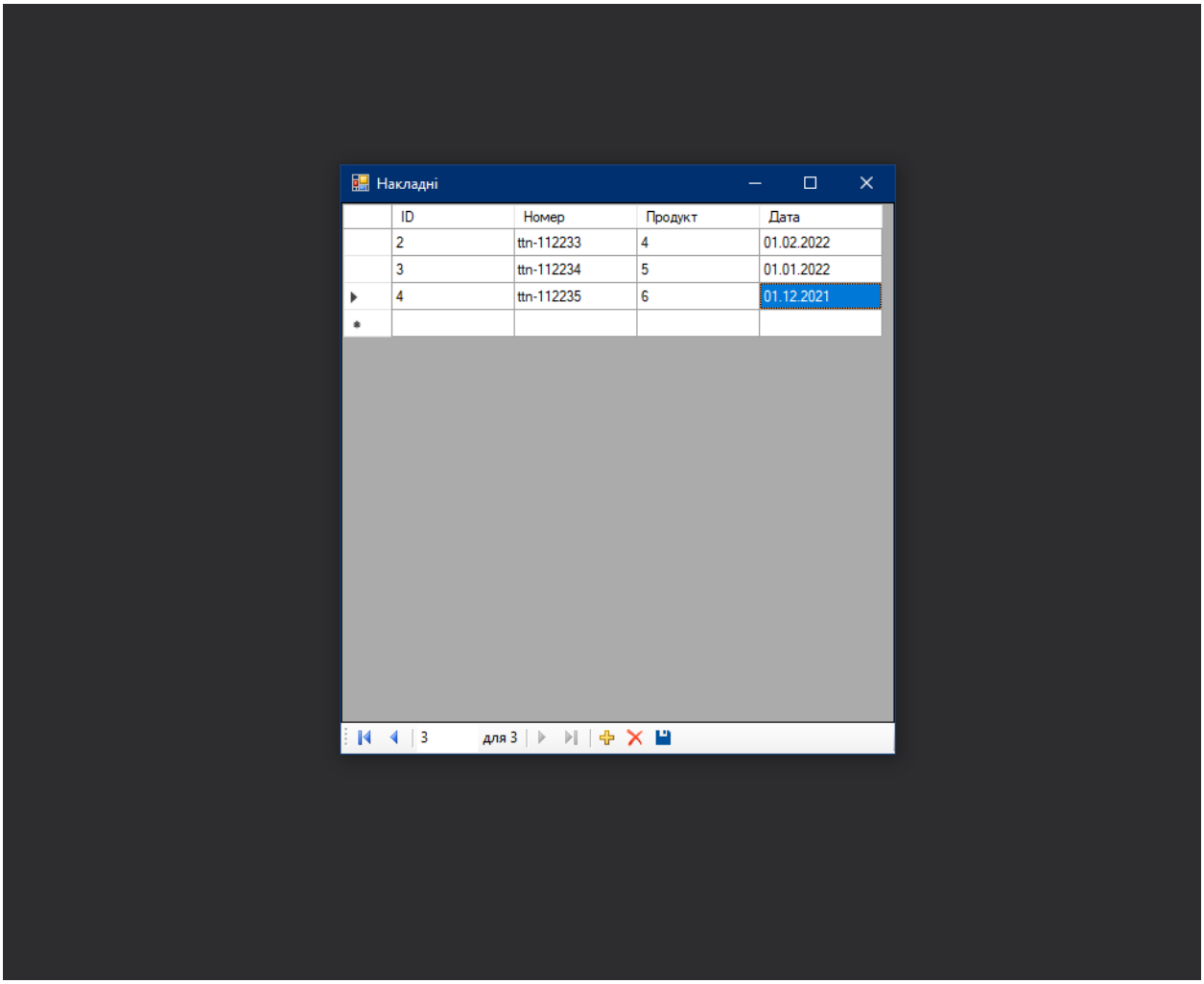

Рисунок 13 — Форма накладних

