Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна система складського обліку товарно-матеріальних цінностей»

Виконав: студент 2 курсу, групи 2КІ-20м
напряму підготовки (спеціальності)
123 — «Комп'ютерна інженерія»
_____Даниленко М. С.

Керівник: к.т.н., доц. каф ОТ

_____Колесник І.С.

«____» _____ 2021 р.

Опонент: к.т.н., доц., зав. каф МБІС

_____Карпінець В.В.

«____» _____ 2021 р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
«____» _____ 2021 р.

Вінниця ВНТУ - 2021

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень магістр
Спеціальність 123 — «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
обчислювальної техніки
_____ проф., д.т.н. О.Д. Азаров

«___»_____2021 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Даниленко Максима Сергійовича

1 Тема роботи «Інформаційна система складського обліку товарно-матеріальних цінностей»
керівник роботи Колесник Ірина Сергіївна, д.т.н., доцент,
затверджені наказом вищого навчального закладу від 24.12.2021 р. №227

2 Строк подання студентом роботи 15.12.2021 р.

3 Вихідні дані до роботи: Проаналізувати методи і засоби для розробки інформаційної системи складського обліку. Розробити програму, що дозволяє ефективно керувати товарно-матеріальними цінностями складу. Розробити інтегровану рекомендаційну систему для підвищення ефективності закупки товару.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити). Вступ. Огляд і аналіз існуючих систем складського обліку. Огляд і аналіз існуючих рекомендаційних систем. Розробка методів обробки інформації системою. Розробка алгоритму рекомендаційної системи. Розробка програми системи складського обліку. Розрахунок економічної доцільності створення програмного засобу інформаційної системи складського обліку товарно-матеріальних цінностей.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Графічні зображення роботи програми

6 Консультанти розділів роботи представлені в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1,2,3	Колесник І. С., д.т.н., доцент		
4	Лесько О. Й., к.е.н., професор		

7 Дата видачі завдання 07.09.2021 р.

Календарний план наведено в таблиці 2.

Таблиця 2 — Календарний план

№	Назва етапів виконання магістерської роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі роботи	07.09.21	
2	Огляд і аналіз існуючих систем складського обліку	08.09-09.09.21	
3	Огляд і аналіз існуючих рекомендаційних систем	10.09-18.09.21	
4	Розробка методів обробки інформації системою	19.09-01.10.21	
5	Розробка алгоритму рекомендаційної системи	12.10-22.10.21	
6	Розробка програми системи складського обліку	22.10-31.10.21	
7	Розробка інтерфейсу та тестування програми системи складського обліку	01.11-10.11.21	
8	Підготовка матеріалів та опис розробки системи складського обліку	11.11-16.11.21	
9	Розрахунок економічної частини роботи	17.11-30.11.21	
10	Оформлення пояснювальної записки та ілюстративного матеріалу	01.12-06.12.21	
11	Аналіз виконання роботи, висновки, додатки	07.12-06.12.21	
12	Перевірка якості виконання магістерської роботи та усунення недоліків	15.12.21	

Студент _____ Даниленко М. С.

Керівник роботи _____ Колесник І. С.

АНОТАЦІЯ

УДК 004.42

Даниленко М. С. Інформаційна система складського обліку товарно-матеріальних цінностей. Магістерська кваліфікаційна робота зі спеціальності 123 – комп'ютерна інженерія, освітня програма – комп'ютерна інженерія. Вінниця: ВНТУ, 2021. 136с.

На укр.мові. Бібліогр.: 31 назв; рис.: 15 ; табл. 11.

Магістерська робота присв'ячена розробці інформаційної системи складського обліку з інтегрованим сервісом рекомендації закупівлі. В роботі був проведений аналіз відомих методів розробки систем складського обліку та алгоритмів для роботи рекомендаційних систем.

У ході роботи було розроблено систему складського обліку з інтегрованою рекомендаційною системою, що працює на базі «бандитських» алгоритмів.

В магістерській роботі було виконано економічні розрахунки по визначенню доцільності розробки нового програмного продукту.

Ключові слова: інформаційна система, складський облік, рекомендаційна система, товарно-матеріальні цінності.

ANNOTATION

Danylenko M. S. Information system of warehouse accounting of inventory. Master's thesis in specialty 123 – computer engineering, educational program – computer engineering. Vinnytsia: VNTU, 2021 – 136 p.

In Ukrainian language. Bibliografer: 31 titles; fig .: 15; tabl. 11.

The master's thesis is devoted to the development of a warehouse accounting information system with an integrated procurement recommendation service. The analysis of known methods of development of warehousing systems and algorithms for the work of recommendation systems was carried out in the work.

In the course of the work, a warehouse accounting system was developed with an integrated recommendation system operating on the basis of "gangster" algorithms.

In the master's thesis, economic calculations were performed to determine the feasibility of developing a new software product.

Keywords: information system, warehouse accounting, recommendation system, inventory.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ССО — система складського обліку,

ТМЦ — товарно-матеріальні цінності,

ЕОМ — електронна обчислювальна машина,

ІСР — інтегроване середовище розробки,

РПК — редактор початкового коду,

БД — база даних

ВСТУП

При недостатньому рівні контролю складу, ускладнюється процес знаходження товарів, перевірка залишків і управління логістикою. Немає інформації про поточне місце розташування конкретного замовлення або певна його частина. В результаті можуть виникнути непередбачувані ситуації, які сповільнюють або повністю зупиняють нормальне функціонування підприємства, наприклад перегрупування або створення замовлення та подальший продаж товару, якого може в принципі не існувати. Аналогічні проблеми можуть виникнути і при інвентаризації устаткування, необхідного для функціонування компанії та її працівників, що теж має пагубний вплив на нормальне функціонування [1].

Використання на підприємстві автоматизованої електронної системи складського обліку не допустить виникнення описаних вище ситуацій, що підвищить ефективність та комфорт роботи, що в результаті дасть компанії більші прибутки. В процесі реалізації товарів буде точно відома поточна кількість, залишок після купівлі та продажу і точне місце розташування товару в будь-який момент часу і ці дані будуть точними.

Актуальність дослідження пов’язана з відсутністю на ринку систем складського обліку універсального та високоефективного засобу, який буде мати інтуїтивно зрозумілий для користувача інтерфейс та вбудовану рекомендаційну систему для закупок.

Об’єктом дослідження є робота систем складського обліку в інтеграції з рекомендаційною системою.

Предметом дослідження є методи і алгоритми роботи рекомендаційних систем для їх застосування в системі складського обліку.

Мета роботи полягає у розробці інформаційної системи складського обліку, що надасть підприємству можливість ефективно керувати запасами на складах, закупівлею та продажом товарів.

Для досягнення цієї мети потрібно вирішити такі **задачі:**

— проаналізувати методи розробки систем складського обліку та рекомендаційних систем;

— виявити та обрати найбільш ефективні технології для розробки програмного засобу;

— розробити метод роботи рекомендаційної системи;

— розробити алгоритм роботи рекомендаційної системи;

— обрати інструментарій для розробки програмного продукту;

— розробити та дослідити ефективність створеного прикладного програмного продукту.

Наукова новизна роботи — в функціональні можливості системи складського обліку інтегровано рекомендаційну систему, що є стійкою до проблеми холодного старту в результаті застосування алгоритмів корекції вибірки результатів.

Практична цінність роботи полягає в використанні її для продуктивної та прибуткової роботи підприємства.

Методи дослідження магістерської роботи: збір та обробка статистичних даних.

Апробація результатів роботи здійснена під час публікації тез доповіді.

Публікації: СИСТЕМА СКЛАДСЬКОГО ОБЛІКУ ТОВАРНО – МАТЕРІАЛЬНИХ ЦІННОСТЕЙ НА БАЗІ ПЛАТФОРМИ .NET FRAMEWORK [Текст] М.С. Даниленко // XLIX Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2020): Тез. Доп. — Вінниця, 2020. — 1. Режим доступу https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2020/paper/view/9353/7714 [27].

1 АНАЛІЗ ІСНУЮЧИХ СИСТЕМ СКЛАДСЬКОГО ОБЛІКУ ТА РЕКОМЕНДАЦІЙНИХ СИСТЕМ

1.1 Аналіз ринку систем складського обліку

На ефективність роботи компанії впливає велика кількість різноманітних факторів та підхід до виконання ключових функцій. Одним з ключових факторів впливу є підхід до обліку ТМЦ на складі підприємства, від якого напряму залежить стабільність роботи. Цей підхід використовується для постійного обліку кількості та асортименту, що зберігається на складі. Забезпечити їх збереження без обліку товарів на складі складно. Для цієї процедури використовується інвентарна картка – це офіційно затверджена форма, що забезпечує рух на складі матеріалів різних типів, розмірів та класів. Заповнюється кожного номенклатурного номера матеріалу. Ними керує фінансово відповідальна людина, наприклад, керуючий складом або депозитарієм [1].

Письмова домовленість із компанією зазвичай укладається до передачі складських матеріалів керуючому складу. У ньому описані види робіт, що виконуються співробітником, та ступінь відповідальності у разі втрати або пошкодження продукції, що зберігається на складі.

Дуже важливий та необхідний сегмент діяльності компанії – це налагоджений процес обліку складських матеріалів. Для ефективного функціонування складу використовуються дві загальні системи обліку: серійна та сортова. Але незалежно від зробленого вибору матеріально відповідальні співробітники ведуть облік натурою. Ця процедура виконується виходячи з товарних і дохідних замовлень [2].

Програми управління складом спрямовані на керування переміщенням товарів на складах в режимі реального часу. Незалежно від профілю та сфери діяльності підприємства, автоматизація керування складськими масивами дає можливість вивести продуктивність на новий рівень для будь-якої компанії, вести належний облік складу та зручно керувати документообігом і логістикою.

Системи керування складом використовуються компаніями для автоматичного проведення прийому, розташування, збереження, обробка та вивантаження товарних одиниць.

Сьогодні існує велика кількість систем управління складом, і кожен власник бізнесу може індивідуально підібрати відповідну систему для свого підприємства. Є складські програми від українських, російських, а також західних розробників, численні системи управління складом для малих і великих підприємств.

Ці системи дозволяють спростити основні процеси вашого складу. Послуги можуть покращити управління всім торговим процесом і приймати нові ефективні рішення, які зменшують витрати, збільшують продажі, підвищують продуктивність і якість продукції.

Основні цілі при використанні ССО:

— покращення управління приміщеннями складу;

— конкретна інформація про розташування товару в межах конкретного приміщення складу;

— автоматизація процесу обробки товару;

— оптимізація простору складського приміщення;

— більш ефективне зберігання товару з обмеженим терміном реалізації;

— групування замовлень;

— автоматизація упакування товару;

— автоматизація створення звітів;

— полегшення при проведенні інвентаризації;

— керування обслуговуючим персоналом;

— транзитне вивантаження з використанням складського приміщення;

— відслідковування термінів реалізації товару;

— керування кінцевими вузлами зберігання товару;

— підвищення рівня безпеки при роботі з небезпечними матеріалами.

Існує багато систем, як розроблених з нуля під ключ, так і системи обліку товарів на складах на основі платформи 1С. Розглянемо чотири найпопулярніші за статистичними даними з порталу CMS CS-CART з урахуванням наступних критеріїв:

— складність при освоєнні;

— кількість функціоналу;

— відмовостійкість;

— наявність і якість технічної підтримки [3].

1С-Логістика: Управління складом — програмне забезпечення однієї з найстаріших компаній, що працюють на ринку - 1С (рисунок 1.1). Платформа є універсальною, є можливість налаштування під потреби конкретного підприємства.

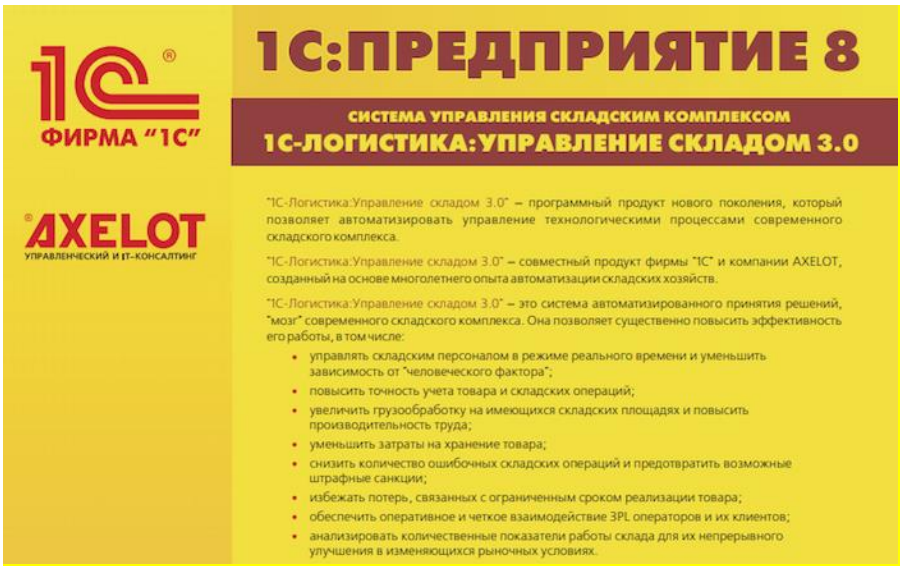


Рисунок 1.1 — ССО 1С-Логістика: Управління складом

До базової системи можуть бути підключені додаткові плагіни. Наприклад, побудова 3D-складів, підключення додаткового обладнання, що працює з голосовими технологіями «Voice Picking» і «Peak to Light», розрахунок відповідальних служб зберігання.

Працює з різними пристроями, включаючи TSD, принтери етикеток, сканери штрих-кодів тощо [4].

Факт надходження товару відбувається після прибуття або на підставі планового поступлення. В останньому випадку система перевіряє розбіжності

між запланованими та фактично отриманими товарами. Якщо присутні якісь розбіжності, ви можете отримати список невідповідностей.

Наявна оптимізація розподілу товарних одиниць на складі відповідно до ABC-класифікації або атрибутів товарів.

Можна автоматично отримувати інформацію про замовлення на доставку у форматі XML або JSON з підключених інформаційних систем.

Залежно від терміну придатності товару, враховуючи партію товару (FIFO, LIFO), є можливість вибору товару (палетного, картонного, штучного) для подальшого пакування та транспортування. Принцип максимального вивільнення осередку, мінімізація часу тощо.

Є можливість налаштування зони швидкого вибору з адресного складу.

Підприємство має необмежену кількість складів.

У системі наявні два варіанти публікації задач: «паперові» та за допомогою бездротових терміналів.

«Паперовий» - підходить для автоматизованих складських операцій. Задачі видаються на папері, а працівники складу вручну описують факт виконання. Після цього заповнені форми передаються працівникам, які вручну контролюють і вводять інформацію про ці операції.

Бездротовий термінал підключається до системи онлайн, щоб забезпечити роботу в режимі реального часу. Усі дані вводяться та виводяться в систему особою, яка виконує завдання, безпосередньо без участі оператора. У цей момент оператор може стежити за ходом роботи [5].

Недоліки:

— опираючись на відгуки користувачів, обширний набір функціоналу робить систему громіздкою та незрозумілою для застосування новим користувачем, для оптимізації функціоналу підприємство вимушене звертатись за допомогою кваліфікованих програмістів;

— оновлення відбувається за оплату, в результаті підприємство вимушене завжди вкладати в систему гроші для її актуалізації та доповнення новими утилітами.

1С-Логістика: Управління складом є більш підходящою для середніх і великих підприємств, з великим товарообігом та асортиментом.

МійСклад (рисунок 1.2) — повністю хмарний сервіс, який надає широкий асортимент функціоналу для керування складом: ведення обліку товарів, автоматизація їх продажів, керування фінансами, має лаконічний і зрозумілий інтерфейс користувача.

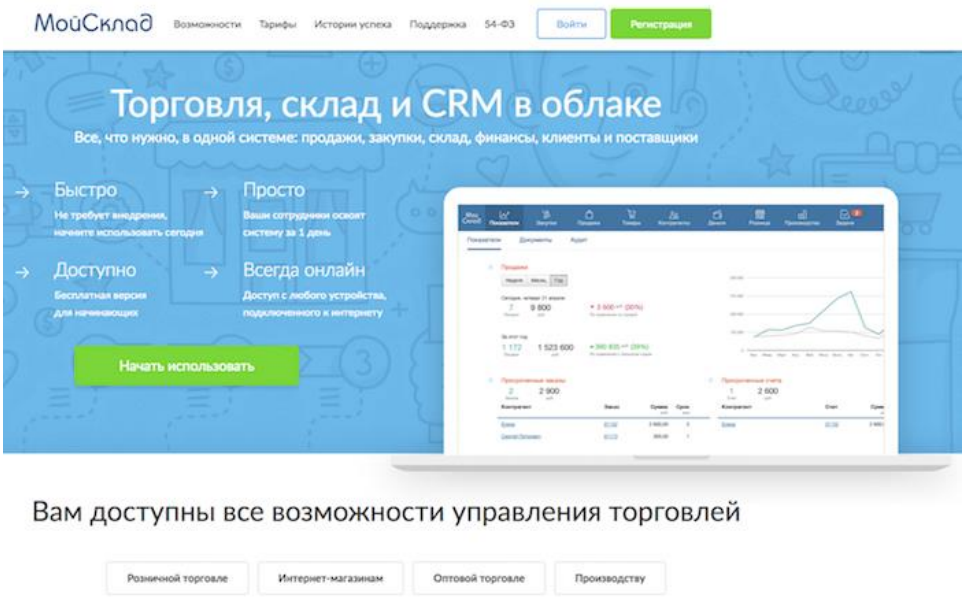


Рисунок 1.2 — ССО МійСклад

Можливості:

— підтримка корекції товарів в веб-вітрині, обробка замовлень з кількох джерел в одній системі, докладна аналітика;

— відсутнє програмування та синхронізація з сторонніми ССО;

— проста і зрозуміла для нових користувачів;

— режими роботи «під замовлення» і зі складу, друк чеків на налаштованому в системі фіскальному реєстраторі;

— гнучкі фільтри та масова обробка замовлень;

— наявність функціонального мобільного додатку, наявна можливість роботи без підключення до мережі з синхронізацією при повторному зв’язку;

— функціонал для роботи з кур’єрською доставкою;

— інформативні звіти про залишки — поточні і з урахуванням резервувань;

— комплектація товарів та їх подальший продаж;

— статистика купівлі по товарній одиниці і по контрагенту, імпортування документів постачальників з Excel;

— налаштування планів закупок;

— можливість налаштування API, що надає можливість інтеграції з іншими сервісами [6, 7].

Клієнти визначають найсуттєвішим недоліком відсутність шаблонізації для продажу і неможливість створити архівування даних.

МійСклад є підходящою системою для інтернет-магазинів, що не мають можливості постійно витрачати значні грошові суми для покупки і актуалізації системи.

СКІФ (рисунок 1.3) — хмарна система управління торгівлею і складом, завдяки лаконічному інтерфейсу і необхідному мінімуму функціоналу є простою в використанні новими користувачами.

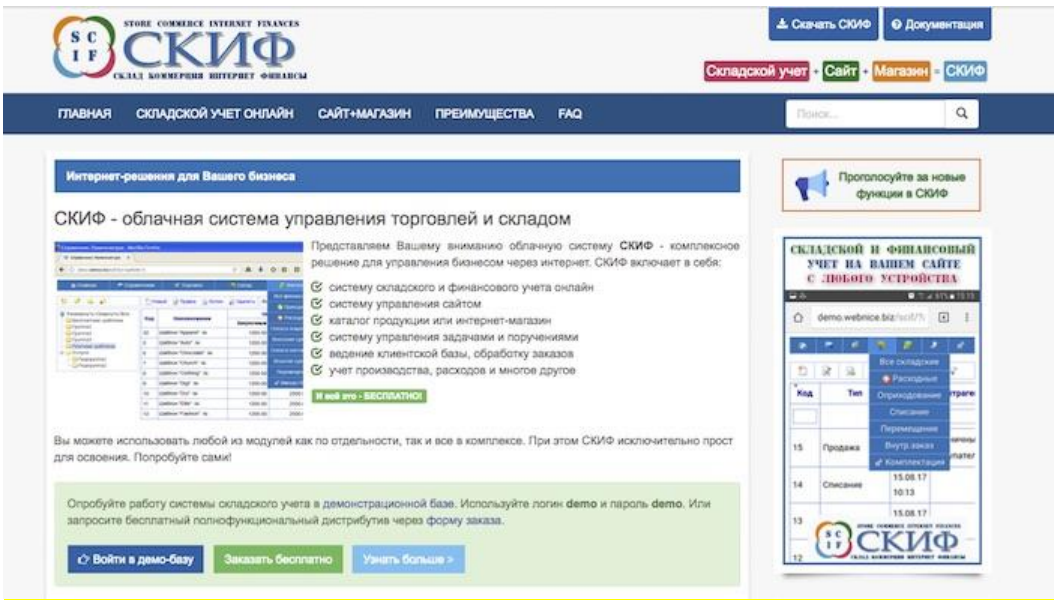


Рисунок 1.3 — ССО СКІФ

СКІФ абсолютно безкоштовний як для приватних осіб, так і для підприємств. Немає обмежень щодо кількості клієнтських програм, розміру бази та інших обмежень. За словами розробників, вони отримують прибуток,

адаптуючи систему під особисті потреби своїх клієнтів (як правило, великих підприємств).

