МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра інженерії програмного забезпечення

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри Олексій Горський
"____"_____2023 р.

ДИПЛОМНИЙ ПРОЕКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСНИКА ОСВІТНЬОГО СТУПЕНЯ
МАГІСТРА

Тема: " Система управління логістичним центром підприємства. "

Виконавець: Аврамич Андрій Миколайович

Керівник: к.т.н Вовк Володимир Григорович

Нормоконтролер: ст.викл Гололобов Дмитро Олександрович

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра інженерії програмного забезпечення

Освітній ступінь магістр

Спеціальність 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Програмне забезпечення систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри Олексій Горський

"___" _____ 2023 р

ЗАВДАННЯ

на виконання дипломного проекту студента

Аврамича Андрія Миколайовича

1. Тема проекту: «Система управління логістичним центром підприємства.» затверджена наказом ректора від 29.09.2023 р. № 1994/ст.
2. Термін виконання проекту: з 02.09.20223р. по 31.12.2023 р.
3. Вихідні данні до проекту: програмний продукт розробити у вигляді , бекенд сервісу та веб-додатку для користувацького інтерфейсу

4. Зміст пояснювальної записки:

1. Існуючі системи управління логістичними центрами огляд роботи логістичних центрів.
2. Вимоги до системи управління логістичним центром та його архітектури.

3. Реалізація системи управління логістичним центром та його архітектури.
4. Прототип програми

5. Перелік обов'язкових слайдів презентації:

1. Структурна схема логістичних центрів України
2. Підходи до оцінки надійності архітектури бекенд частини
3. Вимоги до програмного засобу
4. Огляд користувацького інтерфейсу
5. Аналіз структури веб-додатків та принципів їх побудови
6. Демонстрація прототипу роботи засобу

6. Календарний план-графік

№ пор	Завдання	Термін виконання	Відмітка про виконання
1.	Складання та затвердження графіку роботи дипломного проектування	09.10.23 – 15.10.23	
2.	Підготовка та написання 1 розділу. Відсилка керівнику	15.10.23 – 22.10.23	
3.	Підготовка та написання 2 розділу. Відсилка керівнику	22.10.23 – 5.11.23	
4.	Написання 3 розділу, представлення керівнику	6.11.23 – 19.11.23	

5.	Написання 4 розділу, представлення керівнику	20.11.23 – 26.11.23	
6.	Редагування та друк пояснювальної записки, графічного матеріалу Відсилка ПЗ для перевірки на плагіат одним файлом.	27.11.23-3.12.23	
7.	Проходження нормо-контролю, перепліт пояснювальної записки. Отримання відгуку керівника. Підготовка презентації та тексту доповіді.	4.12.23-10.12.23	
8.	Передзахист (наявність віддрукованої ПЗ, презентації, положительного відгуку керівника). При наявності цього приймається рішення про допуск захисту дипломного проекту перед Екзаменаційною комісією. Що оформлюється протоколом засідання кафедри	11.12.23-17.12.23	
9.	Отримання рецензії. (список рецензентів буде пізніше)	18.12.23-24.12.23	
10.	Здати секретарю ДЕК: ПЗ, ГМ, CD-R з електронними версіями ПЗ, ГМ, презентацію, відгук керівника, рецензію, довідку про успішність, 2 папки, 2 конверта)	25.12.23-31.12.23	

11.	Захист дипломної роботи перед ЕК	25.12.23	

Дата видачі завдання 02.10.2023 р. початок проектування

Керівник дипломної роботи: к.т.н Вовк Володимир Григорович

Завдання прийняв до виконання: Оксана ЗОЛОТОВЕРХ

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Система управління логістичним центром підприємства»

Кількість сторінок: 70

Кількість таблиць: 2

Кількість рисунків: 25

ВЕБ-СЕРВІС, ДИЗАЙН, ІНТЕРФЕЙС, ПРОЕКТУВАННЯ, РОЗРОБКА СИСТЕМИ, БРАУЗЕР, ЛОГІСТИЧНИЙ ЦЕНТР, МІКРОСЕРВІСИ

Об’єкт дослідження - система управління логістичним центром підприємства

Мета дипломної роботи – створення системи управління логістичним центром підприємства, що дасть змогу створити нову систему управління, підвищивши якість надавання логістичних послуг.