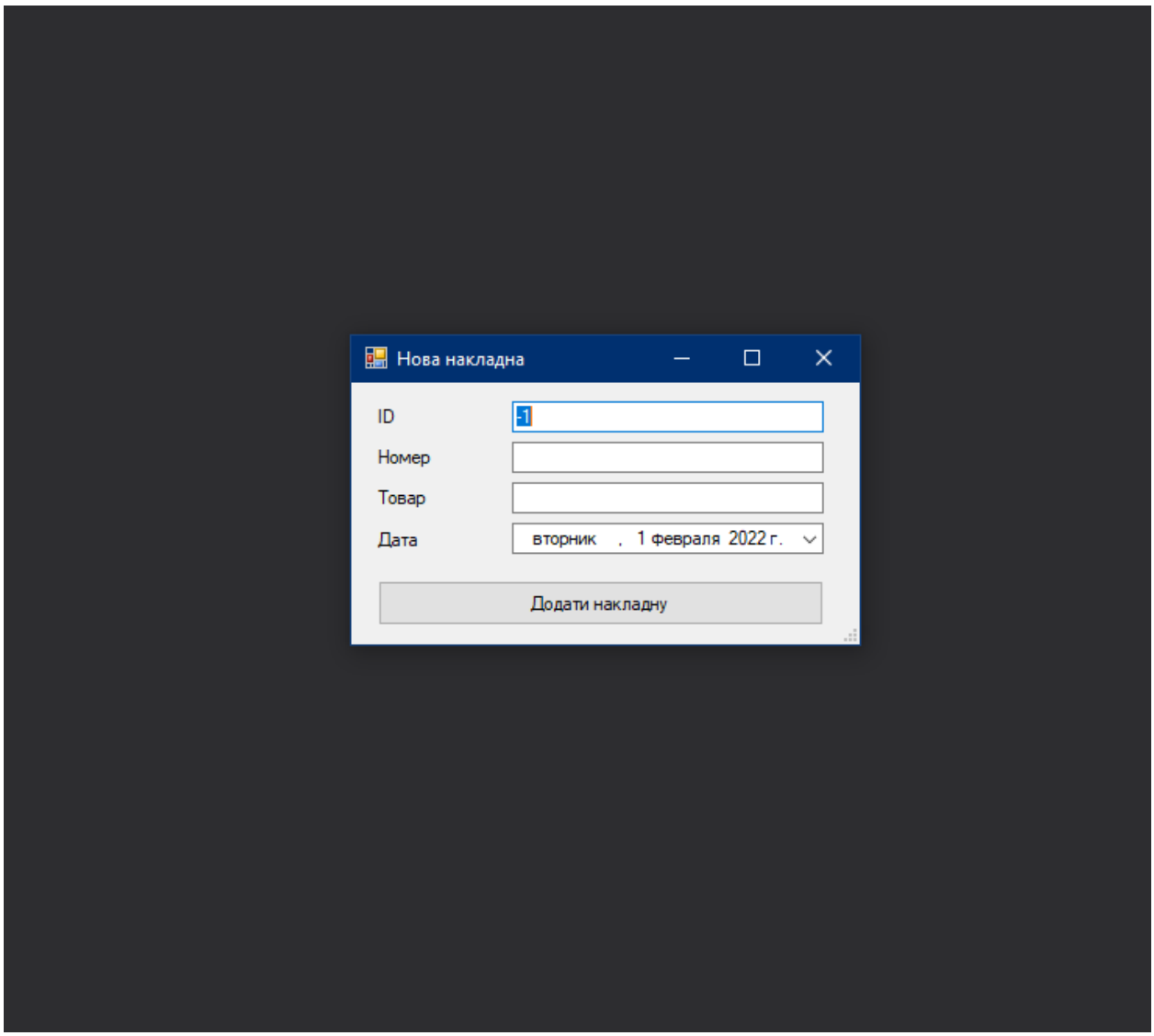

Рисунок 14 — Форма нової накладної