БД знаходиться на веб-сайті клієнта, тому лише клієнт може отримати до неї доступ. Систему можна вдосконалювати самому. Систему легко зрозуміти як тим, хто вже працював в інших бухгалтерських системах, наприклад, 1С, так і тим, хто тільки вивчає основи користування комп'ютером. Інтерфейс програми схожий на інтерфейс звичайної програми Windows. Доступ до системи захищений двофакторною аутентифікацією. Ви можете підключити будь-яку клієнтську програму та оновлювати інформацію в режимі реального часу. Дозволяє створювати робочі місця та відстежувати ефективність для співробітників. Сервіс інтегрується з інтернет-магазином: продукти, залишки, ціни та фотокартки синхронізуються автоматично. Імпорт та експорт рахунків-фактур з Excel.

Найбільш вагомим недоліком є те, що систему, швидше за все, рано чи пізно потрібно буде вдосконалити. А це вимагає певних грошових вкладень.

Висновок: підходить для початківців інтернет-магазинів і тих, хто ніколи не користувався такою системою і хоче зрозуміти, що це таке [8].

Офісофф (рисунок 1.4) — мінімалістичний хмарний додаток для ведення обліку на виробничому підприємстві, спеціалізований під використання малим і середнім бізнесом.

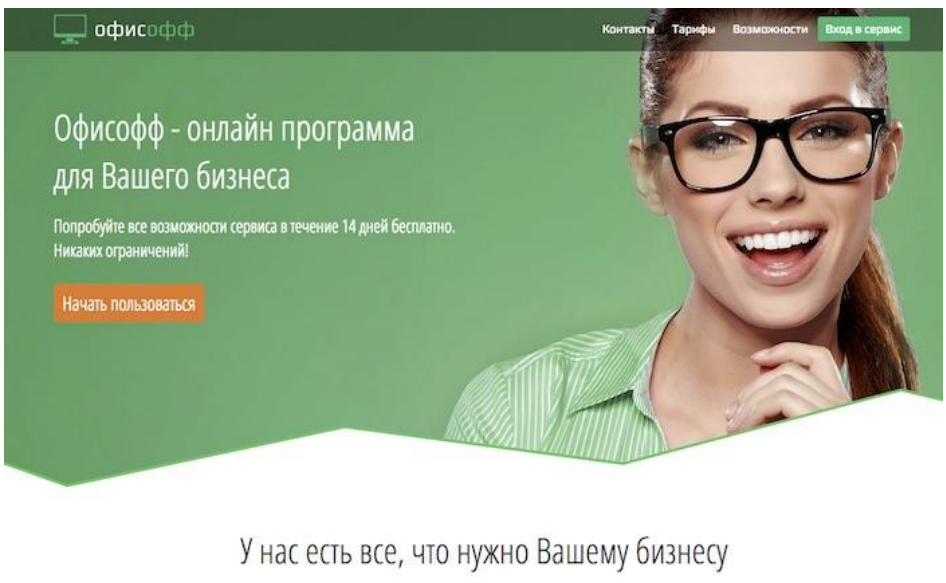


Рисунок 1.4 — ССО Офісофф

Можливості:

— повністю хмарне розміщення, що надає можливість підключитись до сервісу виключно за допомогою доступу в Інтернет;

— наявність повноцінної цілодобової технічної підтримки;

— захист збережених даних та регулярне створення резервних копій;

— ведення обліку і систематизація товарних одиниць, автоматизоване створення форм, звітів та документів та подальша відправка контрагентам за допомогою електронної пошти;

— швидке створення повноцінного набору документів відштовхуючись від рахунку;

— підтримка УНС, ЕНВД, ПДВ, можливість інтеграції за допомогою веб-інтерфейсу.

Офісофф найкраще підходить для невеликих компаній.

Проаналізувавши наявне на ринку програмне забезпечення для керування складом ми визначили основні переваги та недоліки запропонованих систем. На основі цього аналізу буде розроблено власний програмний продукт.

1.2 Рекомендаційні системи

Рекомендаційні системи стали невід'ємною частиною сучасних веб-сервісів, орієнтованих на роботу з користувачами та споживачами. Рекомендаційна система - комплекс алгоритмів, програм та сервісів, завдання якого передбачити, що може зацікавити того чи іншого користувача. В основі роботи лежить персоналізація. Вони направлені як на збільшення ефективності роботи сервісів, так і на покращення ко-ристувацького досвіду та збільшення проінформованості клієнта про надавані послуги. В залежності від моделі роботи сервісу рекомендації можуть бути як додатковою частиною до основного функціоналу, так і слугувати основою для моделі [8].

Існує чотири основних підходи до побудови прогнозів в рекомендаційних системах: колаборативна фільтрація, заснована на контенті, заснована на знаннях та гібридні.

Колаборативна фільтрація (рис. 1.5) — метод, який лежить в основі деяких рекомендаційних сервісів. Колаборативна фільтрація має два значення: конкретне і більш загальне. В загальному, колаборативна фільтрація — процес фільтрації інформації за допомогою засобів за участю співпраці між певною кіль-кістю агентів, точок зору, джерел даних і т. д. Застосування колаборативної фільтрації найчастіше пов'я-зане з дуже об’ємними вибірками даних. Колаборативні методи фільтрації були застосовані до різних видів даних, зокрема до таких як зондування та моніторинг даних, які виникають при розвідці корисних копалин на великих площах; до фінансових даних, таких як установи фінансових послуг, які об'єднують багато фінансових джерел; або в електронній торгівлі та веб-додатках, що зосереджуються на даних користувача, і т. д. Решта цієї проблеми зосереджена на колаборативній фільтрації даних, призначених для користувача, хоча деякі з методів та підходів можуть застосовуватися так само і у багатьох інших випадках [9].

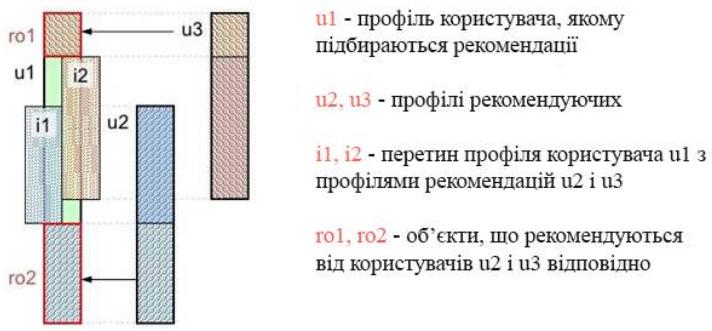


Рисунок 1.5 — Принцип роботи кореляційної фільтрації

У більш сучасному, вужчому значенні, колаборативна фільтрація — це один з методів побудови прогнозу в рекомендаційних системах, який використовує відомі уподобання (оцінки) групи користувачів для прогнозування невідомих уподобань іншого користувача. Основне припущення колаборативної фі-льтрації полягає в наступному: ті, хто однаково оцінювали будь-які предмети в минулому, схильні дава-ти схожі оцінки інших предметів і в майбутньому. Наприклад, за допомогою колаборативної фільтрації музичний додаток здатний прогнозувати, яка музика сподобається користувачеві, маючи неповний список його уподобань

(симпатій та антипатій). Прогнози складаються індивідуально для кожного користу-вача, хоча інформація, що використовується, зібрана від багатьох учасників. Це відрізняє колаборативну фільтрацію від більш простого підходу, дає усереднену оцінку для кожного об'єкта інтересу, наприклад того, що базується на кількості поданих за нього голосів. Дослідження в даній області активно ведуться і в наш час, що зокрема обумовлюється наявністю невирішених проблем у методі колаборативної фільтра-ції [8, 9].

Заснована на контенті фільтрація є основою багатьох рекомендаційних систем. На відміну від ко-лаборативної фільтрації етап знайомства з користувачем опускається. Товари та послуги рекоменду-ються з урахуванням знання них: жанр, виробник, конкретні функції тощо. Загалом застосовують будь-які дані, які можна зібрати.

За таким принципом працюють системи інтернет-магазинів, онлайн кінотеатрів та інших сервісів. Наприклад, IVI вибудовує рекомендації щодо жанрів, країн-виробників фільмів, акторів тощо.

Автори платформ використовують цей тип систем, щоб не втратити нових користувачів, даних про яких ще немає. Звідси випливають два недоліки: спочатку системи діють неточно і потрібно більше часу на реалізацію.

Заснована на знаннях фільтрація працює на основі знань про якусь предметну область: про корис-тувачів, товари та інші, які можуть допомогти в ранжируванні. Як і у випадку з фільтрацією за контентом, оцінки інших користувачів системи не враховують. Є кілька різновидів: case-based, demographic-based, utility-based, critique-based, whatever-you-want-based і т.д.

При реалізації нового проекту в залежності від сфери діяльності у рекомендаційну систему можна закласти будь-яку предметну область та ранжувати за нею.

Мінус фільтрації за знаннями — для розробки цієї системи потрібно багато часу та ресурсів, але результат їх виправдовує.

Комбінування кількох алгоритмів в рамках гібридної системи у межах однієї платформи дозволяє мінімізувати недоліки кожного. Великі сервіси та інтернет-магазини найчастіше використовують гібридні варіанти.

Існує кілька найбільш розповсюджених типів комбінування:

— реалізація окремо колаборативних та контентних алгоритмів та поєднання їх припущень;

— включення деяких контентних правил до колаборативної методики;

— включення деяких колаборативних правил до контентної методики;

— побудова загальної моделі, що включає правила обох методик.

Холодний старт — потенційна проблема в комп'ютерній інформації системи, яка включає ступінь автоматизованого моделювання даних. Зокрема, це стосується проблеми, що полягає в тому, що система не може зробити жодних висновків для користувачів або елементів, про які вона ще не зібрала достатню інформацію [5].

При проектуванні рекомендаційної системи необхідно визначити найбільш ефективний підхід для задоволення потреб сервісу, опираючись на вже наявну базу інформації, з якою може працювати система, джерела цієї інформації, орієнтованість рекомендаційної системи та можливість виникнення пробле-ми холодного старту або повторюваного холодного старту за умови додавання в сервіс нової інформа-ції.

Завдання рекомендаційної системи – проінформувати користувача про товар, який може бути най-цікавіший у час. Клієнт отримує інформацію, а сервіс заробляє на наданні якісних послуг. Послуги - це не обов'язково прямий продаж запропонованого товару. Сервіс також може заробляти на комісійних або просто збільшувати лояльність користувачів, яка потім виливається у рекламні та інші прибутки [7].

Залежно від моделі бізнесу рекомендації можуть бути його основою, як, наприклад, у TripAdvisor, а можуть бути просто зручним додатковим сервісом

(як, наприклад, у якомусь інтернет-магазині одягу), покликаним покращити Customer Experience та зробити навігацію за каталогом. зручною.

Персоналізація онлайн-маркетингу – очевидний тренд останнього десятиліття. За оцінками McKinsey, 35% виручки Amazon або 75% Netflix припадає саме на рекомендовані товари і відсоток цей, ймовірно, зростатиме. Рекомендаційні системи – це про те, що запропонувати клієнту зробити його щас-ливим.

Стандартну модель рекомендаційної системи можна описати наступними параметрами:

— предмет рекомендації – що рекомендується;

— ціль рекомендації – навіщо рекомендується (наприклад: купівля, інформування, навчання, конта-кти);

— контекст рекомендації – що користувач робить у цей момент (наприклад: дивиться товари, слу-хає музику, спілкується з людьми);

— джерело рекомендації – хто рекомендує: аудиторія (середній рейтинг ресторану в TripAdvisor), схожі за інтересами користувачі, експертна спільнота (буває, коли йдеться про складний товар, такий, як, наприклад, вино);

— ступінь персоналізації. Неперсональні рекомендації – коли вам рекомендують те саме, що й ін-шим. Вони допускають націлення регіону або часу, але не враховують ваші особисті переваги. Більше сучасний варіант – коли рекомендації використовують дані з вашої поточної сесії. Ви переглянули кілька товарів, і внизу сторінки вам пропонуються схожі. Персональні рекомендації використовують усю досту-пну інформацію про клієнта, зокрема історію його покупок;

— прозорість. Люди більше довіряють рекомендації, якщо розуміють, як саме її було отримано. Так менший ризик нарватися на «несумлінні» системи, які просувають проплачений товар або дорожчі това-ри, що ставлять вище в рейтингу. Крім того, хороша рекомендаційна система сама повинна вміти боро-тися з купленими відгуками та накрутками продавців. Маніпуляції до речі бувають і ненавмисними;

— формат рекомендації. Це може бути спливаюче віконце, що з'являється в певному розділі сайту відсортований список, стрічка внизу екрана або ще щось;

— алгоритми. До найкласичніших відносяться алгоритми Summary-based (неперсональні), Content-based (моделі засновані на описі товару), Collaborative Filtering (колаборативна фільтрація), Matrix Factorization (методи засновані на матричному розкладанні) та деякі інші [8].

2 РОЗРОБКА МЕТОДІВ ТА АЛГОРИТМІВ СИСТЕМИ СКЛАДСЬКОГО ОБЛІКУ

2.1 Розробка архітектури програмної реалізації

Для розробки програмного застосунку логіку буде інкапсульовано в клієнтську та серверну програму. Розглянемо особливості такого архітектурного рішення.

Клієнт-серверний архітектурний стиль (рисунок 2.1) є одним із архітектурних шаблонів програмного забезпечення та є найбільш розповсюдженим принципом у розробці розподілених мережних застосунків і надає можливість зв’язку та обміну даними між його частинами. Архітектура передбачає такі основні складові:

— масив серверів або одиночний сервер, який обробляє та видає інформацію за допомогою запитів;

— масив клієнтів або одиночний клієнт, який ініціює виклики до серверів і надає користувачу інтерфейс доступу;

— мережа для надання зв’язку [10].

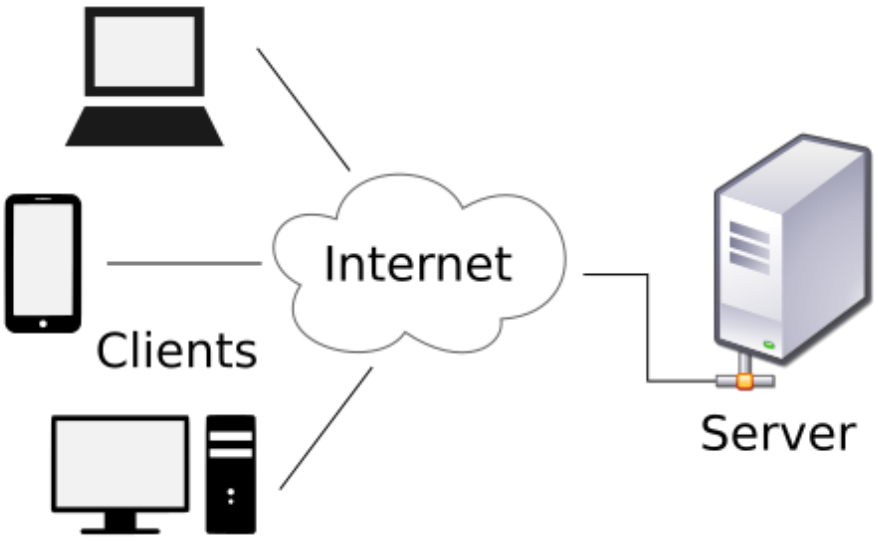


Рисунок 2.1 — Візуалізація клієнт-серверної взаємодії

Сервери працюють незалежно і паралельно, як і окремі клієнти. Між сервером і клієнтом немає жорсткого з'єднання. Часто один сервер одночасно обробляє запити від великої кількості клієнтів. З іншого боку, клієнти можуть

отримати доступ до інших серверів. Клієнт повинен мати інформацію про доступні сервери, але може не мати інформації про дії інших клієнтів.

Загалом клієнти та сервери в першу чергу вважаються програмними застосунками. Вони часто знаходяться на різних комп'ютерах, але бувають ситуації, коли дві програми, клієнт і сервер, розташовані в межаш однієї машини. У цьому контексті сервер часто називають локальним [11].

Архітектура взаємодії клієнт-сервер передбачає, серед іншого, поділ роботи між клієнтом і сервером. Логічно можна виділити три рівні роботи:

— рівень візуалізації інформації, який є передусім інтерфейсом користувача і відповідає за його доступ до даних і введення команд керування;

— прикладний рівень, на якому реалізовано більшість алгоритмів програми та виконується аналіз необхідних даних;

— рівень управління інформацією, що забезпечує зберігання та надає доступ.

Дворівнева модель клієнт-сервер передбачає взаємодію двох частин програмного забезпечення: клієнта і сервера. Починаючи з середини 1990-х років, трирівнева архітектура клієнт-сервер передбачає відокремлення рівня програми від управління даними. Є окремий програмний рівень, де зосереджена логіка програми. Програми середнього рівня можуть працювати на виділених серверах додатків, але ці програми також можуть працювати на звичайних веб-серверах. Даними також можна керувати на сервері даних.

Для роботи системи користувачі використовують стандартне програмне забезпечення, наприклад веб-браузер. Це позбавляє від необхідності завантажувати та встановлювати спеціальні програми (хоча іноді вони все одно потрібні). Однак користувачі повинні надати інтерфейс, за допомогою якого вони можуть взаємодіяти з системою та робити запити до системи. Форма, яка визначає цей інтерфейс, розміщується на веб-сайті та завантажується разом із нею.

Веб-браузер створює запит і надсилає його на сервер обробки. У разі необхідності сервер викликає програмний модуль сервера, який обробляє запит і, якщо необхідно, звертається до сервера даних. Сервери даних

виконують операції з даними, що зберігаються в системі, і формують інформаційну базу. Зокрема, за потреби, зразки можуть бути вилучені з бази даних і передані в модулі середнього рівня для подальшої обробки. Дані, з якими працюють сервери даних, часто організовуються в реляційні бази даних.

Модулі веб-сервера та сервера середнього рівня є окремими та логічно незалежними програмними модулями, але часто розміщуються на одному комп'ютері [12].

2.2 Розробка серверної частини застосунку

REST (Representational State Transfer) — стиль архітектури зв'язку розподілених компонентів програми в мережі. REST — це набір узгоджених правил, до яких звертаються при побудові розподілених гіпермедійних систем. У деяких програмних рішеннях це підвищує ефективність і спрощує архітектуру. Частини REST мають між собою взаємодію подібну до взаємодії клієнтів та серверів взаємодіють у Інтернеті.

В Інтернеті віддалене звертання до набору функціоналу може бути звичайним HTTP-запитом (зазвичай GET або POST; ці запити називаються «запитами REST»), а необхідний набір вхідних даних надсилається в якості атрибутів запиту [12].

Для веб-сервісів, створених з урахуванням REST (тобто без порушення будь-яких обмежень, накладених на них), використовується термін «RESTful».

На відміну від веб-сервісів на основі SOAP, не існує урегульованого та задокументованого стандарту для RESTful API. Насправді REST — це архітектурний стиль написання веб застосунків, а SOAP — це протокол. Хоча REST не існує як задокументований стандарт, більшість реалізацій RESTful включають в себе використання поширених в мережі Інтернет стандартів, таких як HTTP, URL, JSON і XML.

Існує п'ять основних обмежень для створення розподілених програм REST у Філдінгу. Системи REST повинні відповідати цим обмежуючим вимогам. Накладені обмеження визначають, як сервер обробляє запити

клієнтів і реагує на них. Система, що працює в рамках цих обмежень, досягає бажаних властивостей, таких як продуктивність, масштабованість, простота, змінюваність, переносимість, відстежуваність і надійність [13].