Метод розробки – ООП – об’єктно-орієнтований підхід.

Технічні та програмні засоби – ПК з ОС Windows 7, 8 або 10; середовище об’єктно-орієнтованого програмування Microsoft visual studio.

Результати проекту рекомендується використовувати у вільному доступі на просторах інтернету.

Прогнозні припущення щодо розвитку об'єктів розроблення – розроблюваний програмний засіб може бути розширено у майбутньому шляхом створення мобільного додатку та інтеграції з системами оплати

ABSTRACT

Explanatory note to the thesis "The management system of the logistics center of the enterprise"

Number of pages: 70

Number of tables: 2

Number of drawings: 25

WEB SERVICE, DESIGN, INTERFACE, DESIGN, SYSTEM DEVELOPMENT, BROWSER, LOGISTICS CENTER, MICROSERVICES

The object of the study is the management system of the logistics center of the enterprise

The aim of the diploma work is to create a management system for the logistics center of the enterprise, which will make it possible to create a new management system, increasing the quality of the provision of logistics services.

The development method is OOP - an object-oriented approach.

Hardware and software – PC with Windows 7, 8 or 10; Microsoft visual studio object-oriented programming environment.

The results of the project are recommended to be freely accessible on the Internet.

Predictive assumptions regarding the development of development objects - the developed software can be expanded in the future by creating a mobile application and integration with payment systems

Зміст

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

ОС – операційна система

КІ – користувацький інтерфейс

КД – користувацький досвід

БД – баз а даних

ООП – об’єктно-орієнтовне програмування

ОС – операційна система

СУБД – система управління базами даних

ШІ – штучний інтеллект

ВСТУП

Ефективний логістичний центр є важливою складовою успішної діяльності підприємств у сучасному світі. Забезпечення надійного управління та координації всіх логістичних процесів на підприємстві вимагає вдосконаленої системи, яка б забезпечувала оптимальне розподілення ресурсів та максимальну ефективність у виконанні логістичних завдань. Логістичний центр відіграє стратегічну роль у забезпеченні постачання товарів і послуг, оптимізації витрат та забезпеченні високої якості обслуговування клієнтів.++

Данна дипломна робота присвячена вивченню, аналізу та розробленню системи управління логістичним центром підприємства. В контексті постійних змін в глобальному бізнес-середовищі, де ефективність та швидкість поставок стають важливими перевагами над конкурентами, дослідження цієї теми стає надзвичайно актуальним завданням. Дипломна робота включає в себе аналіз поточних логістичних процесів, визначення ключових проблем та пропозиції щодо впровадження вдосконалень у систему управління логістичним центром підприємства з метою підвищення ефективності, зниження витрат та підвищення задоволеності клієнтів та створення прототипу власної логістичної системи.++

Інтернет технології дозволяють надавати підприємствам ряд додаткових логістичних послуг, підвищуючи конкурентоспроможність на ринку: відстеження перевезення вантажів, довідкові дані, порівняння альтернативних варіантів, індивідуалізацію обслуговування.

Також покращується сама система товаропостачання. Перед товаропостачанням стоїть завдання:

1. Вчасне надходження товарів та виконання планового товарообігу на підприємстві;
2. Задоволення попиту на ринку;
3. Забезпечення безперебійної торгівлі та постачання послуг;
5. Підвищення культури торгівлі;
6. Підвищення ефективності роботи торговельного підприємства.

Інформаційна система, що зосередить у собі функції доставки і логістичних операцій дозволить оптимізувати час підприємства, а також

працівників, які працюють у різних сферах економіки. Всі функцію на себе візьме окрема система.

У даній дипломній роботі будуть розглянуті ключові аспекти управління логістичним центром, стратегічне планування, оптимізація процесів та впровадження сучасних технологій. Також, вихідний прототип системи управління впровадить сучасні технології.++

Ця дипломна робота має на меті зрозуміти важливість системи управління логістичним центром та її вплив на ефективність підприємства, а також надати рекомендації для подальшого вдосконалення прототипа системи та способи його впровадження на існуючому підприємстві.