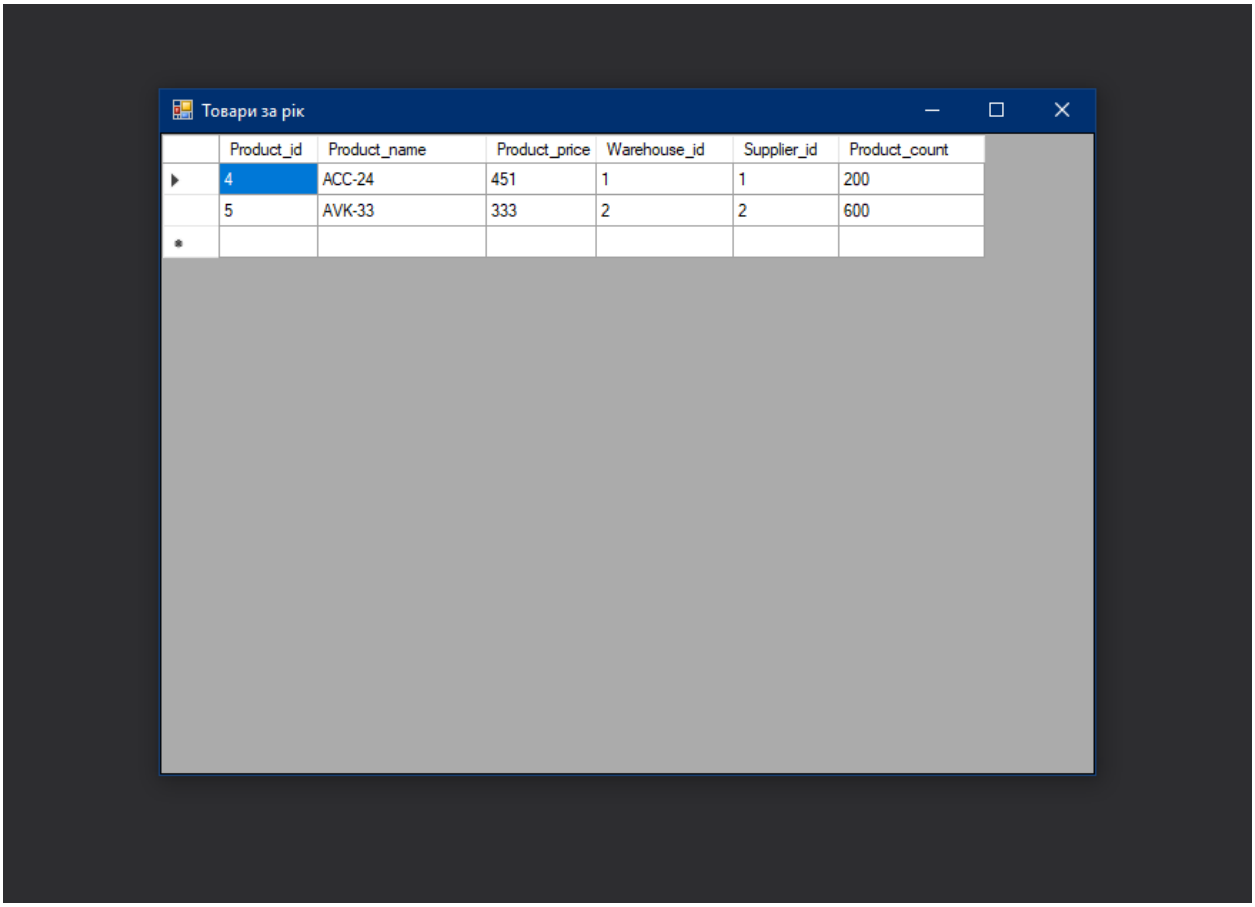

Рисунок 15 — Форма товарів за рік