Якщо програма служби порушує ці обмеження, систему не можна вважати системою REST.

Обов'язковими для створення REST-системи обмеженнями є:

— використання клієнт-серверної взаємодії;

— не зберігається стан;

— присутність кешу;

— уніфікований інтерфейс;

— багатошаровість.

Застосунки, які не відповідають описаним вище умовам, не можуть називатися RESTful-застосунками. Якщо всі описані вимоги витримано, то додаток буде мати наступні особливості:

— відмовостійкість;

— ефективність;

— масштабованість;

— зрозумілість інтерфейсів взаємодії;

— простота інтерфейсів;

— компактність складових елементів;

— можливість робити зміни;

— розширюваність при появі нових вимог.

Серверна частина застосунку буде розробленя як RESTful API з дотриманням описаних вище вимог. Сервіс має мати такий мінімальний набір ендпоінтів:

— робота з товарами;

— робота з контрагентами;

— робота з працівниками;

— робота зі складами;

— виконання закупівлі або прийому товару;

— виконання продажу або відвантаження товару;

— формування звітів;

— імпорт та експорт даних за допомогою таблиць Excel;

— імпорт та експорт даних за допомогою HTTP-запитів з інформацією у форматі JSON.

2.3 Розробка сервісу рекомендаційної системи

Проблема холодного старту є одним із основних завдань, що виникають при побудові рекоменда-ційних систем: що рекомендувати новим користувачам та кому показувати нові об'єкти. У безконтекст-них алгоритмах це завдання вирішується за допомогою накопиченої статистичної інформації. Для вико-ристання в рекомендаційних системах розглянемо так звані «алгоритми багаторуких бандитів» та їх ефективність в умовах холодного старту [14].

Уявимо, що маємо N різних ігрових автоматів. При натисканні на ручку ми отримуємо якийсь ви-граш. Необхідно максимізувати сумарний прибуток всіх натискань на ручки. Завдання полягає у знахо-дженні оптимального способу вибору ручки на черговому кроці. Математично це можна записати так:

A – безліч доступних дій («ручок»);

Xt ∈ Rd – контекст (інформація про об'єкти та/або користувачів) на певному кроці. Він визнача-ється середовищем, у якому розглядаються багаторукі бандити;

Pt, a, x – очікувані виплати для ручки a, які можна отримати в момент часу t при заданому кон-тексті Xt;

Rt – реальний виграш.

$$a_t = argmax\, p_{t,a,x},\ д\ е\ a \in A$$

$$R_t = \sum_t r_t \to max, \text{при } t = 1,2, \ldots$$

Розглянемо деякі види алгоритмів багаторуких бандитів, а саме UCB1 та ε-greedy, disjoint-linUCB та hybrid-linUCB. Перевірку їх ефективності

проведемо на прикладі завдання Yahoo! за рекомендацією новин користувачам. При цьому використаємо метрика CTR (відношення числа кліків до показів). Результати наведено на рисунках 2.2, 2.3. При цьому під час роботи алгоритмів вважалося, що спочатку всі об'єкти та користувачі є новими для системи, оскільки до старту алгоритмів не використовувалась інформація про історію показів. Вся робота методів заснована на статистичних даних та ознакових уявленнях об'єктів та користувачів (тільки для контекстних алгоритмів).

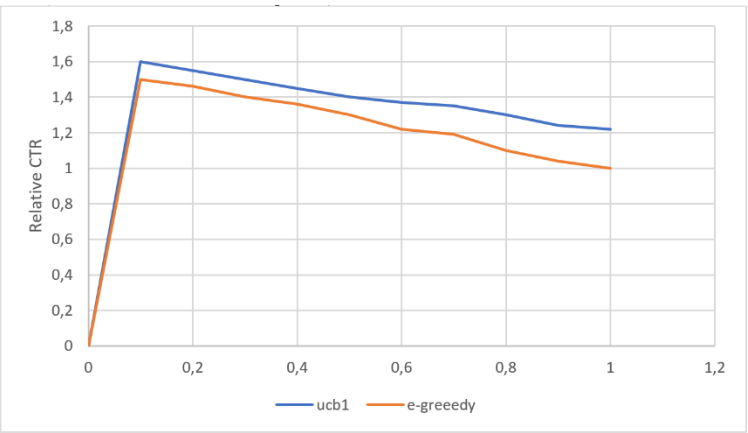


Рисунок 2.2 — Графік ефективності безконтекстних алгоритмів

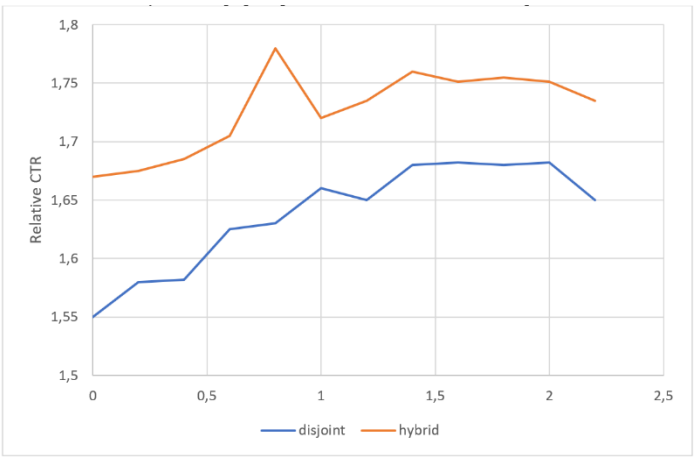


Рисунок 2.3 — Графік ефективності контекстних алгоритмів

Виявлено, що «бандитські алгоритми» досить добре справляються із проблемою холодного старту, що демонструють наведені графіки. Також варто зауважити, що ці методи працюють добре і для старих об'єктів. Рекомендаційна система буде відокремлена в окремий сервіс та мати ендпоінти для її налаштування та отримання рекомендацій по закупівлі.

2.4 Розробка клієнтської частини

Для розробки клієнтської частини застосунку існує два основних підходи: товстий клієнт та тонкий клієнт. Розглянемо їх переваги та недоліки. Товстий клієнт — це програмний засіб, який надає більш повний функціонал, та є частково або повністю незалежним від стану мережі та серверної частини. Найчастіше при такому архітектурному стилі серверна частина лише частково або повністю бере на себе роль бази даних, задачі по обробці отриманої інформації виконує власне товстий клієнт.

Переваги:

— кількість і повнота функціоналу;

— можливість роботи з багатьма користувачами;

— продовжує функціонувати при втраті з’єднання з сервером;

— швидкість роботи (пов’язана з апаратною частиною).

Недостатки:

— великий розмір програмного комплексу;

— прив’язка програмного засобу до архітектури платформи, для якої був написаний;

— непостійність віддаленого доступу до інформації;

— трудоємкий процес налаштування;

— складність оновлення програмної частини, в результаті ризик втратити актуальнсть;

— неінкапсульована бізнес-логіка програми.

Тонкий клієнт в архітектурі клієнт-сервер або термінал - це комп'ютер або клієнтська програма, яка передає всі або більшість своїх завдань обробки даних на серверний час. Цей термін також може позначати однорангові вузли, які використовують інші вузли мережі як сервери.

Визначення «тонкий клієнт» можна застосувати до відносно великої групи рішень, апартних пристроїв і програмних додатків, які мають спільну здатність працювати в режимі терміналу. Через це для повноцінної роботи тонкого клієнта необхідно мати сервер-термінал. У цьому полягає відмінність

між тонким клієнтом і товстим клієнтом, що обробляє інформацію без участі серверу, і цей сервер в основному використовується для збереження інформації [15].

Крім описаного прикладу, окремо виділяються апаратні тонкі клієнти. Цей клієнт являє собою особливий девайс, який повністю відрізняється від персонального комп'ютера. В апаратних тонких клієнтах не завжди наявні пристрої для збереження інформації, вони використовують вузьконаправлену операційну систему для встановлення зв'язку з термінальним сервером, мають повністю пасивну або максимально спрощену систему терморегуляції.

Веб-додаток — це програмний застосунок, що часто використовується при клієнт-серверній взаємодії в якості клієнта та використовує браузер інтерфейс для взаємодії з сервером. Бізнес-логіка веб-додатку розподілена між сервером і клієнтом, інформація переважно зберігаються на сервері, а обмін відбувається за допомогою Інтернет-з'єднання. Вагомою перевагою такого підходу є повна незалежність клієнта від апаратної платформи та операційної системи [16].

Важливою особливістю використання веб-програм, які використовують можливості відображення інформації за допомогою функціональності браузера є незалежність додатку від операційної системи клієнта. Тому немає потреби писати та одаптувати програмний код для роботи на операційних системах сімейства Microsoft Windows, Mac OS X, Linux та інших, пишеться та використовується одна і та ж програма на різних апаратних та програмних рішеннях.

Для розробки програмного рішення обрано варіант тонкого клієнта у вигляді веб-сайту, враховуючи можливі варіанти, їх плюси і мінуси. Розроблюваний клієнт має дублювати всі можливості серверної частини та надавати їх користувачу в зрозумілому форматі.

3 РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ СИСТЕМИ СКЛАДСЬКОГО ОБЛІКУ

3.1 Вибір інструментарію для реалізації продукту

Від вибору мови програмування та засобів формування модулів програми залежить продуктивність роботи створеного програмного продукту. В рамках даного розділу буде виконано вибір мов програтування, фреймворків та додаткоких засобів для створення програми системи складського обліку.

3.1.1 Мова C# та ASP.NET Core

Оскільки в веб-сервісі написання логіки обробки запиту та складних алгоритмів є більш пріорітетним, ніж швидкодія, написання серверної частини додатку буде виконуватись високорівневою мовою програмування. Для написання сервісів такого спрямування найчастіше використовуються C# (ASP.NET Core), Java (Spring Framework), Python (Django) та JavaScript (Node.js) [17].

Django (Джанго) — високорівневий відкритий Python-фреймворк (програмний каркас) для розробки веб-систем. Сайт на Django будується з однієї або декількох частин, які рекомендується робити модуль-ними. Це одна з істотних архітектурних відмінностей цього фреймворку від деяких інших (наприклад Ruby on Rails). Архітектура Django подібна на «Модель-Вигляд-Контролер» (MVC). Однак, те що нази-вається «контролером» в класичній моделі MVC, в Django називається «вигляд» (англ. view), а те, що ма-ло б бути «виглядом», називається «шаблон» (англ. template). Таким чином, MVC розробники Django на-зивають MTV («Модель-Шаблон-Вигляд») [18].

Node.js — платформа з відкритим кодом для виконання високопродуктивних мережевих застосун-ків, написаних мовою JavaScript. Node.js має наступні властивості: асинхронна одно-нитева модель ви-конання запитів, неблокуючий ввід/вивід, система модулів CommonJS, рушій JavaScript Google V8. Для керування модулями використовується пакетний менеджер npm (node package manager).

Spring Framework — це програмний каркас (фреймворк) з відкритим кодом та контейнери з підтримкою інверсії управління для платформи Java. Основні особливості Spring Framework можуть бути використані будь-яким додатком Java, але є розширення для створення веб-додатків на платформі Java EE. Незважаючи на це, Spring Framework не нав'язує якоїсь конкретної моделі програмування, Spring Framework став популярним в спільноті Java як альтернатива, або навіть доповнення моделі Enterprise JavaBean (EJB) [19].

ASP.NET Core представляє технологію для створення веб-додатків на платформі .NET, яку розвиває компанія Microsoft. Як мови програмування для розробки програм на ASP.NET Core використовуються C# і F#.

Історія ASP.NET фактично почалася з виходом першої версії. В той же час спочатку ASP.NET була націлена на роботу виключно в Windows на веб-сервері IIS (хоча завдяки проекту Mono програми на ASP.NET можна було запускати і на Linux).

Однак 2014 ознаменував великі зміни, фактично революцію в розвитку платформи: компанія Microsoft взяла курс на розвитку ASP.NET як кросплатформної технології, яка розвивається як opensource-проект. Даний розвиток платформи надалі отримав назву ASP.NET Core, що як її офіційно називають Microsoft до цих пір. Перший реліз оновленої платформи побачив світ у червні 2016 року. Тепер вона почала працювати не тільки на Windows, а й на MacOS та Linux. Вона стала легковажнішою, модульнішою, її стало простіше конфігурувати, загалом, вона стала більше відповідати вимогам поточного часу.

Поточна версія ASP.NET Core, яка власне і буде використана для розробки, вийшла разом із релізом .NET 6 у листопаді 2021 року.

ASP.NET Core тепер повністю є opensource-фреймворком. Усі вихідні файли фреймворку доступні на github у репозиторії [20].

ASP.NET Core має наступні особливості:

— ASP.NET Core працює поверх платформи .NET і, таким чином, дозволяє використовувати весь її функціонал;

— як мови розробки застосовуються мови програмування, що підтримуються платформою .NET. Офіційно вбудована підтримка для проектів ASP.NET Core має мови C# і F#;

— ASP.NET Core представляє крос-платформний фреймворк, додатки на якому можна розгорнути на всіх основних популярних операційних системах: Windows, Mac OS, Linux. І таким чином, за допомогою ASP.NET Core ми можемо як створювати крос-платформні програми на Windows, на Linux та Mac OS, так і запускати на цих ОС;

— завдяки модульності фреймворку, всі необхідні компоненти веб-програми можуть завантажуватися як окремі модулі через пакетний менеджер Nuget;

— підтримка роботи з більшістю поширених систем баз даних: MS SQL Server, MySQL, Postgres, MongoDB;

— ASP.NET Core характеризується розширюваністю. Фреймворк побудований із набору щодо незалежних компонентів. І ми можемо або використовувати вбудовану реалізацію цих компонентів, або розширити їх за допомогою механізму спадкування, або створити і застосовувати свої компоненти зі своїм функціоналом;

— багатий інструментарій для розробки програм. Як інструментарій розробки ми можемо використовувати таке середовище розробки з багатим функціоналом як Visual Studio від компанії Microsoft [21].

Для розробки серверної частини програмного продукту системи складського обліку найдоречнішим буде використання мови програмування C# та фреймворку ASP.NET Core. Також таке рішення є доречним з точки зору майбутнього масштабування системи, оскільки на базі обраної платформи побудовані такі системи суміжної направленості, як K2 Blackpearl або Orchard Core, а отже є й велика кількість кваліфікованих спеціалістів.

3.1.2 Мова Javascript та React

Сьогоднішній світ веб-сайтів важко уявити без JavaScript. JavaScript це те, що робить живими веб-сторінки, які ми щодня переглядаємо у своєму веб-браузері.

JavaScript був створений в 1995 році в компанії Netscape розробником Брендоном Айком (Brendon Eich) як мову сценаріїв у браузері Netscape Navigator 2. Спочатку мова називалася LiveScript, але на хвилі популярності на той момент іншої мови Java LiveScript був перейменований на JavaScript. Однак даний момент досі іноді призводить до деякої плутанини: деякі розробники-початківці вважають, що Java і JavaScript мало не один і той же мову. Ні, це абсолютно дві різні мови, і вони пов'язані лише за назвою.

Спочатку JavaScript володів досить невеликими можливостями. Його ціль полягала лише в тому, щоб додати трохи поведінки на веб-сторінку. Наприклад, обробити натискання кнопок на веб-сторінці, зробити якісь інші дії, пов'язані перш за все з елементами керування [22].

Проте розвиток веб-середовища, поява HTML5 та технології Node.js відкрило перед JavaScript набагато більші горизонти. Наразі JavaScript продовжує використовуватись для створення веб-сайтів, тільки тепер він надає набагато більше можливостей.

Також він застосовується як мова серверної сторони. Тобто якщо раніше JavaScript застосовувався тільки на веб-сторінці, а на стороні сервера нам треба було використовувати такі технології, як PHP, ASP.NET, Ruby, Java, то завдяки Node.js ми можемо обробляти всі запити до сервера також за допомогою JavaScript [23, 24].

Останнім часом переживає бум сфера мобільної розробки. І JavaScript знову ж таки не залишається осторонь: збільшення потужності пристроїв і повсюдне поширення стандарту HTML5 призвело до того, що для створення програм для смартфонів, планшетів та настільних комп'ютерів ми також можемо використовувати JavaScript. Тобто JavaScript вже переступив межі веб-браузера, які були окреслені при його створенні.

І що взагалі раніше здавалося фантастикою, але сьогодні стало реальністю - javascript може використовуватися для напряму розробки, що набирає популярність, для IoT (Internet of Things або Інтернет речей). Тобто JavaScript можна використовувати для програмування різних "розумних" пристроїв, які взаємодіють з інтернетом.

Таким чином, ви можете зустріти застосування JavaScript практично всюди. Сьогодні це справді одна з найпопулярніших мов програмування, і його популярність ще зростатиме [25].

Із самого початку існувало кілька веб-браузерів (Netscape, Internet Explorer), які надавали різні реалізації мови. І щоб звести різні реалізації до загального стрижня та стандартизувати мову під керівництвом організації ECMA було розроблено стандарт ECMAScript. В принципі самі терміни JavaScript і ECMAScript є багато в чому взаємозамінними і відносяться до однієї мови.

На сьогодні ECMA було розроблено кілька стандартів мови, що відображають його розвиток. Останнім часом майже щороку виходить новий стандарт. На даний момент останнім прийнятим стандартом є ECMAScript 2021, схвалений 22 червня 2021 року. Проте реалізація стандартів у браузерах займає тривалий час. Одні браузери швидше реалізують нові стандарти, інші повільніше. Крім того, є великий пласт старих версій браузерів, якими прості користувачі продовжують користуватися і які можуть не підтримувати нововведення останніх стандартів. І це треба враховувати під час розробки програм на JavaScript. У цьому ж посібнику будуть розглядатися переважно ті можливості JavaScript, які підтримуються всіма найпоширенішими сучасними браузерами [26].

JavaScript є мовою, що інтерпретується. Це означає, що код JavaScript виконується за допомогою інтерпретатора. Інтерпретатор отримує інструкції мови JavaScript, визначені на веб-сторінці, виконує їх (або інтерпретує).

Для прискорення процесу розробки та полегшення масштабування проекту в перспективі необхідно будувати клієнтську частину на базі

Javascript-фреймворку. Найбільшу долю сучасних програм розробляють на React, Angular або Vue.js. Розглянемо особливості кожного з них.

Angular представляє фреймворк від компанії Google для створення клієнтських програм. Насамперед він націлений розробку SPA-решений (Single Page Application), тобто односторінкових додатків. У цьому плані Angular є спадкоємцем іншого фреймворку AngularJS. У той же час Angular це не нова версія AngularJS, а новий фреймворк [27].

Angular надає таку функціональність як двостороннє зв'язування, що дозволяє динамічно змінювати дані в одному місці інтерфейсу при зміні даних моделі в іншому, шаблони, маршрутизація і так далі.

Однією з ключових особливостей Angular є те, що він використовує мову програмування TypeScript. Тому перед початком роботи рекомендується ознайомитись з основами даної мови, про які можна прочитати тут.

Але ми не обмежені мовою TypeScript. За бажанням можемо писати програми на Angular за допомогою таких мов як Dart або JavaScript. Проте TypeScript є основною мовою для Angular.

Остання версія Angular – Angular 13 вийшла у листопаді 2021 року.

Vue.js представляє сучасний прогресивний фреймворк, написаний мовою JavaScript та призначений для створення веб-додатків клієнтського рівня. Основна сфера застосування даного фреймворку - це створення та організація інтерфейсу користувача.

Перший реліз фреймворку відбувся у лютому 2014 року. Його творцем є Еван Ю (Evan You), який до цього працював у Google над AngularJS. З того часу фреймфорк динамічно розвивається, його поточною версією є версія 2.3.

Vue.js має досить невеликий розмір - не більше 20 кБ, і при цьому має гарну продуктивність у порівнянні з такими фреймворками як Angular або React. Тому не дивно, що цей фреймворк останнім часом набирає обертів і стає дедалі популярнішим [25].

Одним із ключових моментів у роботі Vue.js є віртуальний DOM. Структура веб-сторінки зазвичай описується за допомогою DOM (Document Object Model), яка представляє організацію елементів html на сторінці. Для

взаємодії з DOM (додавання, зміни, видалення HTML-елементів) застосовується JavaScript. Але коли ми намагаємося маніпулювати html-елементами за допомогою JavaScript, ми можемо зіткнутися зі зниженням продуктивності, особливо при зміні великої кількості елементів. А операції над елементами можуть зайняти деякий час, що неминуче позначиться на досвіді користувача. Однак якби ми працювали з коду js з об'єктами JavaScript, то операції робилися б швидше.