РОЗДІЛ 1. Аналіз існуючих систем управління логістичними центрами

1.1 Поняття системи управління логістичним центром. Огляд сучасного стану логістики в Україні

Логістика в Україні є важливою складовою економіки країни. Забезпечує ефективне управління потоками матеріальних, фінансових та інформаційних ресурсів у всіх сферах бізнесу. Останні десятиліття відомі значними змінами в логістичній сфері України, які впливають на стан і перспективи її розвитку.

Однією з ключових проблем логістики в Україні є недостатня інфраструктура. Багато регіонів країни не мають належної мережі доріг, що ускладнює процес доставки вантажів. Крім того, загальна зношеність транспортних засобів і недосконаліть техніки впливають на швидкість і ефективність доставки вантажів. Для вирішення цих проблем необхідні інвестиції в інфраструктуру та впровадження сучасних технологій.

Ще одним важливим аспектом логістики в Україні є комунікація з міжнародними логістичними мережами. Україна є транзитною країною, через яку транспортуються великі обсяги вантажів між Європою та Східною Азією. Однак погіршення транспортних умов через конфлікт на сході України та зміни торгово-економічної політики Росії загрожують стабільності та розвитку української логістичної галузі. Для забезпечення сталого розвитку логістики українські компанії повинні диверсифікувати свої ринки збуту та шукати нові способи доставки товарів.

Проте, незважаючи на існуючі виклики, логістика в Україні має й позитивні риси. За останні роки в Україні було створено кілька нових логістичних підприємств, які надають широкий спектр послуг з обробки вантажів, управління складом і доставки

Крім того, українські логістичні компанії активно впроваджують нові технології, такі як моніторинг автотранспорту та автоматизовані системи управління складом.

Україна також активно співпрацює з міжнародними організаціями у сфері логістики. Наприклад, Україна є членом Європейської логістичної асоціації логістики та Євразійського Економічного економічного співтовариства. Це дозволяє українським компаніям співпрацювати з логістичними компаніями з інших країн, обмінюватися досвідом та реалізовувати спільні проекти.

Загалом стан логістики в Україні має свої позитивні та негативні аспекти сторони. Недосконала інфраструктура та проблеми в зв'язку з міжнародними логістичними ланцюгами ланцюжками ставлять під загрозу розвиток галузі. Однак, Проте активність діяльність українських компаній та співпраця з міжнародними партнерами стимулюють розвиток української логістичної галузі. Для успішного розвитку логістики в Україні необхідно інвестувати в інфраструктуру, впроваджувати сучасні технології та шукати нові шляхи співпраці з міжнародними логістичними компаніями.

1.2 Рейтинг та вибір логістичних систем

Системи управління логістичним центром підприємства корисні як для малого так і великого бізнесу. Однак у зв'язку з невеликими витратами на організацію логістики на ринок просочуються несумлінні компанії. Звертаючись в подібні організації, можна втратити власний час і гроші. Ось чому користувачі довіряють рейтингам логістичних систем . Рейтинг дозволяє відсортувати компанії за певними вимогами та полегшити вибір для кінцевого клієнта.

У сучасному бізнес-середовищі, де конкуренція загострюється, вибір логістичної системи стає для підприємств все складнішим завданням. Правильний вибір логістичної системи може забезпечити компанії значні конкурентні переваги, такі як зниження витрат, підвищення якості обслуговування та підвищення задоволеності клієнтів. Тому все більше підприємств в Україні звертають увагу на оцінку логістичних систем, щоб зробити ефективний вибір.

Оцінка логістичних систем дозволяє класифікувати та оцінювати логістичні компанії за різними критеріями. Як правило, такими критеріями є якість обслуговування клієнтів, швидкість і надійність поставок, наявність необхідних складських приміщень і обладнання, гнучкість упаковки і транспортування, цінова політика, технологічне оснащення і відповідність стандартам якості. Оцінка системи логістики дає підприємствам можливість зрозуміти, які логістичні компанії найкраще відповідають їхнім вимогам і очікування.

В Україні є кілька організацій та агентств, які займаються оцінкою логістичних систем. Ці організації аналізують та оцінюють роботу логістичних компаній, а також формують рейтингові списки, що дозволяє підприємствам зробити об'єктивний вибір логістичної системи.

Вибір логістичної системи для підприємства – складний і відповідальний процес. Підприємства повинні ретельно проаналізувати оцінку логістичних систем, оцінити їх можливості та позитивний досвід роботи. При виборі системи також важливо враховувати індивідуальні потреби та особливості вашої компанії.