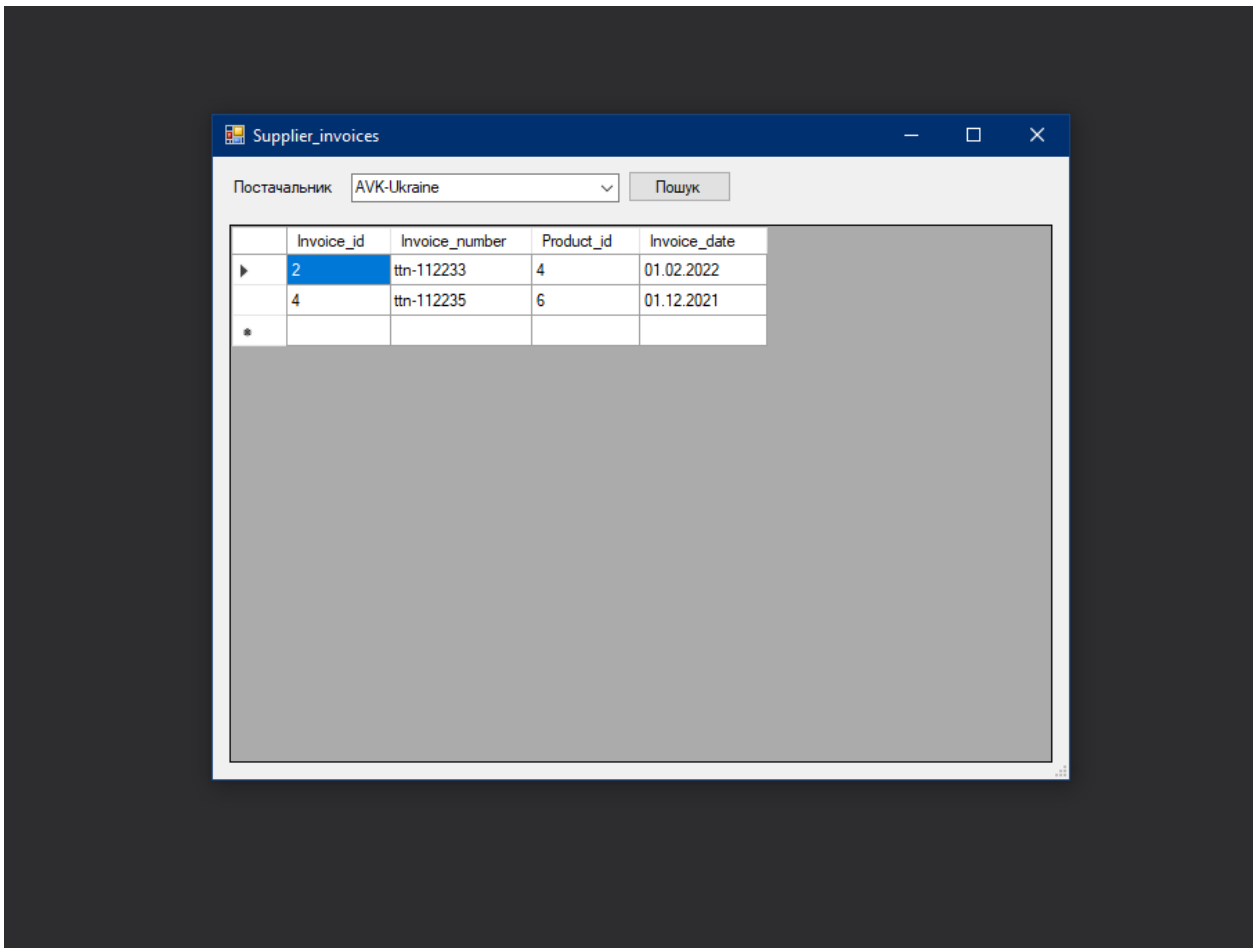

Рисунок 16 — Форма поставок постачальника