Для цього Vue.js використовує віртуальний DOM. Віртуальний DOM є легковажною копією звичайного DOM. Якщо програмі потрібно дізнатися інформацію про стан елементів, відбувається звернення до віртуального DOM. Якщо дані, які використовуються у програмі Vue.js, змінюються, то зміни спочатку вносяться до віртуального DOM. Потім Vue вибирає мінімальний набір компонентів, для яких треба виконати зміни на веб-сторінці, щоб реальний DOM відповідав віртуальному. Завдяки віртуальному DOM підвищується продуктивність програми.

Vue.js підтримується всіма браузерами, сумісними з ECMAScript 5. На даний момент це все сучасні браузери, у тому числі IE11.

React - це бібліотека JavaScript, яка використовується для створення інтерфейсу користувача. React було створено компанією Facebook, а перший реліз бібліотеки побачив світ у березні 2013 року. Поточною версією на даний момент (жовтень 2020 року) є версія React v17.0.

Спочатку React призначався для Інтернету, для створення сайтів, але потім з'явилася платформа React Native, яка вже призначалася для мобільних пристроїв [27].

React представляє ідеальний інструмент для створення масштабованих веб-додатків (в даному випадку йдеться про фронтенд), особливо в тих ситуаціях, коли програма представляє SPA (односторінковий додаток).

React відносно простий у освоєнні, має зрозумілий та лаконічний синтаксис.

Вся структура веб-сторінки може бути представлена за допомогою DOM (Document Object Model) – організація елементів html, якими ми можемо

маніпулювати, змінювати, видаляти чи додавати нові. Для взаємодії з DOM використовується мова JavaScript. Однак, коли ми намагаємося маніпулювати html-елементами за допомогою JavaScript, ми можемо зіткнутися зі зниженням продуктивності, особливо при зміні великої кількості елементів. А операції над елементами можуть зайняти деякий час, що неминуче позначиться на досвіді користувача. Однак якби ми працювали з коду js з об'єктами JavaScript, то операції робилися б швидше.

Для вирішення проблеми продуктивності якраз і виникла концепція віртуального DOM.

Віртуальний DOM є легковажною копією звичайного DOM. І відмінністю React є те, що дана бібліотека працює саме з віртуальним DOM, а не звичайним [27].

Якщо програмі потрібно дізнатися інформацію про стан елементів, відбувається звернення до віртуального DOM.

Якщо потрібно змінити елементи веб-сторінки, то зміни спочатку вносяться до віртуального DOM. Згодом новий стан віртуального DOM порівнюється з поточним станом. І якщо ці стани різняться, то React знаходить мінімальну кількість маніпуляцій, які потрібні до оновлення реального DOM до нового стану і виробляє їх.

У результаті така схема взаємодії з елементами веб-сторінки працює набагато швидше і ефективніше, ніж якщо ми працювали з JavaScript з DOM безпосередньо [27].

З описаних фреймворків найфункціональнішим є Angular, але одночасно з цим і найменш ефективним. Vue.js пропонує високу продуктивність роботи, але має достатньо обмежений функціонал. Найдоцільнішим варіантом є використання React, оскільки він поєднує в собі достатньо високу продуктивність та широкий функціонал, підтримується компанією Facebook та має найбільшу долю на ринку, а в результаті і велику кількість спеціалістів.

3.1.3 ICP VisualStudio

Для розробки серверної частини програми було обрано інтегроване середовище розробки Visual Studio 2019 Community. Visual Studio надає можливість редагувати вихідний код, підтримує технологію автодоповнення коду IntelliSense, можливість просунутого рефакторингу коду. Включені в ICP інструменти дозволяють відлагоджувати програму на рівні вихідного коду, а також відладчик коду машинного рівня, редагувати візуальні форми за допомогою інтуітивно зрозумілого графічного інтерфейсу, редагувати веб-форми, виконувати дизайн архітектури класів та таблиць в БД. Visual Studio надає можливість самостійно розробляти та використовувати вже готові плагіни для розширення функціональності майже на всіх рівнях [23].

Альтернативою є продукт Rider компанії JetBrains, який розповсюджується виключно методом платної підписки, а тому в рамках даної роботи розглядатись не буде.

3.1.4 РПК Visual Studio Code

Для розробки клієнтської частини програми буде доцільно використати високоефективний редактор коду з можливістю встановлення плагінів для полегшення розробки. Visual Studio Code – це редактор вихідного коду, який має багатомовний інтерфейс користувача та підтримує ряд мов програмування, підсвічування синтаксису, IntelliSense, рефакторинг, налагодження, навігацію за кодом, підтримку Git та інші можливості. Багато можливостей Visual Studio Code недоступні через графічний інтерфейс, часто вони використовуються через палітру команд або JSON-файли (наприклад, налаштування користувача). Палітра команд є подібністю командного рядка, що викликається поєднанням клавіш.

VS Code також дозволяє замінювати кодову сторінку за збереження документа, символи перекладу рядка та мову програмування поточного документа.

Також VS Code підтримує редагування та виконання файлів типу "Блокнот Jupyter" (Jupyter Notebook (англ.)) безпосередньо "з коробки" без встановлення зовнішнього модуля в режимі візуального редагування та в режимі редагування вихідного коду.

На березень 2019 року за допомогою вбудованого в продукт інтерфейсу користувача можна завантажити і встановити кілька тисяч розширень тільки в категорії «programming languages» (мови програмування).

Також розширення дозволяють отримати більш зручний доступ до програм, таких як Docker, Git та інші. У розширеннях можна знайти лінтери коду, теми для редактора та підтримку синтаксису окремих мов [21].

Для роботи з React встановимо в Visual Studio Code необхідний набір розширень.

3.2 Розробка програми

Створимо за допомогою ICP Visual Studio 2019 проект серверної частини програмного застосунку. За допомогою вбудованих засобів ICP використаємо шаблон «Веб застосунок ASP.NET» в якості каркасу (рисунок 3.1) та виконаємо його початкове налаштування.

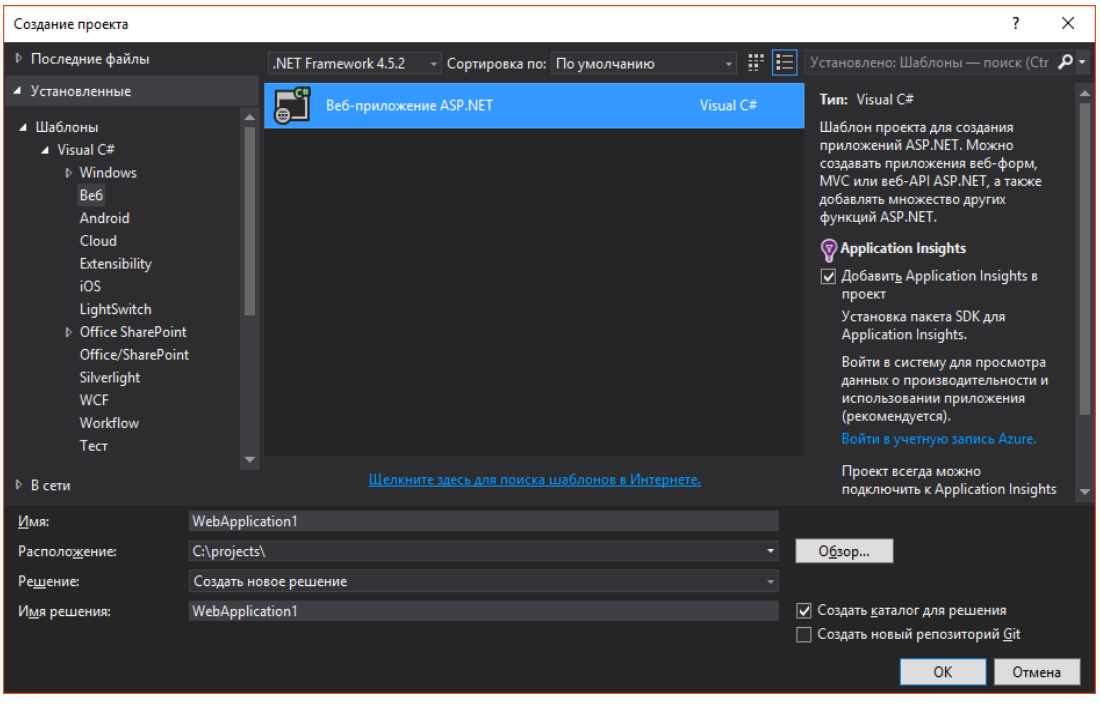


Рисунок 3.1 — Вибір шаблону при створенні проекту в Visual Studio

Структура створеного проекту включає в себе наступні вузли:

— /App_Data (в цій директорії містяться закриті дані, наприклад XML-файли або файли БД, якщо присутнє використання SQL Server Express, SQLite або подібні бази на основі файлів);

— /App_Start (в цій директорії міститься перелік базових налаштувань конфігурації, таких як визначення маршрутизації, фільтрів, пакетів вмісту);

— /Content (в цій директорії міститься статичний вміст, наприклад CSS-таблиці стилів, файли зображень);

— /Controllers (в цій директорії міститься опис контролерів);

— /Models (в цій директорії міститься опис класів моделей представлень і моделей, зякими оперує програма);

— /Scripts (в цій директорії містяться файли JavaScript, що застосовуються в проекті);

— /Views (в цій директорії містяться представлення і часткові представлення);

— /Views/Shared (в цій директорії містяться компонування та представлення, які є загальними для проекту або його частини);

— /Views/Web.config (конфігураційний файл, який містить налаштування обробки представлень);

— /Global.asax (це глобальний клас проету, файл відокремленого коду (Global.asax.cs) містить інформацію про маршрутизацію, програмний код, що виконується в момент старту або завершення роботи додатку, глобальну обробку виключень);

— /Web.config (конфігураційний файл програмного додатку).

Інформація, з якою працює застосунок, зберігається на віддаленому сервері за допомогою реляційної бази даних. Оскільки для побудови бази даних Code First, необхідно задати моделі даних та їх атрибути програмним кодом. Повна схема бази даних, у відповідності до якої розробляються моделі даних, наведена у Додатку Б.

Визначимо модель для номенклатури товару. Оскільки в подальшому для зберігання інформації буде застосовано БД, в моделі має бути описано ідентифікатор цілочисельного типу, цей атрибут буде використовуватись як первинний ключ. Також визначимо атрибут імені номенклатури рядкового типу. Для використання певних обмежень при записі даних до змінних можна застосовувати анотації, які записуються в квадратних дужках перед змінною. Для прикладу, анотація [Required] робить поле обов'язковим для запису:

```
public class Item {
public int Id { get; set; }
[Required]
[StringLength(100)]
public string Name { get; set; }}
```

Для збереження і обробки інформації про контрагента задамо модель Partner.

Для використання та маніпуляцій з інформацією в БД застосуємо Entity Framework Core, який за замовчуванням налаштований в застосунку на ASP.NET Core. Для створення та редагування даних у базі використовуються так звані «міграції», файли яких зберігаються у папці /Migrations. Для роботи з ними скористаємось системою управління пакетами NuGet та наявним в ICP плагіном Package Manager Console, що надає можливість користуватись NuGet PowerShell для знаходження, додавання та актуалізації пакетів.

Для взаємодії з БД необхідно створити ініціюючу міграцію. Викличемо в терміналі ICP наступну команду:

```
add-migration InitialModels
```

Після цього запишемо зміни до БД:

```
update-database
```

У класі ApplicationDbContext необхідно задати атрибути моделей, які будуть використані в БД.

Контролери задаються в директорії Controllers за допомогою вибору пункту контекстного меню Додати/Контролер… (рисунок 3.2). Для

контролерів загальноприйнятим є наступне іменування: <НазваУМножині>Controller.

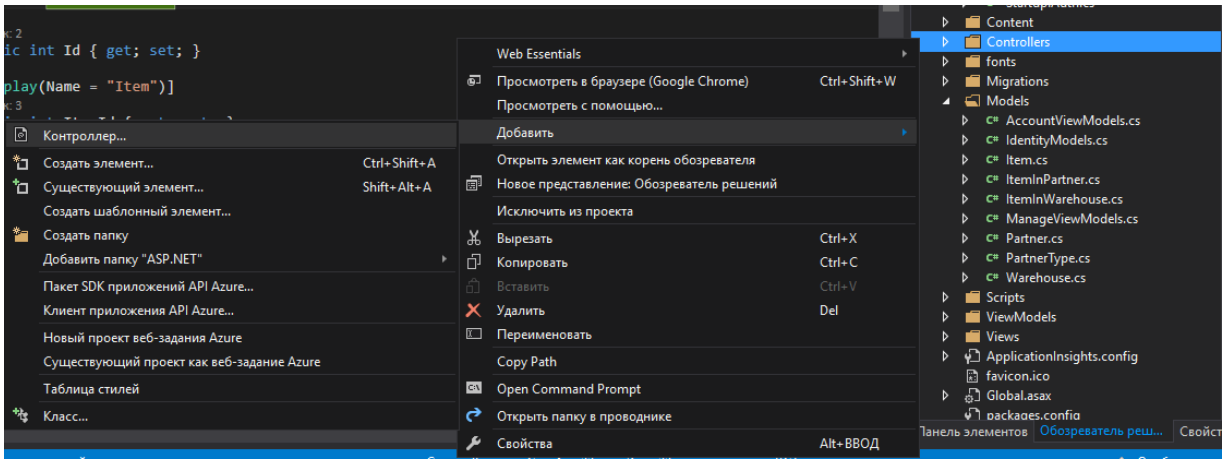


Рисунок 3.2 — Додавання нового контролеру

Визначимо контролер для операцій над товарами — ItemsController. Налаштуємо доступ до БД в класі ApplicationDbContext, виконаємо його ініціалізацію в конструкторі та перепишемо метод Dispose() для закінчення з'єднання з БД:

```
private ApplicationDbContext _context;
public ItemsController(){
_context = new ApplicationDbContext();}
protected override void Dispose(bool disposing){
_context.Dispose();}
```

Для відображення списку товарів опишемо активність Index, яка слугує для передачі у представлення набору об'єктів номенклатур, які було отримано з БД:

```
public ViewResult Index(){
/*…*/
return View(items);}
```

Опис екземпляру товару визначимо в активності Details, в якості атрибуту вхідних даних буде отримуватись унікальний ідентифікатор номенклатури. Використовуючи функцію SingleOrDefault() прочитаємо в базі інформацію про конкретну одиницю товару:

```
public ActionResult Details(int id){
```

```
/*…*/
return View(item);}
```

Для передачі в необхідної інформації створимо модель ItemViewModel з атрибутами, що зберігають номенклатуру та заголовок. Для створення нової номенклатури визначимо активність New:

```
public ViewResult New(){
/* … */
return View("ItemForm", viewModel);}
```

Активність Edit надає можливість корегувати вже існуючі номенклатури товарів:

```
public ActionResult Edit(int id){
/*…*/
return View("ItemForm", viewModel);}
```

Після редагування вже наявної або створення нової номенклатури необхідно внести актуальну інформацію в БД. Визначимо активність Save. Після внесених коректив записуємо інформацію в БД за допомогою функції SaveChanges(), а також виконуємо переадресацію на сторінку зі списком номенклатур. До метода Edit застосуємо атрибут HttpPost:

```
[HttpPost]
public ActionResult Save(Item item){
/*…*/
return RedirectToAction("Index", "Items");}
```

Для видалення запису про номенклатуру з БД визначимо активність Delete. Метод Remove() видаляє обраний запис. Також після виконаних дій необхідно зберегти зміни в БД та виконати переадресацію на попередню сторінку:

```
public ActionResult Delete(int id){
/*…*/
return RedirectToAction("Index", "Items");}
```

Повний лістинг контролерів наведено у Додатку В.

По аналогії з описаними вище кроками створимо та опишемо сервіс для роботи рекомандаційної системи. Повний лістинг сервісу наведено у Додатку В.

Створимо проект клієнтської частини системи складського обліку. Після створення директорії клієнтської частини програми проведемо ініціалізацію проекту для подальшого використання Node Packet Manager. Для цього викличемо наступну команду в директорії client (прапорець -y для пропуску всіх питань):

npm init -y

Після цього в директорії було створено файл package.json.

Webpack – це модульний складальник, який дозволяє об'єднувати ресурси і бібліотеки, необхідні для проекту, в один файл. Для його встановлення використаємо наступну команду:

npm install webpack webpack-cli --save-dev.

Після цього webpack та інтерфейс доступу до нього через командний рядок були додані в проект в якості залежностей dev.

React - JavaScript-бібліотека з відкритим вихідним кодом для розробки призначених для користувача інтерфейсів. React розробляється і підтримується Facebook, Instagram і співтовариством окремих розробників і корпорацій. React може використовуватися для розробки односторінкових і мобільних додатків. Його мета - надати високу швидкість, простоту і масштабованість. Додамо дану бібліотеку в проект:

npm install react react-dom –save

Babel - це транспайлер, який транслює код сучасного стандарту Javascript (ES2015) на більш пізній. ECMA International випускає оновлення мови Javascript щороку. У зв'язку з цим у розробників з'являються нові можливості: більш короткий синтаксис, стрілкові функції, проміси і.т.д. Відповідно, не всі сучасні браузери можуть або хочуть йти в ногу з цими змінами. А ще додаємо старі браузери, які більше ніколи не буде оновлено. Але їх все одно поки використовують, наприклад Internet Explorer. Ось тут і потрібен Babel. Він допомагає не чекати оновлень браузерів, а відразу

використовувати сучасні стандарти Javascript. Встановимо babel-core, babel-loader, babel-preset-env, babel-preset-react в якості залежності dev:

npm install @babel/core babel-loader @babel/preset-env @babel/preset-react --save-dev

Після встановлення всіх необхідних для розробки компонентів заповнимо файли конфігурацій та базову структуру React-проекту.

Для управління станом веб-додатку використаємо бібліотеку Redux. Redux – бібліотека для JavaScript з відкритим кодом, призначена для управління станом додатку. Найчастіше використовується в зв'язці з React або Angular для розробки клієнтської частини. Містить ряд інструментів, що дозволяють значно спростити передачу даних сховища через контекст. Для встановлення redux в проект в якості залежності потрібно викличемо наступну команду:

npm install redux, react-redux, redux-thunk

Основні елементи для роботи зі сховищем:

— state — власне сховище;

— thunk — асинхронний аналог action;

— reducer — обробник «сигналів» (actions), що змінює структуру сховища у відповідності до отриманого action;

— dispatch — функція, яка надсилає action до глобального обробника, який у свою чергу передає action до відповідного reducer.

— action — простий об’єкт, за допомогою якого можна просигналізувати, що потрібно змінити стан додатку через певні зміни.

Спроектуємо структуру директорій, призначену для роботи сховища Redux. Сховище містить наступні елементи:

— store/actions — містить в собі опис сигналів;

— store/reducers — містить опис обробників сигналів;

— store/types.js — містить опис типів сигналів;

— store/store.js — містить в собі опис головного об’єкта сховища.

В React є власна система маршрутизації, що надає змогу порівнювати запити до додатку із визначеними компонентами. Головним елементом у

роботі маршрутизації є react-router, що містить в собі основний функціонал для роботи з нею. Проте, якщо необхідно підтримувати навігацію у браузері, тоді необхідно використовувати react-router-dom. Встановимо в проект необхідні залежності:

npm install react-router, react-router-dom

Розглянемо, яким чином можна спроектувати маршрутизацію по додатку.

Елемент Router містить усередині набір маршрутів, й коли до додатку надходить запит, Router співставляє запит із визначеними маршрутами. Якщо ж певний співпадає, тоді він береться для обробки запитів. Router приймає як аргумент об’єкт типу History, що являє собою історію навігації користувача по веб-сайту та дозволяє маніпулювати адресою користувача у залежності від події або його взаємодії з інтерфейсом (перенаправляти користувача на адресу вибірки пошуку, функціональність кнопки «назад»).

Елемент Switch надає змогу обрати перший маршрут, який співпадає і застосовувати тільки його, проте, без Switch, Router може використати для одного запиту кількох маршрутів, якщо вони усі відповідають правилам та стрічці запиту.