Клієнти формують рейтинг за даними параметрами:

1. **Вартість послуг доставки**. Для більшості клієнтів цей фактор залишається вирішальним. Мало знайдеться охочих переплачувати за аналогічну послугу, якщо компанія конкурент надає її дешевше.
2. **Швидкість доставки.** Перевагу клієнти віддають організаціям, готовим якомога швидше виконати доручення і доставити посилку адресату.
3. **Охоплення території**. Чим в більшій кількості міст відкриті філії компанії, тим краще.
4. **Якість обслуговування**. Клієнти з великим задоволенням повертаються туди, де отримують якісне обслуговування. Це стосується не тільки вартості послуг і швидкості доставки, але і ввічливого спілкування з боку персоналу. Цей фактор потрібно враховувати тим, хто бажає заробити репутацію бізнесу з гарною логістикою
5. **Графік роботи**. Компанії, в яких логістика працює цілодобово мають переваги над компаніями із звичайним графіком. З цієї причини логістична служба підприємства може позбавитись значної частини потенційних клієнтів, що вкрай небажано.
6. **Відсутність черг**. Швидкість обслуговування стосується не тільки швидкості доставки посилок. Фахівці мають досить оперативно обслуговувати клієнтів, щоб уникати скупчення черг, викликаючи тим самим обурення з боку клієнтів.
7. **Зручність оплати**. У рейтинг логістичних служб доставки потрапляють компанії, здатні запропонувати клієнтам найбільш гнучкі умови. Наприклад, не всім зручний варіант передоплати. Деякі клієнти бажають спочатку переконатися в цілості й схоронності посилки, а потім здійснити оплату.
8. **Наявність кур`єрів**. Можливість доставити доставити товар прямо додому клієнту, без потреби приходити у відділення, або у спеціальну точку видачі.

Загалом оцінка та вибір логістичних систем підприємств в Україні є важливим етапом розвитку бізнесу. Надійність, якість і швидкість логістичних процесів є ключовими факторами, що впливають на конкурентоспроможність підприємств. Ретельний аналіз і вибір логістичної системи допоможе компанії досягти успіху і стати лідером ринку.

1.3 Відмінності різних типів додатків

Додатки (також звані програмами або додатками) різняться за різними критеріями і можуть бути класифіковані на різні типи відповідно до їх призначення та функцій. Ось основні типи додатків та їхні відмінності:

1. Мобільні додатки (додатки для смартфонів і планшетів):
   1.1. Нативні додатки: розроблені для певної платформи (iOS, Android, Windows Phone). Розроблені для конкретної платформи (iOS, Android, Windows Phone), які більш ефективно взаємодіють з операційною системою і дозволяють отримати доступ до всіх її функцій.
   1.2. Веб-додатки: запускаються у веб-браузері і доступні через інтернет. Вони не обмежені певною платформою і легко оновлюються.
   1.3. Гібридні додатки: поєднують елементи нативних і веб-додатків. Можна використовувати переваги обох типів.
2. Десктопні додатки (додатки для ПК):
   2.1. Офісні додатки: офісні додатки: включають текстові редактори, електронні таблиці та програми для створення презентацій. Призначені для оптимізації офісної роботи та бізнес-процесів.
   2.2. Графічні редактори: дозволяють редагувати та створювати графічний контент.
   2.3. Антивірусне та захисне програмне забезпечення: забезпечує безпеку та захист від шкідливих програм.
3. Ігрові програми
   3.1. Консольні ігри: розроблені для конкретних ігрових консолей, таких як PlayStation та Xbox.
   3.2. Мобільні ігри: розроблені для смартфонів і планшетів, часто доступні в магазинах додатків, таких як App Store і Google Play.
   3.3. Комп'ютерні ігри: призначені для гри на персональних комп'ютерах; жанри та графічні можливості варіюються.
4. Спеціалізовані програми
   4.1. Медичні програми: надають інформацію про здоров'я та графіки лікування.
   4.2. Освітні додатки: допомагають у навчанні та розвитку різних навичок.
   4.3. Фінансові програми: дозволяють користувачам вести сімейний бюджет і здійснювати операції з банківським рахунком.
5. Споживчі додатки

5.1. Соціальні мережі: забезпечують спілкування та обмін інформацією між користувачами.
5.2. Додатки для шопінгу: дозволяють здійснювати покупки в Інтернеті, порівнювати ціни та отримувати знижки.
5.3. Медіаплеєри: надають можливість транслювати аудіо- та відеоконтент.
5.4. Програми цих категорій представляють лише невелику частину розмаїття ринку програмного забезпечення, і з розвитком технологій можуть з'являтися нові типи додатків.