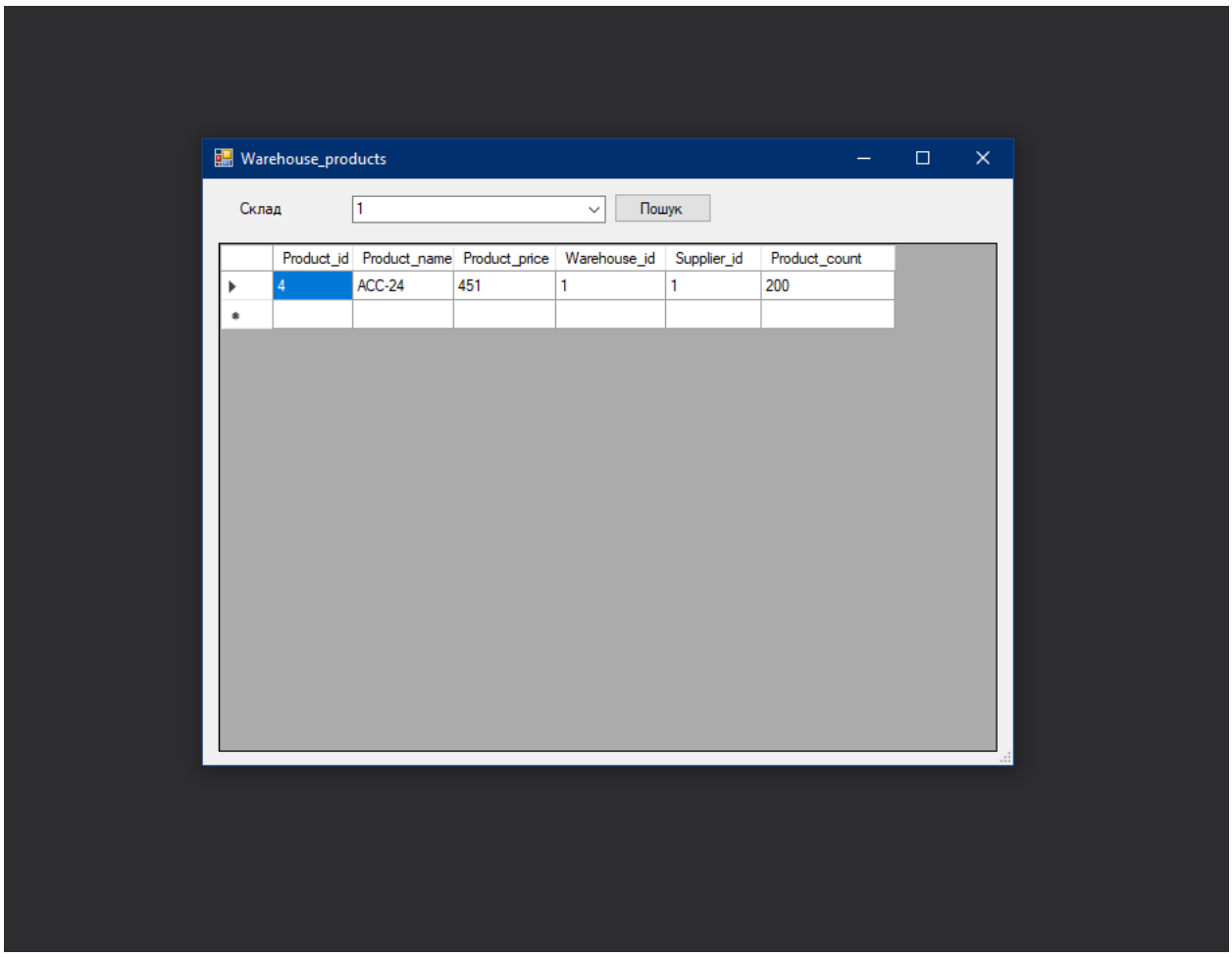

Рисунок 17 — Форма товарів на складі