Кожен маршрут зазначений елементом Route, що містить кілька атрибутів для конфігурації.

— path — шаблон адреси, із якою для підбору компоненти буде співставлятись клієнтський URL;

— component — та компонента, яка буде відображена при обробці запиту за умови відповідності «path»;

— render — приймає функціональну компоненту, дозволяє передати об’єкт props для цієї компоненти;

— exact — вказує, що при наявності даного атрибути, маршрути будуть порівнюватись строго.

У фойлі index.js виклик створення головного компоненту огорнемо за допомогою роутера BrowserRouter:

<BrowserRouter>

```
            <Provider store={store}>
               <App />
            </Provider>
      </BrowserRouter>
```

Всередині головного компонента програми опишемо необхідні шляхи, такі як перехід на домашню сторінку, перехід в каталог книжок, перехід на сторінку пошуку та перехід на сторінку конкретної книжки з каталогу:

```
      <Switch>
                  <Route exact path="/">
                     <HomePage />
                  </Route>
                  <Route exact path="/library">
                     <CatalogPage />
                  </Route>
                  <Route exact path="/search/:searchString">
                     <SearchPage />
                  </Route>
                  <Route exact path="/library/:bookId">
                     <BookPage />
                  </Route>
                  <Route>
                     <NotFoundPage />
                  </Route>
      </Switch>
```

Останній пустий маршрут відповідає за виклик сторінки помилки. Оскільки пошук підходящого маршруту здійснюється зверху вниз, то він буде перехоплювати всі запити, що не підпадають під існуючі маршрути і направляти на компонент «NotFoundPage».

Для спрощення задання CSS-стилів при розробці та responsive-дизайну додатку використаємо CSS-бібліотеку стилів Bootstrap 5. Встановимо її за допомогою наступної команди менеджера пакетів:

npm install bootstrap

Імпортуємо стилі всередині файлу index.js:

import 'bootstrap/dist/css/bootstrap.min.css';

import 'bootstrap/dist/js/bootstrap.bundle.min';

Компоненти дозволяють розділити інтерфейс користувача на незалежні частини, придатні до повторного використання, і сприймати їх як такі, що функціонують окремо один від одного. Концептуально компоненти є подібними до функцій JavaScript. Вони приймають довільні вхідні дані (так звані “пропси”) і повертають React-елементи, що описують те, що повинно з’явитися на екрані. Повний код клієнтської частини наведено в додатку Г.

3.3 Перевірка роботи програми

Для виконання тестів розробленого API будем використовувати веб-браузер Google Chrome та утиліту Postman. Виконаємо налаштування Postman на отримання GET-запитів у з тілом у форматі JSON. Відправимо запит по ендпоінту http://localhost:4000/api/items. Отримуємо інформацію у JSON-форматі в наступному вигляді:

Status: 200 OK, Time: 68ms

[{ ”Id”: 5, ”Name”: ”Товар5” },

{ ”Id”: 6, ”Name”: ”Товар6” },

{ ”Id”: 7, ”Name”: ”Товар7” },

{ ”Id”: 8, ”Name”: ”Товар8” },

{ ”Id”: 9, ”Name”: ”Товар9” },

{ ”Id”: 10, ”Name”: ”Товар10” },

{ ”Id”: 11, ”Name”: ”Товар11” },

{ ”Id”: 12, ”Name”: ”Товар12” },

{ ”Id”: 13, ”Name”: ”Товар13” },

{ ”Id”: 14, ”Name”: ”Товар14” },

{ ”Id”: 15, ”Name”: ”Товар15” },

{ ”Id”: 16, ”Name”: ”Товар16” },

{ ”Id”: 17, ”Name”: ”Товар17” },

{ ”Id”: 18, ”Name”: ”Товар18” },

{ ”Id”: 19, ”Name”: ”Товар19” },

{ ”Id”: 20, ”Name”: ”Товар20” },

{ ”Id”: 21, ”Name”: ”Товар21” }]

Виконаємо GET-запит за ендпоінтом http://localhost:4000/api/items/7. Отримуємо наступну відповідь:

Status: 200 OK, Time: 68ms

{

“Id“: 7,

“Name“: “Товар7“

}

Виконаємо POST-запит за ендпоінтом http://localhost:4000/api/items з наступним тілом запиту у вигляді JSON:

{

“Name“: “Товар23“

}

Отримуємо наступну відповідь у вигляді JSON:

Status: 200 OK, Time: 112ms

{

“Id“: 23,

“Name“: “Товар23“

}

Виконаємо PUT-запит за ендпоінтом http://localhost:4000/api/items/23 з наступним тілом запиту у вигляді JSON:

{

“Name“: “Товар23_Відредаговано“

}

У відповідь отримуємо:

Status: 200 OK, Time: 68ms

{

“Id“: 23,

“Name“: “Товар23_Відредаговано“

}

Виконаємо запит DELETE за ендпоінтом http://localhost:4000/api/items/23:

Status: 204 No Content, Time: 78ms

Оскільки метод DeleteItem не повертає значення, перевіряємо наявність запису в базі даних GET-запитом, отримуємо наступну відповідь:

Status: 404 Not Found, Time: 26ms.

Виконана перевірка розробленого API показує його повну працездатність та ефективність.

Виконаємо тестування клієнтської частини програми та її взаємодії з сервером за допомогою відладки в ICP через запуск веб-браузеру Google Chrome.

Проведемо тестування візуального інтерфейсу на прикладі чотирьох основних взаємодій з сутністю товарів: читання (перегляд списку та конкретного об’єкту), запис (додавання нового екземпляру), редагування (зміна властивостей існуючого екземпляра), видалення (видалення конкретного екземпляру). Після використання кнопки «Товари» в верхньому меню інтерфейсу виконується перенаправлення по маршруту /Items, у робочій області сайту виводиться перелік номенклатур у вигляді таблиці з двома стовпцями – «Назва» та «Редагувати». Після натискання на кнопку «Додати товар» виконується перенаправлення по маршруту /Items/New.

В полях на формі вводимо необхідні значення та натискаємо на «Зберегти». Після цього виконується перенаправлення на попередню сторінку з актуальними даними та новою номенклатурою.

Після взаємодії з кнопкою «Редагувати» виконується переадресація за маршрутом /Items/Edit/22. Крім можливості збереження, відображається кнопка «Видалити». Після корекції полів на формі та збереження виконується переадресація по маршруту з переліком товарів. Після видалення виконується переадресація на попередню сторінку, з оновленою інформацією та без видаленого запису. Після натискання на назву товару виконується

переадресація за маршрутом /Items/Details/5.

В ході тестувань програма видає очікуваний результат, користувацький інтерфейс є лаконічним і інтуїтивно зрозумілим. Графічний матеріал з тестуванням клієнтської частини застосунку наведено у Додатку Д.

4 РОЗРАХУНОК ЕКОНОМІЧНОЇ ДОЦІЛЬНОСТІ СТВОРЕННЯ ПРОГРАМНОГО ЗАСОБУ ІНФОРМАЦІЙНОЇ СИСТЕМИ СКЛАДСЬКОГО ОБЛІКУ ТОВАРНО-МАТЕРІАЛЬНИХ ЦІННОСТЕЙ

Дослідження та впровадження в роботу нових технологій є витратними. Тому витрати на виробництво та реалізацію товарів необхідно постійно зменшувати, оскільки в перспективі кошти можуть принести прогрес для будь-якого виробництва. На основі економічних розрахунків [28] можна продемонструвати рентабельність та ефективність впровадження результатів досліджень у виробництво, тобто комерціалізація наукових досліджень. Дана магістерська кваліфікаційна робота відноситься до розряду прикладної науково-технічної роботи. Прогнозується виведення науково-технічної розробки на ринок із залученням потенційним інвестора. Дану послідовність приведено на рисунку 4.1.

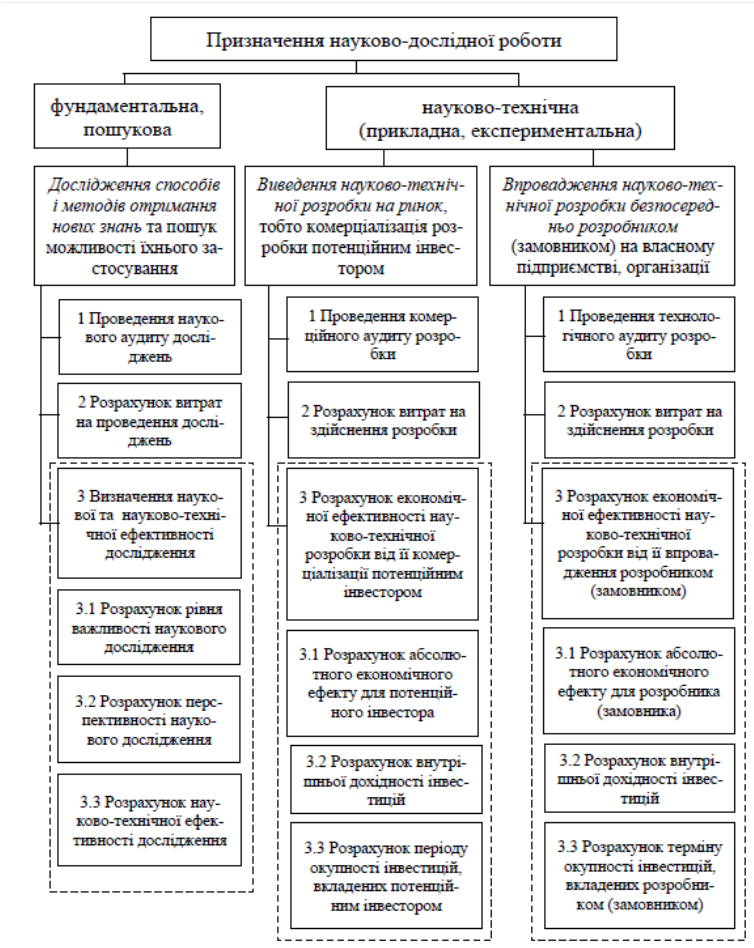


Рисунок 4.1 — Складові економічної частини магістерської кваліфікаційної роботи

Економічна частина цієї магістерської роботи буде поділена на такі елементи. Усі подальші економічні розрахунки будуть розглянуті у згаданих розділах економічної частини.

4.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою оцінки потенціалу комерційного розвитку є оцінка потенціалу комерційного розвитку, що випливає з науково-технічних досліджень. За результатами оцінки робляться висновки про напрямки (особливості) організації в майбутньому її впровадження з урахуванням встановленої оцінки. Комерційний потенціал інвестицій буде оцінюватись відповідно до дванадцяти критеріїв, наведених у таблиці 4.1.

Таблиця 4.1 — Оцінювання комерційного потенціалу розробки

Критерії оцінювання та бали (за 5-бальною шкалою)					
Критерій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Багато аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів

Продовження таблиці 4.1

5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
Практична здійсненність					
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні.	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси.Джерела фінансування є	Потрібні незначніфінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років.Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років.Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років.Термін окупності інвестицій менше 3-х років

Закінчення таблиці 4.1

12	Необхідно регламентні документи та велика кількість дозвільних документів на виробництво продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

На основі таблиці різні експерти, у нашому випадку викладачі кафедри ОТ визначають різні результати. Результати цієї оцінки комерційного потенціалу узагальнено у таблиці 4.2.

Таблиця 4.2 — Результати оцінювання комерційного потенціалу розробки

Критерії	Експерт (ПІБ, посада)		
	1 Черняк О.І., к.т.н., доц. кафедри ОТ	2 Семеренко В. П., к.т.н., доц. кафедри ОТ	3 Крупельницький Л.В., к.т.н., доц. кафедри ОТ
	Бали:		
1.Технічна здійсненність концепції	3	3	3
2.Ринкові переваги (наявність аналогів)	4	4	2
3. Ринкові переваги (ціна продукту)	3	3	3
4.Ринкові переваги (технічні властивості)	3	4	3
5. Ринкові переваги (експлуатаційні витрати)	3	2	3
6. Ринкові перспективи (розмір ринку)	4	4	4
7. Ринкові перспективи (конкуренція)	3	3	2
8. Практична здійсненність (наявність фахівців)	3	3	3
9. Практична здійсненність (наявність фінансів)	2	3	3

Закінчення таблиці 4.2

10.Практична здійсненність (необхідність нових матеріалів)	3	3	3
11. Практична здійсненність (термін реалізації)	3	3	4
12. Практична здійсненність (розробка документів)	3	3	3
Сума балів	$СБ_1 = 37$	$СБ_2 = 38$	$СБ_3 = 36$
Середньо-арифметична сума балів $СБ_С$	$СБ_С = \frac{\sum_1^3 СБ_i}{3} = \frac{37 + 38 + 36}{3} = 37$		

Відповідно до таблиці 4.2, а також відповідно до рекомендацій, наведених у таблиці 4.3, можна зробити висновок про рівень потенціалу комерційного розвитку [29].

Таблиця 4.3 — Рівні комерційного потенціалу розробки

Середньоарифметична сума балів $\overline{СБ}$, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 — 10	Низький
11 — 20	Нижче середнього
21 — 30	Середній
31 — 40	Вище середнього
41 — 48	Високий

З урахуванням середніх арифметичних балів $СБ_С$ = 37, які були визначені експертами, можна зробити висновок, що рівень комерційного потенціалу цієї розробки буде вище середнього.

Програмний ресурс BAS Бухгалтерія було використано для порівняння властивостей. Це програмне забезпечення має більш широкий спектр застосування та деяку кількість додаткового функціоналу.

Нова розробка, навпаки, є вузькоспрямованою і тому має більшу швидкість. Окрім того, інтерфейс не перевантажений зайвою інформацією, що значно полегшує роботу. Розповсюдження аналога відбувається лише в рамках

передплати, тоді як плата за використання програми, яка є продуктом розробки, є одноразовою.

Також, ще один аналог розробленого програмного забезпечення — МійСклад, який є повністю хмарною технологією. Дане програмне забезпечення є максимально простим для освоєння та використання, підтримує обробку замовлень, наявний мобільний застосунок, має розширену можливість роботи з кур'єрами.

Основний недолік – повна відсутність інтеграції з сторонніми сервісами та вивантаження даних. Також для корпоративних клієнтів може стати недоліком неможливість встановлення серверної частини програми в локальній мережі підприємства, лише в хмарі. Порівння розробки з її аналогами приведено в таблиці 4.4.

Таблиця 4.4 — Порівняння характеристик розробки із аналогом

Показники	Розробка	Аналог1	Аналог2
Функціонал	9	9	9
Швидкодія	8	7	8
Надійність	8	8	7
Метод розповсюдження	8	7	8
Інтерфейс, простота використання	8	6	8

Продукт буде просуватись за допомогою його презентації потенційним клієнтам. Додатково потенційного клієнта можна знайти за допомогою реклами в соціальних мережах, пошукових системах та багатьох інших джерелах Інтернету. Використовуючи аналітику цих сервісів, можна буде націлити рекламу на цільову групу захисників інформації.

Продукт може бути використаний як для підприємств великого, так і середнього та малого бізнесу. Продукт є універсальним в плані сфери направленості підприємства.

Новизна дослідження полягає у використанні в системі складського обліку рекомендаційної системи, яка дозволить менеджеру по закупівлі більш ефективно закупляти товар.

Виходячи з результатів цього порівняння, можна з упевненістю сказати, що новий дизайн є конкурентоспроможним, оскільки в деяких аспектах він переважає одного з найкращих аналогів на ринку. Даний рівень було досягнуто за рахунок покращення та/або розширення функціональних можливостей нової науково-технічної розробки порівняно з аналогічними розробками, існуючими в цей час на ринку.

4.3 Розрахунок витрат на здійснення науково-дослідної роботи

У магістерській роботі розглядається програмне забезпечення для розпізнавання осіб на основі зображення особи, тому значну частину витрат складають витрати на розробку, а не на виробництво та відтворення [29]. Відповідно, є певна специфіка розрахунків.

4.3.1 Витрати на оплату праці

Основна заробітна плата розробників, що працюють над проектом, визначена у формулі:

$$З_о = \sum_{i=1}^{k} \frac{М_{ni} \cdot t_i}{Т_р}, \quad (4.1)$$

де k — кількість посад дослідників, залучених до процесу досліджень;

$М_{ni}$ — місячний посадовий оклад конкретного дослідника, грн;

$Т_р$ — середня кількість робочих днів в місяці, $Т_р$ = 21…23 дні; обрано 22 дні;

t_i— кількість днів роботи конкретного дослідника, дн.

Над створенням розробки працював менеджер проекту та інженер програмного забезпечення, тому ми виконаємо для данх працівників усі необхідні розрахунки, та після чого вносимо їх до таблиці 4.5:

$$З_{о.к.} = \frac{16500 \cdot 8}{22} = 6000(грн),$$

$$З_{о.в.} = \frac{11000 \cdot 40}{22} = 20000\ (грн).$$

Таблиця 4.5 — Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	16500	750	8	6000
Старший інженер-програміст	11000	500	40	20000
Всього				26000

Витрати на основну заробітну плату робітників за відповідними найменуваннями робіт відсутні, тобто $З_р = 0$. Додаткова винагорода ($З_{дод.}$) усіх розробників та працівників, які брали участь у цьому етапі роботи, обчислюється як 10 ... 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод.} = (З_о + З_р) \cdot \frac{Н_{дод.}}{100\%}, \quad (4.2)$$

де $Н_{дод.}$ — норма нарахування додаткової заробітної плати.

$$З_{дод.к.} = \frac{10 \cdot 6000}{100} = 600\ (грн),$$

$$З_{дод.в.} = \frac{10 \cdot 20000}{100} = 2000\ (грн),$$

$$З_{дод} = З_{дод.к.} + З_{дод.в.} = 2600\ (грн).$$

4.3.2 Відрахування на соціальні заходи

Заробітна плата робітників відсутня, тому $З_р = 0$. Нарахування на заробітну плату дослідників та нарахування на заробітну плату працівників, які брали участь у цьому етапі роботи, розраховується як 22% від суми основної та додаткової заробітної плати дослідників і робітників за наступною формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%}, \quad (4.3)$$

де $Н_{зп}$ — норма нарахування на заробітну плату.

$$З_н = (26000 + 0 + 2600) \cdot \frac{22\%}{100\%} = 6292 \text{ (грн.)}$$

4.3.3 Сировина та матеріали

Витрати на матеріали (М), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за наступною формулою:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}, \quad (4.4)$$

де $Н_j$ — кількість матеріалу j-го виду, шт.;

n — кількість видів матеріалу.

$Ц_j$ — ціна матеріалу j-го виду, грн;

$К_j$ — коефіцієнт транспортних витрат, $К_j$ = (1,1…1,15), обираємо $К_j$ 1,15;

$В_j$ — маса відходів j-го найменування, кг;

$Ц_{вj}$ — вартість відходів j-го найменування, грн/кг.;

Результати розрахунків занесено до таблиці 4.6.

Таблиця 4.6 — Витрати на матеріали

Найменування комплектуючих	Ціна за 1 штуку, грн	Кількість матеріалу, штук	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Ручка	30,00	1	0,06	1,80	21,53
Пачка офісного папіру	176,00	1	0,5	2,50	45,25
Всього (з урахуванням транспортних витрат)					66,78

4.3.4 Розрахунок витрат на комплектуючі

Оскільки кінцевий продукт, який ми створюємо — це програмний інструмент, це не спричиняє жодних витрат на компоненти та $К_в = 0$.

4.3.5 Спецустаткування для наукових (експериментальних) робіт

Спецустаткування для проведення експериментальних робіт по створенню програмного продукту по розпізнаванню особи за зображенням її обличчя не має потреби залучати.

4.3.6 Програмне забезпечення для наукових (експериментальних) робіт

Програмне забезпечення для створення програмного продукту по розпізнаванню особи за зображенням її обличчя використовується таке, що є у вільному розповсюдженні, тому витрати на придбання такого забезпечення відсутні. Це мова програмування Python та програмні продукти із бібліотек із відкритим кодом OpenCViDLib.