По платформам на яких працюють додатки можна виділити 3 основних типи: десктопні, мобільні, веб додатки. Розробка додатку під кожний тип платформи буде мати свої відмінності.

Платформа	Desktop	Web	Mobile
Інтернет з'єднання	Не потрібен	Потрібен	Не потрібен для нативних додат потрібен для інших
Встановлення Додатку користувацький пристрій	Додаток повинен бути встановлений	Потрібен бути тіл інстальований браузер	Додаток повинен б встановлений спеціалізованого магазину
Як користу взаємодіє з додатком	Через стандартні інтерфейси	Через встановле браузер	Через стандартні інтерфейси
Сумісність пристроями	Залежність операційної сист користувача	Потрібен тіл завантажений браузер	Залежність операційної сист встановленої мобільному пристрої
Розробка програмного забезпечення	Найчастіше розроблюється під кожну платфо	Кроссплатформенна, потрібен тільки браузер, а програ	На більшості програмування розроблюється під кож

	окремо	забезпечення на сервері багатоплатформовий.	платформу окремо

Отже кожна програмна платформа буде мати свої відмінності і від вибору платформи буде багато, що залежати під час розробки, що вплине на кінцевий результат.

Нативну розробку можна назвати "рідною" для таких операційних систем, як Android, IOS та Win Phone. Такі мобільні додатки органічно інтегровані в саму операційну систему, оскільки вони написані мовами програмування, затвердженими розробником програмного забезпечення для кожної платформи. Додатки завантажуються через магазини додатків (наприклад, App Store, Google Play) і відповідають вимогам цих магазинів.

Нативні додатки можуть працювати повністю або частково без підключення до інтернету, щоб користувачі менше залежали від якості свого з'єднання і могли користуватися додатком у зручному для них місці та часі.

Звичайно, написання таких продуктів вимагає від розробників спеціальних знань і навичок роботи зі специфічними середовищами розробки (xCode для iPhone, eclipse для Android). Як наслідок, вартість таких додатків значно вища через витрачені зусилля та необхідність писати окремі додатки різними мовами для кожної платформи.

Для створення кроссплатформених додатків на платформі .NET зараз можна використати фреймворк MAUI.

NET MAUI об'єднує інтерфейси Android, iOS, macOS та Windows в єдиний API, що дозволяє розробникам працювати будь-де за допомогою єдиного входу та надає глибокий доступ до всіх аспектів кожної нативної платформи.

NET 6 і вище пропонує набір платформних фреймворків для створення додатків: .NET Android, .NET iOS, .NET macOS та бібліотека Windows UI 3 (WinUI 3). Всі ці фреймворки мають доступ до однієї і тієї ж бібліотеки базових класів .NET (BCL). Ця бібліотека абстрагує деталі базової платформи від вашого коду; BCL базується на середовищі виконання .NET і забезпечує середовище виконання для вашого коду; для Android, iOS і macOS це

середовище забезпечується Mono, додатком для виконання .NET, який забезпечує середовище виконання .NET CoreCLR; для Windows.

BCL дозволяє додаткам, що працюють на різних платформах, використовувати спільну бізнес-логіку, але різні платформи мають різні способи визначення користувацького інтерфейсу програми та способи зв'язку і взаємодії елементів інтерфейсу один з одним. Хоча можна створити окремий інтерфейс користувача для кожної платформи, використовуючи відповідний фреймворк для конкретної платформи (.NET Android, .NET iOS, .NET macOS або WinUI 3), такий підхід вимагає наявності кодової бази для кожного сімейства пристроїв, на яких необхідно підтримувати .NET MAUI.