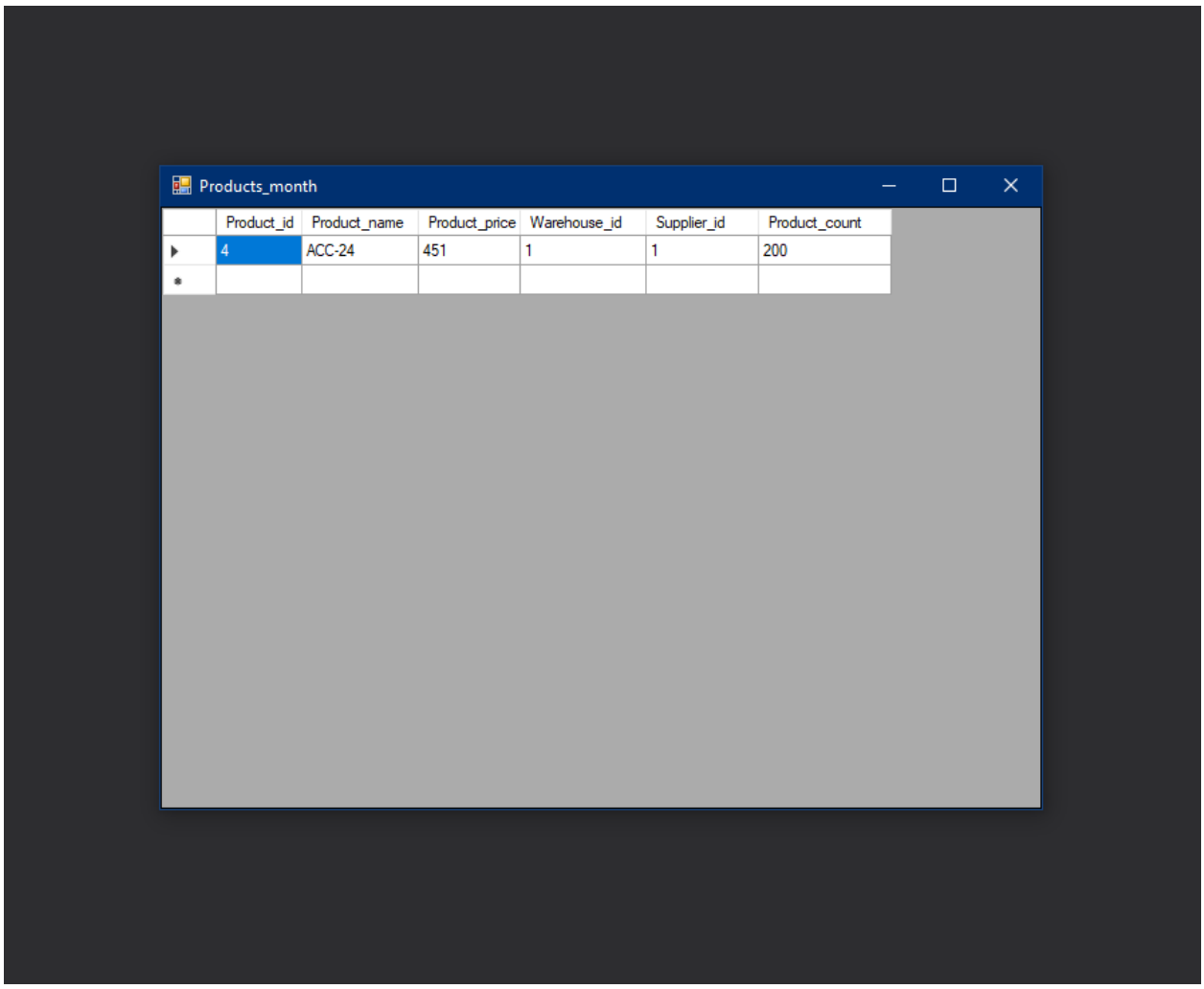

Рисунок 18 — Форма товарів за місяць