4.3.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо можуть бути розраховані з використанням прямолінійного методу амортизації за формулою:

$$А_{обл} = \frac{Ц_б}{Т_в} \cdot \frac{t_{вик}}{12,} \quad (4.5)$$

Таблиця 4.7 — Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання, місяців.	Амортизаційні відрахування, грн
ЕОМ	11000	2	2	916,67
Приміщення	100000	20	2	833,33
Всього				1750

4.3.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію ($В_е$) розраховують за формулою:

$$В_е = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_е \cdot К_{впі}}{\eta_i}, \quad (4.6)$$

де W_{yi} — встановлена потужність обладнання на певному етапі розробки, кВт;

t_i — тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ — вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), $Ц_е = 4,62$ [25];

$К_{впі}$ — коефіцієнт, що враховує використання потужності, $К_{впі}$ <1; обираємо $К_{впі} = 0,7$;

η_i — коефіцієнт корисної дії обладнання, η_i<1; обираємо $\eta_i = 0,8$.

$$В_е = \sum_{i=1}^{1} \frac{0,1 \cdot 352 \cdot 4,62 \cdot 0,7}{0,8} = 142,30 \text{ (грн.)}$$

Проведені розрахунки необхідно звести до таблиці 4.8.

Таблиця 4.8 — Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
ЕОМ	0,1	352	142,30
Всього			142,30

4.3.9 Службові відрядження

Під час розробки програмного забезпечення відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, магістрів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень, не плануються.

4.3.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» не плануються, так як у цьому не має потреби.

4.3.11 Інші витрати

Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_в = \left(З_о + З_р\right) \cdot \frac{Н_{ів}}{100\%}, \qquad (4.7)$$

де $Н_{ів}$ — норма нарахування за статтею «Інші витрати».

$$I_{в.к.} = 6000 \cdot \frac{50\%}{100\%} = 3000 \text{ (грн.)},$$

$$I_{в.в.} = 20000 \cdot \frac{50\%}{100\%} = 10000 \text{ (грн.)},$$

$$I_в = I_{в.к.} + I_{в.в.} = 13000 \text{ (грн.)}.$$

4.3.12 Накладні (загальновиробничі) витрати

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (4.8)$$

де $Н_{нзв}$ — норма нарахування за статтею «Накладні (загальновиробничі) витрати».

Беремо норму нарахування 100%.

$$В_{нзв.в.} = 6000 \cdot \frac{100\%}{100\%} = 6000 \text{ (грн.)},$$

$$В_{нзв.в.} = 20000 \cdot \frac{100\%}{100\%} = 20000 \text{ (грн.)},$$

$$В_{нзв} = В_{нзв.к.} + В_{нзв.в.} = 26000 \text{ (грн.)}.$$

Витрати на проведення науково-дослідної роботи розраховуються як сума всіх попередніх статей витрат за формулою:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (4.9)$$

У нашому випадку:

$$З_р = 0, К_в = 0, В_{спец} = 0, В_{прг} = 0, В_{св} = 0, В_{сп} = 0, \text{тому отримаємо:}$$

$$В_{заг} = 26000 + 2600 + 6292 + 66{,}78 + 1750 + 142{,}30 + 13000 + 260 = 50111{,}08 \text{ (грн)}.$$

Загальні витрати ЗВ на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta}, \quad (4.10)$$

де η — коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи; обираємо η = 0,5.

$$ЗВ = \frac{50111,08}{0,5} = 100222,16 \text{ (грн)}.$$

4.4 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

Розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем.

Для всіх наведених випадків можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки [30], розраховується за формулою:

$$\Delta П_i = (\pm \Delta Ц_o \cdot N \cdot Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\vartheta}{100}\right) \qquad (4.11)$$

де $\pm\Delta Ц_o$— зміна основного якісного показника від впровадження результа-тів науково-технічної розробки в аналізованому році;

N — основний кількісний показник, який визначає величину попиту на аналогічні чи подібні розробки у році до впровадження результатів нової науково-технічної розробки;

$Ц_o$ — основний якісний показник, який визначає ціну реалізації нової науково-технічної розробки в аналізованому році, $Ц_o = Ц_б \pm \Delta Ц_o$;

$Ц_б$ — основний якісний показник, який визначає ціну реалізації існуючої (базової) науково-технічної розробки у році до впровадження результатів;

ΔN — зміна основного кількісного показника від впровадження результа-тів науково-технічної розробки в аналізованому році;

λ — коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2021 році ставка податку на додану вартість становить 20%, а коефіцієнт λ=0,8333;

ρ — коефіцієнт, який враховує рентабельність інноваційного продукту (послуги). Рекомендується брати ρ=0,2...0,5. Обраємо $\rho = 0{,}26$;

ϑ — ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2021 році ϑ=18%.

В результаті впровадження результатів наукових розробок поліпшується якість програмного забезпечення, що дозволяє подорожчати за його впровадження, а кількість потенційних користувачів ресурсу збільшиться — у перший рік — на 110 одиниць, на другий рік — ще на 450 одиниць, на третій рік — ще 550 штук.

Ми прогнозуємо щорічний приріст чистого прибутку компанії від впровадження результатів наукових розробок щодо вихідного стану. Збільшення чистого прибутку підприємства $\Delta П_i$ за перший рік складе:

$$\Delta П_1 = [1100 \cdot 0 + (3500 + 1100) \cdot 100] \cdot 0,8333 \cdot 0{,}3 \cdot \left(1 - \frac{18\%}{100\%}\right) = 94296{,}23 \text{ (грн)}.$$

Збільшення чистого прибутку компанії $\Delta П_i$ на другий рік (порівняно з базовим, тобто роком, що передує впровадженню результатів наукових досліджень) складе:

$$\Delta П_2 = [1100 \cdot 0 + (3500 + 1100) \cdot (100 + 200)] \cdot 0,8333 \cdot 0{,}3 \cdot \left(1 - \frac{18\%}{100\%}\right) = 282888{,}68 \text{ (грн)}.$$

Збільшення чистого прибутку підприємства $\Delta П_i$ на третій рік складе:

$$\Delta П_3 = [1100 \cdot 0 + (3500 + 1100) \cdot (100 + 200 + 400)] \cdot 0,8333 \cdot 0{,}3 \cdot \left(1 - \frac{18\%}{100\%}\right) = 660073{,}60 \text{ (грн)}.$$

Далі розраховують приведену вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (4.12)$$

де $\Delta П_i$— збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

Т — період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ = 0,05…0,15. Обираємо $\tau = 0{,}1$;

t— період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = \frac{94296{,}23}{(1+0{,}1)^1} + \frac{282888{,}68}{(1+0{,}1)^2} + \frac{660073{,}60}{(1+0{,}1)^3} = 85723{,}85 + 233792{,}30 + 495923{,}07$$
$$= 815439{,}22 \text{ (грн.)}$$

Далі розраховують величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} \cdot ЗВ, \quad (4.13)$$

де $k_{інв}$— коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}$=2...5, але може бути і більшим. Обираємо $k_{інв}$ = 2;

ЗВ — загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 \cdot 100222{,}16 = 200444{,}32 \text{ (грн.)}$$

Тоді абсолютний економічний ефект $Е_{абс}$ або чистий приведений дохіддля потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$Е_{абс} = ПП - PV \quad (4.14)$$

де ПП — приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, грн;

PV — теперішня вартість початкових інвестицій, грн.

$$Е_{абс} = 1277828{,}22 - 329231{,}12 = 948597{,}1 \text{ (грн.)}$$

Внутрішня економічна дохідність інвестицій $Е_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, розраховується за формулою:

$$Е_в = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV}} - 1, \quad (4.15)$$

де $Е_{абс}$ — абсолютний економічний ефект вкладених інвестицій, грн;

PV — теперішня вартість початкових інвестицій, грн;

$Т_ж$ — життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, роки.

$$Е_в = \sqrt[3]{1 + \frac{614994{,}90}{200444{,}32}} - 1 = 1{,}02$$

Далі визначають бар'єрну ставку дисконтування τ_{min}, тобто мінімальну внутрішню економічну дохідність інвестицій, нижче якої кошти у впровадження науково-технічної розробки та її комерціалізацію вкладатися не будуть [31].

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$визначається за формулою:

$$\tau_{\text{мін}} = d + f, \qquad (4.16)$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; в 2021 році в Україні d = 0,9...0,12. Обираємо $d = 0{,}11$;

f — показник, що характеризує ризикованість вкладення інвестицій; зазвичай величина f=0,05...0,5, але може бути і значно вищою. Обираємо $f = 0{,}2$;

$$\tau_{\text{мін}} = d + f = 0{,}10 + 0{,}2 = 0{,}30\%$$

Величина $E_в > \tau_{\text{мін}}$, отже інвестор може бути зацікавлений у фінансуванні цього дослідження.

Далі розраховуємо період окупності інвестицій $T_{ок}$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \qquad (4.17)$$

де $E_в$ — внутрішня економічна дохідність вкладених інвестицій.

$$T_{ок} = \frac{1}{1{,}02} = 0{,}98 \text{ року}$$

Оскільки $T_{ок} = 0{,}98$ року тоді розвиток доречний.

ВИСНОВКИ

В даній магістерській кваліфікаційній роботі програмний засіб інформаційної системи складського обліку товарно-матеріальних цінностей з інтегрованим рекомендаційном сервісом. Розроблений сервіс є ефективним, зрозумілим для користувача, рекомендаційна система здатна працювати в умовах холодного старту завдяки застосуванню «бандитських» алгоритмів.

У першому розділі магістерської роботи було виконано аналіз існуючих на ринку систем складського обліку, виділено їх переваги та недоліки, визначено вимоги до розроблюваної системи, розглянуто принцип роботи рекомендаційних сервісів.

У другому розділі магістерської роботи було визначено архітектурні особливості програмного засобу, а саме клієнт-серверну архітектуру з виділенням рекомендаційної системи в окремий сервіс. Розглянуто методи для розробки серверної частини додатку, обрано підхід з побудовою RESTful API. Було розглянуто «бандитські» алгоритми роботи рекомендаційних систем та визначення найбільш ефективного в умовах холодного старту. Для розробки клієнтської частини було обрано тонкий клієнт, а саме інтерфейс доступу у вигляді веб-сайту.

У третьому розділі магістерської роботи було обрано інструментарій для розробки програмного продукту. В якості основного інструменту розробки серверної частини обрано фреймворк ASP.NET Core та мову програмування C#, клієнтської — React та мову програмування Javascript. В якості середовищ для розробки використано Visual Studio (для серверної частини) та Visual Studio Code (для клієнтської частини). На основі вибраних засобів було розроблено та досліджено ефективність створеного прикладного програмного продукту.

У четвертому розділі магістерської кваліфікаційної роботи було виконано обґрунтування доцільності розробки нового вирішення представленої задачі по розробці системи складського обліку, здійснено розрахунок потрібних економічних затрат, що необхідні для реалізації

запропонованих засобів і позначено комерційні переваги впровадження створеного програмного продукту.

Всі задачі виконано, мету досягнено. Розроблений програмний продукт може використовуватися в навчальних цілях та на підприємствах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Гладкий А. Складской учет на компьютере. Лучшие программы, включая 1С 8.2 / Алексей Гладкий., 2013.
2. Складской учет: поступление и оприходование товара [Електронний ресурс] – Режим доступу до ресурсу: https://class365.ru/skladskoi-uchet/oprihodovanie-tovarov.
3. Складской учет это [Електронний ресурс] – Режим доступу до ресурсу: https://center-yf.ru/data/Buhgalteru/skladskoy-uchet-eto.php.
4. Складской учет, ведение складского учета [Електронний ресурс] – Режим доступу до ресурсу: https://spmag.ru/articles/organizaciya-skladskogo-ucheta-na-predpriyatii.
5. Тонкий клиент [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Тонкий_клиент.
6. Веб-приложение [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Веб-приложение.
7. Model-View-Controller [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Model-View-Controller.
8. Spring Framework [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Spring_Framework.
9. Фримен А. ASP.NET Core MVC с примерами на C# для профессионалов / Адам Фримен., 2017. – 992 с. – (6).
10. Арсиновски Д. Рефакторинг в C# и ASP.NET для профессионалов / Даниэль Арсиновски., 2010. – 528 с. – (Программистам от программистов).
11. ASP.NET MVC Framework [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/ASP.NET_MVC_Framework.
12. ASP.NET MVC 5 | Полное руководство [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sharp/mvc5/.
13. SQL - Энциклопедия языков программирования [Електронний ресурс] – Режим доступу до ресурсу: http://progopedia.ru/language/sql/.
14. Что такое SQL и для чего он нужен [Електронний ресурс] – Режим доступу до ресурсу: https://www.zeluslugi.ru/info-czentr/it-glossary/term-sql.

15. What is Entity Framework? [Електронний ресурс] – Режим доступу до ресурсу: https://www.entityframeworktutorial.net/what-is-entityframework.aspx.

16. Что такое Entity Framework и как его использовать [Електронний ресурс] – Режим доступу до ресурсу: https://skillbox.ru/media/code/entity_framework/.

17. Введение в Entity Framework 6 [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sharp/entityframework/1.1.php.

18. Що таке RESTful API? [Електронний ресурс] – Режим доступу до ресурсу: https://codeguida.com/post/601.

19. REST [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/REST.

20. REST: простым языком [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@andr.ivas12/ rest-простым-языком-90a0bca0bc78.

21. Фальк К. Рекомендаційні системи на практиці / Кім Фальк., 2020. – 448 с.

22. Li L., Chu W., Langford J., Schapire R. E. A contextual-bandit approach to personalized news article recommendation // Proceedings of the 19th International Conference on World Wide Web. 2010. P. 661–670.

23. Musical recommendations and personalization in a social network // Proceedings of the 7th ACM Conference on Recommender Systems. 2013. P. 367–370.

24. Auer P., Cesa-Bianchi N., Fischer P. Finite-time analysis of the multiarmed bandit problem // Machine Learning. 2002. Vol. 47. No 2–3. P. 235–256.

25. Yahoo! Front page today module user click log dataset, version 1.0 (1.1 GB) [Електронний ресурс]: URL:https://webscope.sandbox.yahoo.com/catalog.php?datatype=r&did=49 (дата звернення: 15.11.2021).

26. Проблема "холодного старту" / Р. З. Омаров, А. В. Востротіна, А. Д. Лі [та ін.] // Молодий уче-ний. - 2019. - № 26 (264). - С. 85-88. [Електронний ресурс]: URL: https://moluch.ru/archive/264/61285/ (дата звернення: 15.11.2021).

27. Даниленко М.С. Система складського обліку товарно – матеріальних цінностей на базі платформи .NET Framework. / М.С. Даниленко, О.О. Григоришен, І.С. Колесник. // Тези доповіді. XLIX регіональна науково-технічна конференція професорсько-викладацького складу, співробітників та студентів університету з участю працівників науково-дослідних організацій та інженерно-технічних працівників підприємств м. Вінниці та області. – Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2020/paper/view/9353/7714.

28. Кавецький В. В. Економічне обґрунтування інноваційних рішень: Практикум /В.В.Кавецький, В.О.Козловський, І.В.Причепа. — ВНТУ, 2013. — 110 с.

29. Адлер О.О. Методичні вказівки до підготовки та написання курсової роботи з дисципліни «Економічне обґрунтування інноваційних рішень» / Уклад. О.О.Адлер, І.В.Причепа, Н.М.Тарасюк. — Вінниця: ВНТУ, 2014. — 38 с.

30. Тарифи на електроенергію [Електронний ресурс]. Режим доступу: http://index.minfin.com.ua/tarif/electric.php. Дата звернення: Листопад 30, 2021.

31. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. — Вінниця : ВНТУ, 2021. — 42 с.

ДОДАТОК А

Міністерство освіти та науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

_____проф., д.т.н. О. Д. Азаров

«___» _____ 2021 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

«Інформаційна система складського обліку товарно-матеріальних цінностей»

08-23.МКР.021.00.000 ТЗ

Науковий керівник: д.т.н., доцент

_____ Колесник І. С.

Магістрант групи 2КІ-20м

_____ Даниленко М. С.

Опонент: к.т.н., доц., зав. каф МБІС

_____ Карпінець В.В.

Вінниця 2021

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

1.1 Актуальність даного дослідження визначається відсутністю на ринку систем складського обліку універсального та високоефективного засобу, який буде мати інтуїтивно зрозумілий для користувача інтерфейс та вбудовану рекомендаційну систему для закупок.

1.2 Наказ про затвердження теми магістерської кваліфікаційної роботи.

2 Мета і призначення МКР

2.1 Мета магістерської роботи полягає у підвищенні ефективності розпізнавання особи за зображенням обличчя.

2.2 Призначення розробки — виконання магістерської кваліфікаційної роботи.

3 Вихідні дані для виконання МКР

Проаналізувати методи і засоби для розробки інформаційної системи складського обліку. Розробити програму, що дозволяє ефективно керувати товарно-матеріальними цінностями складу. Розробити інтегровану рекомендаційну систему для підвищення ефективності закупки товару.

4 Вимоги до виконання МКР

МКР повинна задовольняти такі вимоги:

— запропонувати нові підходи для реалізації систем складського обліку;

— розробити алгоритм для інтегрованої рекомендаційної системи, який буде працювати в умовах холодного старту;

— результат роботи, а саме функціональний програмний засіб із інтуїтивно зрозумілим користувацьким інтерфейсом.

5 Етапи МКР та очікувані результати таблиця А.1

Таблиця А.1 — Етапи виконання роботи

№	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз завдання. Вступ	07.09.21	10.09.21	Вступ
2	Аналіз літературних джерел для розпізнавання особи	13.09.21	22.09.21	розділ 1

3	Розробка технічного завдання	23.09.2021	24.09.21	Технічне завдання
3	Розробка структури системи розпізнавання особи за зображенням обличчя	25.09.21	08.10.21	Розділ 2, розробка структури
4	Розробка програми, проектування програмного продукту	11.10.21	29.10.21	Розділ 3, розробка програми
5	Практична реалізація, результати.	01.11.21	14.11.21	Розділ 3
6	Розробка економічної частини	15.11.21	30.11.21	Розділ 4
7	Оформлення пояснювальної записки	02.12.21	15.12.21	ПЗ, презентація

6 Матеріали, що подаються до захисту МКР: пояснювальна записка МКР, ілюстративні та графічні матеріали, протокол попереднього захисту МКР на кафедрі, відзив наукового керівника, відзив рецензента, протоколи складання державних екзаменів, анотації до МКР українською та іноземною мовами, довідка про відповідність оформлення МКР діючим вимогам.

7 Порядок контролю виконання та захисту МКР

Виконання етапів розрахункової та графічної документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 Вимоги до оформлення МКР

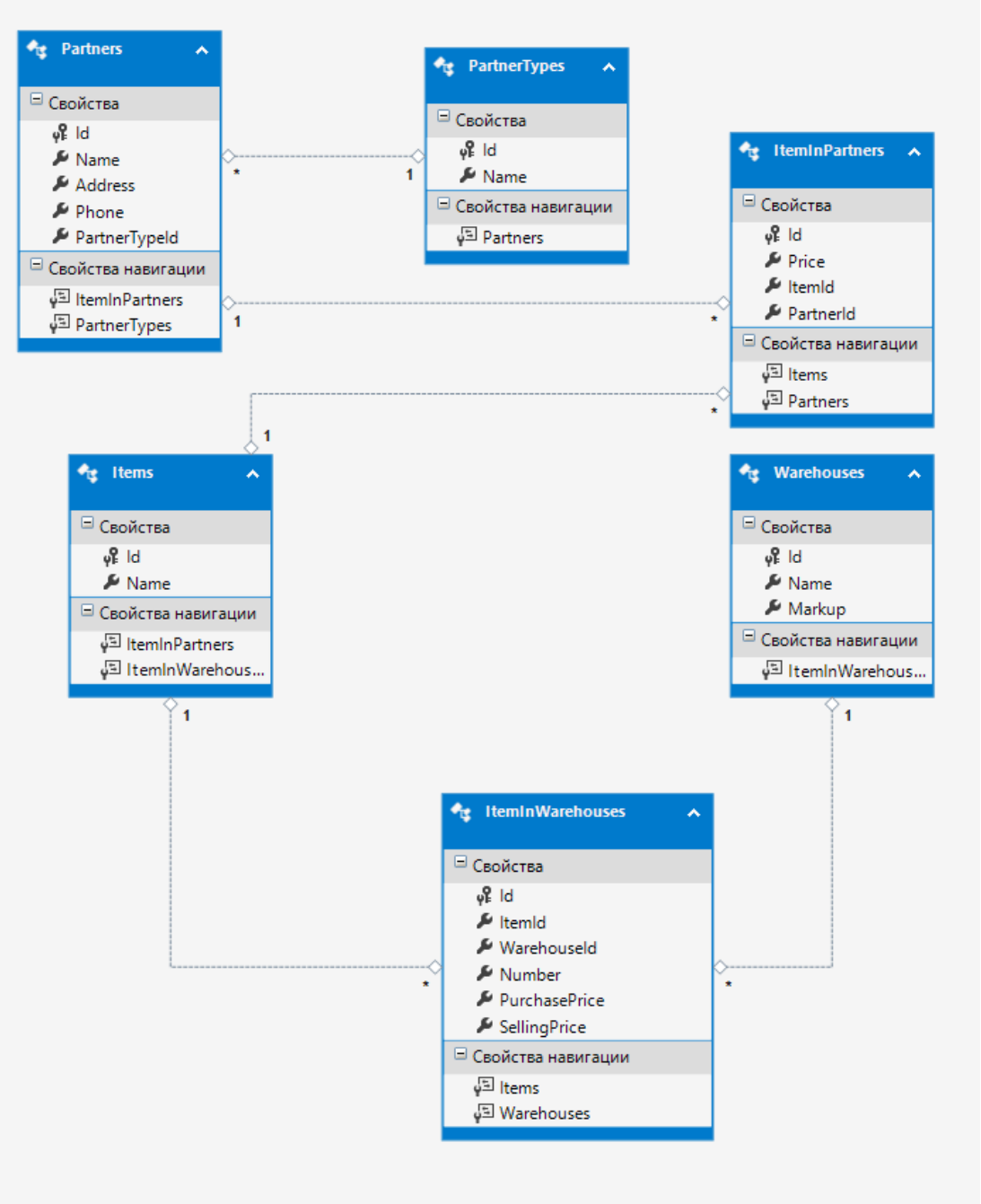
Вимоги викладені в «Положенні про порядок підготовки магістрів у Вінницькому національному технічному університеті» з урахуванням змін, що подані у бюлетені ВАК України № 9-10, 2011р., а також в МЕТОДИЧНИХ ВКАЗІВКАХ до дипломного проектування, ДСТУ 3008-2015, ДСТУ 3974-2000 «Правила виконання дослідно-конструкторських робіт. Загальні положення» та діючого ГОСТ 2.114-95 ЄСКД.