NET MAUI надає єдиний фреймворк для створення користувацьких інтерфейсів для мобільних та десктопних додатків. На наступній схемі показано високорівневий погляд на архітектуру додатку .NET MAUI:

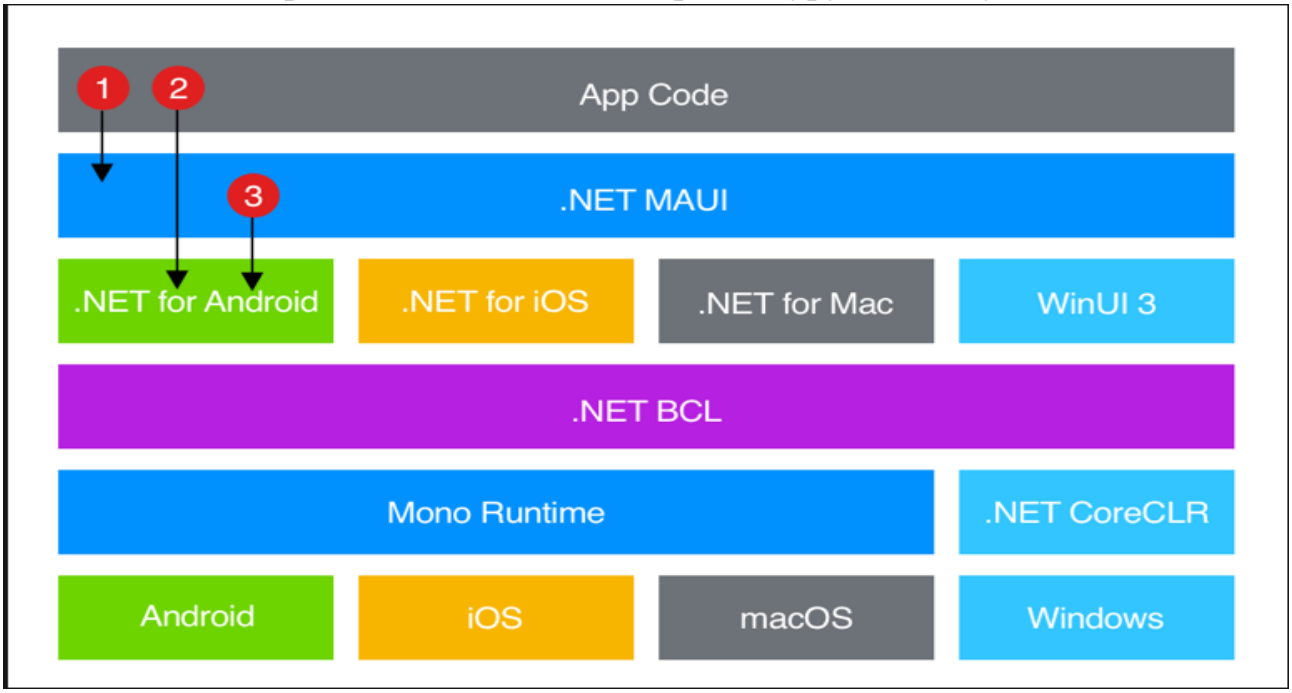

Рис 1.1 Схема роботи фреймворку .NET MAUI

1.4 Аналіз існуючих програмних рішень у цій сфері

На етапі систем управління логістичним центром підприємства доцільно буде розглянути існуючі на ринку аналоги, які можуть забезпечити

виконання подібного функціоналу та зрозуміти механіку їх роботи та слабкі місця.

Загалом, майже всі подібні програми є частинами TMS(Transport Management System) і є потужним інструментом, щополегшує бізнесі надає власнику більший контроль над витратами і прибутковістю послуг. Логістичне програмне забезпечення підтримує компанію в її повсякденній діяльності багатьма способами.

Перший приклад на ринку це веб-додаток: **fireTMS.**

Це система управління трансортом, що має багато переваг:

Логістичне планування:

1. графічне планування транспортних засобів і водіїв
2. розрахунок рентабельності маршруту
3. інструменти для підтримки оптимального використання автопарку
4. внутрішній список заблокованих контрагентів

Управління замовленнями:

1. експедиційно-транспортні замовлення
2. інтеграція GPS
3. моніторинг етапів виконання замовлення
4. спілкування з водіями через мобільний додаток

Фінанси і бухгалтерський облік:

1. автоматичне формування рахунку

2. інтеграція з бухгалтерським програмним забезпеченням
3. автоматичне стягнення боргу
4. аналіз вартості автопарку

Збереження часу

1. легкий доступ до документів
2. миттєвий пошук даних
3. автоматично створені звіти
4. панель для перевізника та клієнта

Другий приклад на ринку це **QguarTMS**

Qguar TMS – це широкий спектр можливостей для користувачів. Завдяки цій інтеграції:

1. користувачі можуть зручно спілкуватися з перевізниками та транспортними компаніями.
2. вантажовідправники можуть надіслати заявки своїм контрагентам безпосередньо та в цифровому форматі.
3. також є можливість скористатися доступом до спотового ринку (публічного та приватного) у ситуації, коли перевізник, з яким ви маєте договір, не може прийняти замовлення.

Це рішення значно скорочує час вибору перевізника та гарантує, що ви співпрацюєте з надійними компаніями. Варто зазначити, що кожен перевізник на Платформі Trans.eu уважно перевіряється та контролюється,

як зі сторони Trans.eu, так і іншими користувачами, які відкрито оцінюють та коментують співпрацю.

Платформа QguarTMS для вантажовласників це інструмент для гнучкого пошуку транспортних ресурсів та цифрового управління реалізацією перевезення. Платформа пропонує такі функції, як цифровий документообіг, та моніторинг. Наразі об’єднує найбільшу мережу транспортних субпідрядників з усієї Європи.

Quantum Qguar пропонує IT-системи для компаній з особливо високими вимогами у сфері управління ланцюгами постачання. Однією з систем, які пропонує Quantum, є Qguar TMS. Описана інтеграція дозволяє користувачам Qguar TMS , серед інших, публікувати пропозиції для перевізників на Платформі Trans.eu, узгоджувати умови та надсилати заявки. Зв’язок між системами здійснюється за найвищими правилами безпеки.

Висновки

У першому розділі даного дипломного проекту було детально описано, що таке система управління логістичним центром підприємства та був проведений аналіз сервісів, які присутні на українському ринку.

Був наданий детальний опис рейтингу логістичних центрів підприємств та рейтингів за якими ці системи управління можна порівнювати між собою. Клієнт під час вибору системи управління логістичним центром підприємства дивиться численні рейтинги та слухає рекомендації від своїх знайомих, щоб порівняти ці системи між собою, та отримати уявлення, що відбувається на ринку.

Була зроблена порівняльна характеристика різних типів додатків: мобільних, десктопних та веб. Була зроблена детальна порівняльна характеристика у вигляді таблиці.

Розробка система управління логістичним центром підприємства дозволить надати кінцевому користувачу програмний продукт, який спростить процеси управління логістикою підприємства та виділити час і персонал підприємства на вирішення інших задач.

РОЗДІЛ 2. Вимоги до системи управління логістичним центром підприємства

2.1 Вимоги до проектування системи управління логістичним центром підприємства

Розробка системи управління розподільчим центром є важливим етапом в забезпеченні ефективності та успішності логістичних операцій. Нижче наведені основні вимоги до проектування такої системи

1. Аналіз бізнес-процесів:

Системний аналіз: провести ретельний аналіз поточних логістичних процесів і визначити області для оптимізації та автоматизації.

Визначення цілей і завдань: зрозуміти бізнес-цілі компанії та визначити, як логістична система сприятиме досягненню цих цілей.

2. Інтеграція та інтероперабельність:

Інтеграція з іншими системами: забезпечення сумісності та обміну даними з іншими інформаційними системами підприємства (наприклад, ERP, CRM).

Підтримка різних форматів даних: можливість обробки та обміну даними в різних форматах для забезпечення гнучкості та інтероперабельності.

3. Технологічні аспекти:

Використання новітніх технологій: використання новітніх технологій, таких як IoT, штучний інтелект та аналітика даних для підвищення ефективності системи.

Масштабованість: забезпечення можливості легкого розширення системи в міру зростання логістичних операцій.

4. Автоматизація та оптимізація процесів:

Автоматизація операцій: заміна ручних завдань автоматизованими процесами для запобігання помилок і підвищення продуктивності.

Оптимізація маршрутів: визначення найкращих маршрутів для доставки та переміщення товарів.