9 Вимоги щодо технічного захисту інформації в МКР з обмеженим доступом відсутні.

Технічне завдання до виконання отримав _____ Даниленко М. С.

ДОДАТОК Б

Діаграма бази даних



ДОДАТОК В

Лістинг серверної частини програми

```
public class ItemsController : ApiController{
    private ApplicationDbContext _context;
    public ItemsController(){
        _context = new ApplicationDbContext();
    }
    //GET /api/items
    public IEnumerable<Item> GetItems(){
        return _context.Items.ToList();
    }
    //GET /api/items/1
    public Item GetItem(int id){
        var item = _context.Items.SingleOrDefault(i => i.Id == id);
        if (item == null){
            throw new HttpResponseException(HttpStatusCode.NotFound);
        }
        return item;
    }
    //POST /api/items
    [HttpPost]
    public Item CreateItem(Item item){
        if (!ModelState.IsValid){
            throw new HttpResponseException(HttpStatusCode.BadRequest);
        }
        _context.Items.Add(item);
        _context.SaveChanges();
        return item;
    }
    //PUT /api/items/1
```

```
        [HttpPut]
        public Item UpdateItem(int id, Item item){
            if (!ModelState.IsValid){
                throw new HttpResponseException(HttpStatusCode.BadRequest);
            }
            var itemInDb = _context.Items.SingleOrDefault(i => i.Id == id);
            if (itemInDb == null){
                throw new HttpResponseException(HttpStatusCode.NotFound);
            }
            itemInDb.Name = item.Name;
            _context.SaveChanges();
            return itemInDb;
        }
        // DELETE /api/items/1
        [HttpDelete]
        public void DeleteItem(int id){
            var itemInDb = _context.Items.SingleOrDefault(i => i.Id == id);
            if (itemInDb == null){
                throw new HttpResponseException(HttpStatusCode.NotFound);
            }
            _context.Items.Remove(itemInDb);
            _context.SaveChanges();
        }
    }
public class ItemsController : Controller{
        private ApplicationDbContext _context;
        public ItemsController(){
            _context = new ApplicationDbContext();
        }
        protected override void Dispose(bool disposing){
            _context.Dispose();
```

```
}
// GET: Items
public ViewResult Index(){
    var items = _context.Items.ToList();
    return View(items);
}
// GET: Items/Details/id
public ActionResult Details(int id){
    var item = _context.Items.SingleOrDefault(i => i.Id == id);
    if (item == null){
        return HttpNotFound();
    }
    return View(item);
}
public ViewResult New(){
    var viewModel = new ItemViewModel();
    return View("ItemForm", viewModel);
}
public ActionResult Edit(int id){
    var item = _context.Items.SingleOrDefault(w => w.Id == id);
    if (item == null)
        return HttpNotFound();
    var viewModel = new ItemViewModel{
        Item = item
    };
    return View("ItemForm", viewModel);
}
[HttpPost]
public ActionResult Save(Item item){
    if (item.Id == 0){
        _context.Items.Add(item);
```

```
            }
            else{
                var itemInDb = _context.Items.Single(w => w.Id == item.Id);
                itemInDb.Name = item.Name;
            }
            _context.SaveChanges();
            return RedirectToAction("Index", "Items");
        }
        public ActionResult Delete(int id){
            var itemInDb = _context.Items.SingleOrDefault(w => w.Id == id);
            _context.Items.Remove(itemInDb);
            _context.SaveChanges();
            return RedirectToAction("Index", "Items");
        }
    }
    public class PartnersController : Controller{
        private ApplicationDbContext _context;
        public PartnersController(){
            _context = new ApplicationDbContext();
        }
        protected override void Dispose(bool disposing){
            _context.Dispose();
        }
        // GET: Partners
        public ViewResult Index(){
            var partners = _context.Partners.Include(p => p.PartnerType).ToList();
            return View(partners);
        }
        public ActionResult Details(int id){
```

```
            var partner = _context.Partners.Include(p => p.PartnerType).SingleOrDefault(p => p.Id == id);
            if (partner == null){
                return HttpNotFound();
            }
            var items = _context.ItemsInPartners.Include(i => i.Item)
                .Where(i => i.PartnerId == id).ToList();
            PartnerViewModel partnerViewModel = new PartnerViewModel{
                Partner = partner,
                Items = items
            };
            return View(partnerViewModel);
        }
        public ViewResult New(){
            var partnerTypes = _context.PartnersTypes.ToList();
            PartnerViewModel viewModel = new PartnerViewModel{
                PartnerTypes = partnerTypes
            };
            return View("PartnerForm", viewModel);
        }
        public ActionResult Edit(int id){
            var partner = _context.Partners.Include(p => p.PartnerType).SingleOrDefault(w => w.Id == id);
            if (partner == null)
                return HttpNotFound();
            var partnerTypes = _context.PartnersTypes.ToList();
            var viewModel = new PartnerViewMode{
                Partner = partner,
                PartnerTypes = partnerTypes
            };
            return View("PartnerForm", viewModel);
```

```
        }
        [HttpPost]
        public ActionResult Save(Partner partner){
            if (partner.Id == 0){
                if (partner.PartnerTypeId == 0){
                    partner.PartnerTypeId = 3;
                }
                _context.Partners.Add(partner);
            }
            else{
                var partnerInDb = _context.Partners.Single(w => w.Id == partner.Id);
                partnerInDb.Name = partner.Name;
                partnerInDb.Phone = partner.Phone;
                partnerInDb.Address = partner.Address;
                partnerInDb.PartnerTypeId = partner.PartnerTypeId;
            }
            _context.SaveChanges();
            return RedirectToAction("Index", "Partners");
        }
        public ActionResult Delete(int id){
            var partnerInDb = _context.Partners.SingleOrDefault(w => w.Id == id);
            _context.Partners.Remove(partnerInDb);
            _context.SaveChanges();
            return RedirectToAction("Index", "Partners");
        }
        public ViewResult NewItem(int id){
            var partner = _context.Partners.Single(w => w.Id == id);
            var items = _context.Items.ToList();
            var partners = _context.Partners.ToList();
```

```
            ItemInPartnerViewModel viewModel = new ItemInPartnerViewModel{
                Partner = partner,
                Partners = partners,
                Items = items,
            };
            return View("NewItemForm", viewModel);
        }
        public ActionResult EditItem(int id, Partner partner){
            var item = _context.ItemsInPartners.Include(i => i.Item).SingleOrDefault(w => w.Id == id);
            var items = _context.Items.ToList();
            var partners = _context.Partners.ToList();
            var viewModel = new ItemInPartnerViewModel{
                Partners = partners,
                Partner = partner,
                Item = item,
                Items = items
            };
            return View("EditItemForm", viewModel);
        }
        [HttpPost]
        public ActionResult SaveNewItem(ItemInPartner item){
            _context.ItemsInPartners.Add(item);
            _context.SaveChanges();
            return RedirectToAction("Details", "Partners", new { id = item.PartnerId });
        }
        [HttpPost]
        public ActionResult SaveEditItem(ItemInPartner item){
```

```
                var itemInDb = _context.ItemsInPartners.Single(w => w.Id ==
item.Id);
                itemInDb.ItemId = item.ItemId;
                itemInDb.PartnerId = item.PartnerId;
                itemInDb.Price = item.Price;
                _context.SaveChanges();
                return RedirectToAction("Details", "Partners", new { id =
item.PartnerId });
            }
        }
    public class WarehousesController : Controller{
            private ApplicationDbContext _context;
            public WarehousesController(){
                _context = new ApplicationDbContext();
            }
            protected override void Dispose(bool disposing){
                _context.Dispose();
            }
            public ViewResult Index(){
                var warehouses = _context.Warehouses.ToList();
                return View(warehouses);
            }
            public ActionResult Details(int id){
                var warehouse = _context.Warehouses.SingleOrDefault(i => i.Id ==
id);
                if (warehouse == null){
                    return HttpNotFound();
                }
                var items = _context.ItemsInWarehouses.Include(i => i.Item)
                    .Where(i => i.WarehouseId == id).ToList();
                for (int i = 0; i < items.Count; i++){
```

```
                for (int j = i + 1; j < items.Count; j++){
                    if (items[i].Id != items[j].Id){
                        if (items[i].ItemId == items[j].ItemId){
                            items[i].Number += items[j].Number;
                            items.Remove(items[j]);
                            j--;
                        }
                    }
                }
            }
            WarehouseViewModel warehouseViewModel = new WarehouseViewModel{
                Warehouse = warehouse,
                Items = items
            };
            return View(warehouseViewModel);
        }
        public ActionResult Items(int warehouseId, int itemId){
            var warehouse = _context.Warehouses.SingleOrDefault(i => i.Id == warehouseId);
            var items = _context.ItemsInWarehouses.Include(i => i.Item).Include(i => i.Warehouse)
                .Where(i => i.WarehouseId == warehouseId && i.ItemId == itemId).ToList();
            WarehouseViewModel warehouseViewModel = new WarehouseViewModel{
                Warehouse = warehouse,
                Items = items
            };
            return View(warehouseViewModel);
        }
```

```
        public ViewResult New(){
            var viewModel = new WarehouseViewModel();
            return View("WarehouseForm", viewModel);
        }
        public ActionResult Edit(int id){
            var warehouse = _context.Warehouses.SingleOrDefault(w => w.Id ==
id);
            if (warehouse == null)
                return HttpNotFound();
            var viewModel = new WarehouseViewModel{
                Warehouse = warehouse
            };
            return View("WarehouseForm", viewModel);
        }
        [HttpPost]
        public ActionResult Save(Warehouse warehouse){
            if (warehouse.Id == 0){
                _context.Warehouses.Add(warehouse);
            }
            else{
                var warehouseInDb = _context.Warehouses.Single(w => w.Id ==
warehouse.Id);
                warehouseInDb.Name = warehouse.Name;
            }
            _context.SaveChanges();
            return RedirectToAction("Index", "Warehouses");
        }
        public ActionResult Delete(int id){
            var warehouseInDb = _context.Warehouses.SingleOrDefault(w =>
w.Id == id);
            _context.Warehouses.Remove(warehouseInDb);
```

```
            _context.SaveChanges();
            return RedirectToAction("Index", "Warehouses");
        }
    }
```

ДОДАТОК Г

Лістинг клієнтської частини програми

```
@model WMS.Models.Item

@{ ViewBag.Title = Model.Name; }

<h2>@Model.Name</h2>

<table class="table table-bordered table-hover">

    <tbody>

        <tr>

            <td>Id</td>

            <td>@Model.Id</td>

        </tr>

        <tr>

            <td>Назва</td>

            <td>@Model.Name</td>

        </tr>

    </tbody>

</table>
```

/Items/Index.cshtml

```
@model IEnumerable<WMS.Models.Item>

@{

    ViewBag.Title = "Товари";
```

```
}

<h2>Товари</h2>

<p>

   @Html.ActionLink("Додати товар", "New", "Items", null, new { @class = "btn btn-primary" })

</p>

<table class="table table-bordered table-hover">

   <thead>

      <tr>

         <th>Назва</th>

         <th>Редагувати</th>

      </tr>

   </thead>

   <tbody>

      @foreach (var item in Model){

         <tr>

            <td>@Html.ActionLink(item.Name, "Details", "Items", new { id = item.Id }, null)</td>

            <td>@Html.ActionLink("Редагувати", "Edit", "Items", new { id = item.Id }, null)</td>

         </tr>

      }
```

```
    </tbody>
</table>
```

/Items/ItemForm.cshtml

```
@model WMS.ViewModels.ItemViewModel
@{
    ViewBag.Title = Model.Title;
}
<h2>@Model.Title</h2>
@using (Html.BeginForm("Save", "Items")){
    <div class="form-group">
        @Html.LabelFor(m => m.Item.Name)
        @Html.TextBoxFor(m => m.Item.Name, new { @class = "form-control" })
    </div>
    @Html.HiddenFor(m => m.Item.Id)
    <button type="submit" class="btn btn-primary">Зберегти</button>
}
<br />
@if (Model.Item != null){
    <p>
        @Html.ActionLink("Видалити", "Delete", "Items", new { id = Model.Item.Id
}, new { @class = "btn btn-primary" })
```

```
    </p>
}
/Orders/BuyList
@model List<WMS.Models.Partner>
@{
    ViewBag.Title = "Продавці";
}
<h2>Продавці</h2>
<table class="table table-bordered table-hover">
    <thead>
        <tr>
            <th>Назва</th>
            <th>Тип</th>
            <th>Адреса</th>
            <th>Телефон</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var partner in Model){
            <tr>
```

```
            <td>@Html.ActionLink(partner.Name, "BuyOrder", "Orders", new { id = partner.Id }, null)</td>
            <td>@partner.PartnerType.Name</td>
            <td>@partner.Address</td>
            <td>@partner.Phone</td>
        </tr>
    }
  </tbody>
</table>
```

/Orders/BuyOrder

```
@model WMS.ViewModels.BuyOrderViewModel
@{
  ViewBag.Title = "Купити";
}
<h2>Купити</h2>
@using (Html.BeginForm("SaveBuyOrder", "Orders")){
  <div class="form-group">
    @Html.LabelFor(m => m.Item.ItemId)
    @Html.DropDownListFor(m => m.Item.ItemId, new SelectList(Model.Items, "Id", "Item.Name"), "", new { @class = "form-control" })
  </div>
  <div class="form-group">
```

```
        @Html.LabelFor(m => m.Item.WarehouseId)
        @Html.DropDownListFor(m => m.Item.WarehouseId, new SelectList(Model.Warehouses, "Id", "Name"), "", new { @class = "form-control" })
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Item.Number)
        @Html.TextBoxFor(m => m.Item.Number, new { @class = "form-control" })
    </div>
    @Html.HiddenFor(m => m.Item.Id)
    <button type="submit" class="btn btn-primary">Зберегти</button>
}
```

/Partners/Details.cshtml

```
@model WMS.ViewModels.PartnerViewModel
@{
    ViewBag.Title = Model.Partner.Name;
}
<h2>@Model.Partner.Name</h2>
<p>
    @Html.ActionLink("Додати товар", "NewItem", "Partners", new { id = Model.Partner.Id}, new { @class = "btn btn-primary" })
</p>
<table class="table table-bordered table-hover">
```

```
<tbody>
    <tr>
        <td>Id</td>
        <td>@Model.Partner.Id</td>
    </tr>
    <tr>
        <td>Назва</td>
        <td>@Model.Partner.Name</td>
    </tr>
    <tr>
        <td>Тип</td>
        <td>@Model.Partner.PartnerType.Name</td>
    </tr>
    <tr>
        <td>Адреса</td>
        <td>@Model.Partner.Address</td>
    </tr>
    <tr>
        <td>Телефон</td>
        <td>@Model.Partner.Phone</td>
    </tr>
```

```
    </tbody>
</table>
<table class="table table-bordered table-hover">
    <thead>
        <tr>
            <th>Назва</th>
            <th>Ціна</th>
            <th>Редагувати</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model.Items){
            <tr>
                <td>@item.Item.Name</td>
                <td>@item.Price</td>
                <td>@Html.ActionLink("Редагувати", "EditItem", "Partners", new { id = item.Id, partner = Model.Partner }, null)</td>
            </tr>
        }
    </tbody>
</table>
```

/Partners/EditItemForm.cshtml

```
@model WMS.ViewModels.ItemInPartnerViewModel

@{
    ViewBag.Title = Model.Title;
}

<h2>@Model.Title</h2>

@using (Html.BeginForm("SaveEditItem", "Partners")){
    <div class="form-group">
        @Html.LabelFor(m => m.Item.PartnerId)
        @Html.DropDownListFor(m => m.Item.PartnerId, new SelectList(Model.Partners, "Id", "Name"), "", new { @class = "form-control" })
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Item.ItemId)
        @Html.DropDownListFor(m => m.Item.ItemId, new SelectList(Model.Items, "Id", "Name"), "", new { @class = "form-control" })
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Item.Price)
        @Html.TextBoxFor(m => m.Item.Price, new { @class = "form-control" })
    </div>
    @Html.HiddenFor(m => m.Item.Id)
```

```
    <button type="submit" class="btn btn-primary">Зберегти</button>
}
```

/Partners/Index.cshtml

```
@model IEnumerable<WMS.Models.Partner>
@{
    ViewBag.Title = "Контрагенти";
}
<h2>Контрагенти</h2>
<p>
    @Html.ActionLink("Додати контрагента", "New", "Partners", null, new { @class = "btn btn-primary" })
</p>
<table class="table table-bordered table-hover">
    <thead>
        <tr>
            <th>Назва</th>
            <th>Тип</th>
            <th>Адреса</th>
            <th>Номер телефону</th>
            <th>Редагувати</th>
        </tr>
```

```
    </thead>
    <tbody>
        @foreach (var partner in Model){
            <tr>
                <td>@Html.ActionLink(partner.Name, "Details", "Partners", new { id = partner.Id }, null)</td>
                <td>@partner.PartnerType.Name</td>
                <td>@partner.Address</td>
                <td>@partner.Phone</td>
                <td>@Html.ActionLink("Редагувати", "Edit", "Partners", new { id = partner.Id }, null)</td>
            </tr>
        }
    </tbody>
</table>
```

/Partners/NewItemForm.cshtml

```
@model WMS.ViewModels.ItemInPartnerViewModel

@{
    ViewBag.Title = "Додати товар";
}
<h2>Додати товар</h2>
```

```
@using (Html.BeginForm("SaveNewItem", "Partners")){
    <div class="form-group">
        @Html.LabelFor(m => m.Item.Partner)
        @Html.DropDownListFor(m => m.Item.PartnerId, new SelectList(Model.Partners, "Id", "Name"), "", new { @class = "form-control" })
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Item.Item)
        @Html.DropDownListFor(m => m.Item.ItemId, new SelectList(Model.Items, "Id", "Name"), "", new { @class = "form-control" })
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Item.Price)
        @Html.TextBoxFor(m => m.Item.Price, new { @class = "form-control" })
    </div>
    @Html.HiddenFor(m => m.Item.Id)
    <button type="submit" class="btn btn-primary">Save</button>
}
```

/Partners/PartnerForm.cshtml

```
@model WMS.ViewModels.PartnerViewModel
@{
    ViewBag.Title = Model.Title;
```

```
}

<h2>@Model.Title</h2>

@using (Html.BeginForm("Save", "Partners")){

    <div class="form-group">

        @Html.LabelFor(m => m.Partner.Name)

        @Html.TextBoxFor(m => m.Partner.Name, new { @class = "form-control" })

    </div>

    <div class="form-group">

        @Html.LabelFor(m => m.Partner.PartnerTypeId)

        @Html.DropDownListFor(m => m.Partner.PartnerTypeId, new
SelectList(Model.PartnerTypes, "Id", "Name"), "", new { @class = "form-control"
})

    </div>

    <div class="form-group">

        @Html.LabelFor(m => m.Partner.Address)

        @Html.TextBoxFor(m => m.Partner.Address, new { @class = "form-control"
})

    </div>

    <div class="form-group">

        @Html.LabelFor(m => m.Partner.Phone)

        @Html.TextBoxFor(m => m.Partner.Phone, new { @class = "form-control" })

    </div>
```

```
    @Html.HiddenFor(m => m.Partner.Id)
    <button type="submit" class="btn btn-primary">Зберегти</button>
}
<br />
@if (Model.Partner != null){
    <p>
        @Html.ActionLink("Видалити", "Delete", "Partners", new { id = Model.Partner.Id }, new { @class = "btn btn-primary" })
    </p>
}
```

/Warehouses/Details.cshtml

```
@model WMS.ViewModels.WarehouseViewModel

@{
    ViewBag.Title = Model.Warehouse.Name;
}
<h2>@Model.Warehouse.Name</h2>
<table class="table table-bordered table-hover">
    <tbody>
        <tr>
            <td>Id</td>
```

```
            <td>@Model.Warehouse.Id</td>
        </tr>
        <tr>
            <td>Назва</td>
            <td>@Model.Warehouse.Name</td>
        </tr>
        <tr>
            <td>Націнка</td>
            <td>@Model.Warehouse.Markup</td>
        </tr>
        <tr>
</tbody>
</table>
<table class="table table-bordered table-hover">
    <thead>
        <tr>
            <th>Товар</th>
            <th>Склад</th>
            <th>Кількість</th>
        </tr>
    </thead>
```

```
    <tbody>
        @foreach (var item in Model.Items){
            <tr>
                <td>
                    @Html.ActionLink(item.Item.Name, "Items", "Warehouses",
                    new { warehouseId = item.WarehouseId, itemId = item.ItemId }, null)
                </td>
                <td>@item.WarehouseId</td>
                <td>@item.Number</td>
            </tr>
        }
    </tbody>
</table>
```

/Warehouses/Index.cshtml

```
@model IEnumerable<WMS.Models.Warehouse>
@{
    ViewBag.Title = "Склади";
}
<h2>Склади</h2>
<p>
```

```
    @Html.ActionLink("Додати склад", "New", "Warehouses", null, new { @class = "btn btn-primary" })
</p>
<table class="table table-bordered table-hover">
    <thead>
        <tr>
            <th>Назва</th>
            <th>Націнка</th>
            <th>Редагувати</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var warehouse in Model){
            <tr>
                <td>@Html.ActionLink(warehouse.Name, "Details", "Warehouses", new { id = warehouse.Id }, null)</td>
                <td>@warehouse.Markup</td>
                <td>@Html.ActionLink("Редагувати", "Edit", "Warehouses", new { id = warehouse.Id }, null)</td>
            </tr>
        }
    </tbody>
```

```
</table>
```

/Warehouses/Items.cshtml

```
@model WMS.ViewModels.WarehouseViewModel
@{
    ViewBag.Title = Model.Warehouse.Name;
}
<h2>@Model.Warehouse.Name</h2>
<table class="table table-bordered table-hover">
    <tbody>
        <tr>
            <td>Id</td>
            <td>@Model.Warehouse.Id</td>
        </tr>
        <tr>
            <td>Назва</td>
            <td>@Model.Warehouse.Name</td>
        </tr>
        <tr>
            <td>Markup</td>
            <td>@Model.Warehouse.Markup</td>
        </tr>
```

```
    </tbody>
</table>
<table class="table table-bordered table-hover">
    <thead>
        <tr>
            <th>Id</th>
            <th>Назва</th>
            <th>Кількість</th>
            <th>Ціна купівлі</th>
            <th>Ціна продажу</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model.Items){
            <tr>
                <td>@item.Id</td>
                <td>@item.Item.Name</td>
                <td>@item.Number</td>
                <td>@item.PurchasePrice</td>
                <td>@item.SellingPrice</td>
            </tr>
```

```
        }
    </tbody>
</table>
```

/Warehouses/WarehouseForm.cshtml

```
@model WMS.ViewModels.WarehouseViewModel
@{
    ViewBag.Title = Model.Title;
}
<h2>@Model.Title</h2>
@using (Html.BeginForm("Save", "Warehouses")){
    <div class="form-group">
        @Html.LabelFor(m => m.Warehouse.Name)
        @Html.TextBoxFor(m => m.Warehouse.Name, new { @class = "form-control" })
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Warehouse.Markup)
        @Html.TextBoxFor(m => m.Warehouse.Markup, new { @class = "form-control" })
    </div>
    @Html.HiddenFor(m => m.Warehouse.Id)
    <button type="submit" class="btn btn-primary">Зберегти</button>
```

```
}

<br />

@if (Model.Warehouse != null){

   <p>

      @Html.ActionLink("Delete", "Видалити", "Warehouses", new { id = Model.Warehouse.Id }, new { @class = "btn btn-primary" })

   </p>

}
```

ДОДАТОК Д

Графічні зображення роботи програми

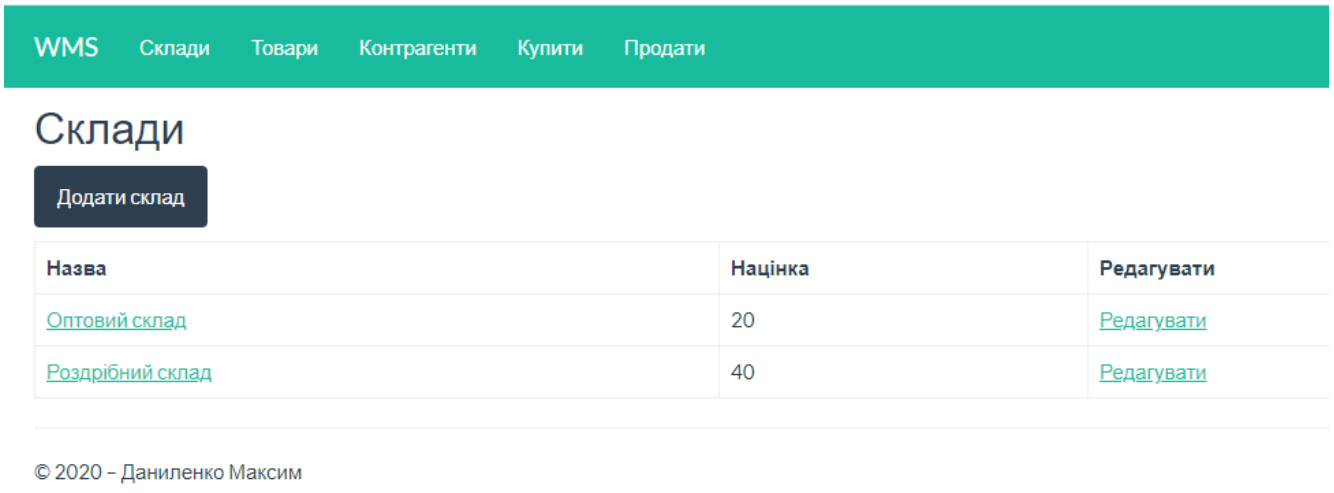


Рисунок Д.1 — Список складів

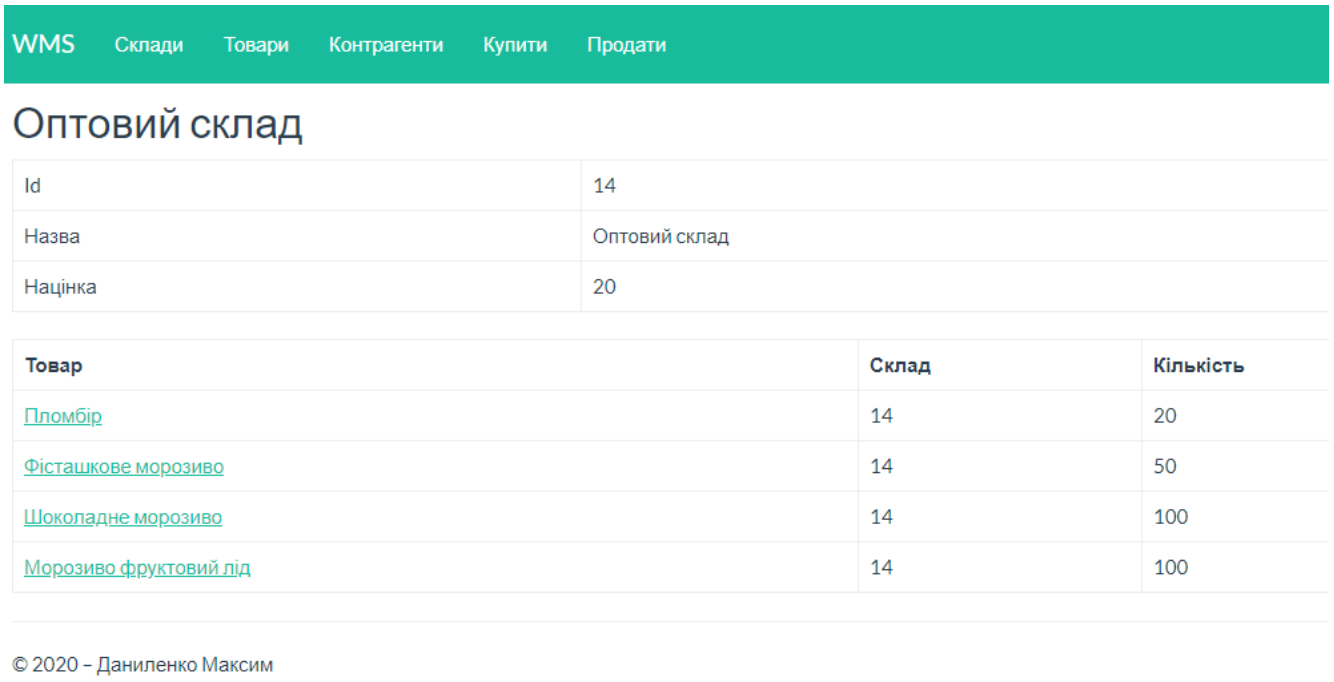


Рисунок Д.2 — Список товарів на складі

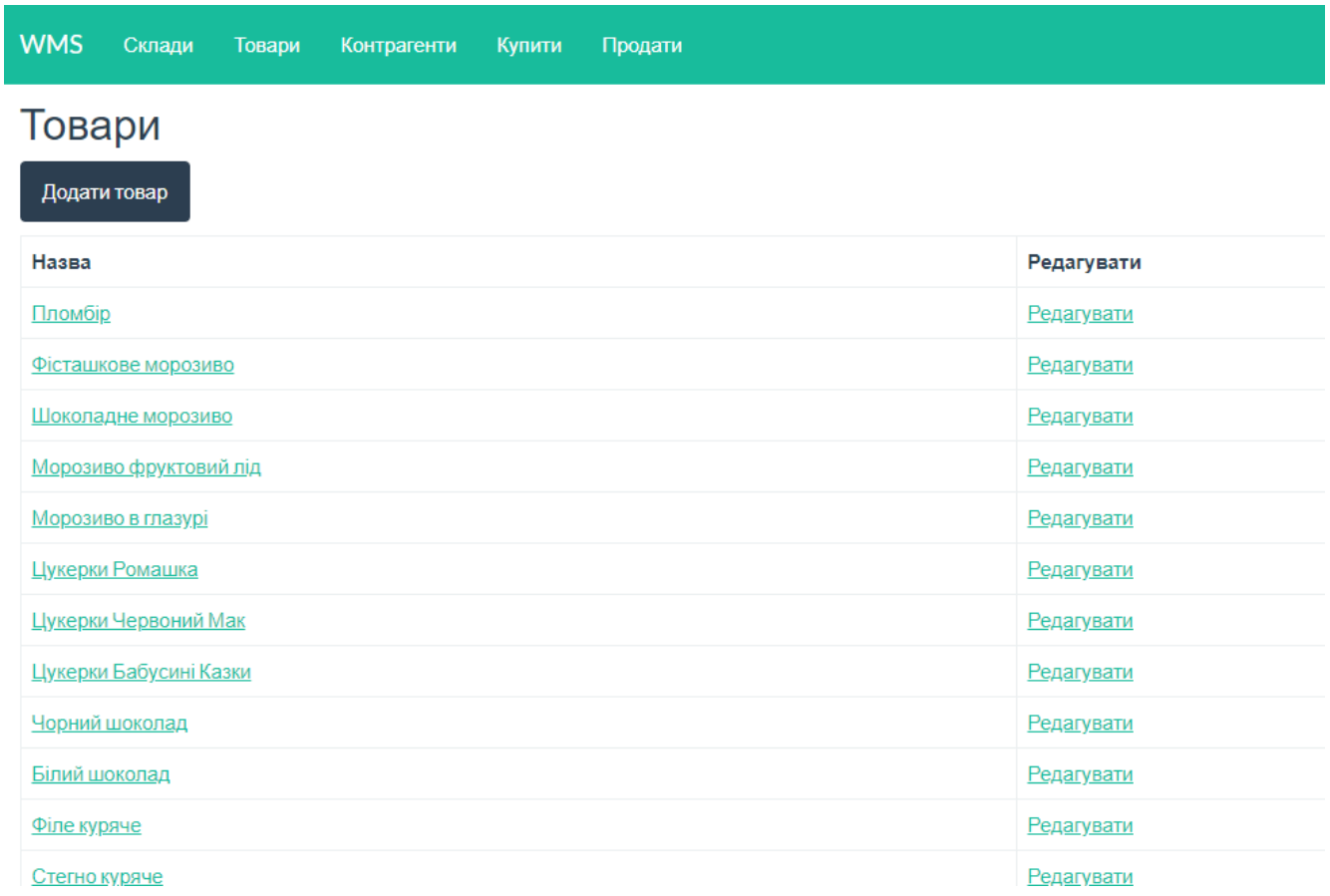


Рисунок Д.3 — Список номенклатур товарів

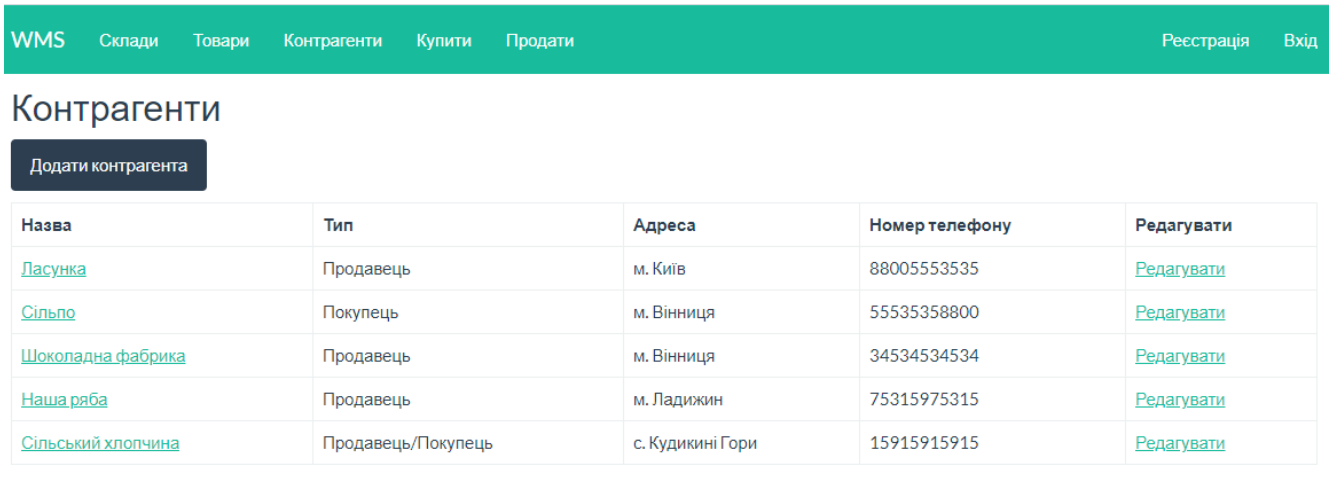


Рисунок Д.4 — Список контрагентів

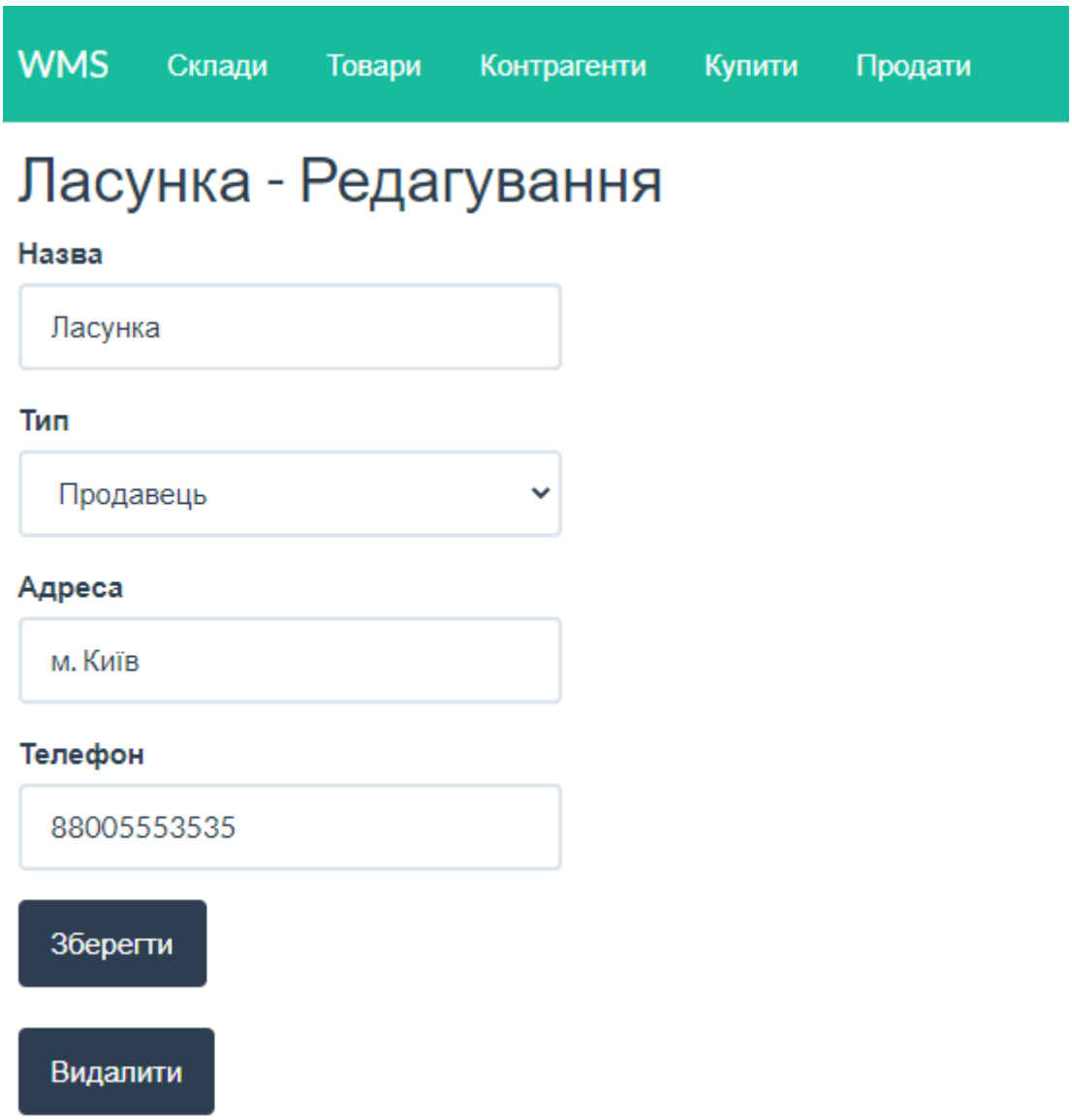


Рисунок Д.5 — Редагування контрагенту