5. Безпека та конфіденційність

Захист даних: забезпечення високого рівня захисту даних в логістичній системі від несанкціонованого доступу та збереження конфіденційності інформації про клієнтів.

Аудит і моніторинг: впровадження систем моніторингу та аудиту для виявлення можливих порушень безпеки та відстеження поведінки користувачів.

6. Простота використання та навчання

Інтерфейс користувача: створення інтуїтивно зрозумілого, простого у використанні інтерфейсу для різних категорій користувачів.

Навчання персоналу: навчання персоналу ефективному використанню системи та технологічним інноваціям.

7. Системи звітності

Звітність та аналіз: автоматична генерація звітів про логістичні операції та аналіз для прийняття стратегічних рішень.

Моніторинг ефективності: Впровадження системи моніторингу для визначення ефективності системи та її впливу на логістичні процеси.

Ці вимоги допомагають компанії побудувати ефективну та гнучку логістичну систему.

2.2 Функціональні вимоги

Функціональні вимоги є переліком дій та можливостей, що користувач може виконати і отримати за допомогою системи. Ключові дії доступні в системі управління логістичним центром підприємства.

1. Користувач повинен мати можливість знайти веб додаток по URL адресі в інтернеті.
2. Користувач повинен мати можливість знайти веб додаток на будь-якому пристрої за допомогою браузера.
3. Користувач повинен мати можливіть зараєстуватися в системі.
4. Даними при реєстрації повинні бути електронна пошта та пароль.
5. Коли користувач заходить у додаток у нього повинна бути можливість увійти до зареєстрованого аккаунту..
6. Користувач повиен бачити анімації завантаження, коли очікує будь які дії на сайті
7. Користувач повинен бачи інформацію про товари на складі підприємства.
8. Користувач повинен мати можливість відкрити певний товар та побачити більш детальну інформацію про нього.
9. Користувач повинен мати можливість бачити загальуі кількість вільного місця на складі.
10. Повинна бути система ролей при авторизації, щоб керівники складів могли вручну міняти кількість товару.
11. На головній сторінці користувач системи повинен бачити її назву та назву компанії для якої він залогінений.
12. Додаток повинен містити форму для вибору адреси доставки вантажів.
13. Якщо компанія користується сторонніми службами доставки, повинна бути інтеграція з їхніми API та можливосты доставки у їх відділення.
14. Повинна бути наявність функції повернення на головну сторінку.
15. Повинні бути наявні фільтри за якими можна оберати доставку: ціна, дата доставки.
16. У користувача повинна бути можливість залогінитися у власний кабінет.
17. Користувач повинен мати можливість вилогінитися з власного профілю.
18. Користувач повинен мати можливість вийти із програми.

19. У власному кабінеті повинна відображатися детальна інформація про користувача.
20. У користувача, на складі куди прибув відправлений товар повинна бути можливість прийняти його і внести в систему.
21. При відпраці товару повинна бути можливість застрахувати його.
22. При відпраці повинна бути можливість вибору страхової компанії.
23. Для кожної страхової компанії повинна бути можливість вибору страхового плану.
24. Якщо користувач вибирає сторонню логістичну компанію у додатку повинен відображатися її рейтинг .
25. Користувач повинен мати можливість зміни кінцевої адреси доставки товару, після його відправки.
26. При відправці товару, користувач повинен мати можливість бачити наближений час, коли товар буде доставлено.
27. Користувач повинен мати можливість передплати за послуги сторонньої логістичної компанії.
28. В кабінеті користувача повинен бути журнал витрат за логістичні послуги
29. Витрати за логістичні послуги повинні сортуватися за певними фільтрами.
30. Користува з відповідним рівнем доступу повинен мати можливість дивитися баланс компанії
31. Користува повинен мати можливість оплатити послуги логістичної компанії у зручний для нього спосіб.
32. Користувач повинен мати можливість подивитися детальний опис логістичної компанії
33. Користувач повинен мати можливість перейти на сайт логістичної компанії.
34. Користувач повинен мати можливість перейти до виклику техпідтримки у разі виникнення запитань
35. Користувач з адмінськими правами доступу повинен мати можливість створити нові склади у системі, що належать даній компанії
36. Додаток повинен мати можливість автоматичного страхування відправлень.

Рис 2.1 Діаграма використання для працівника служби доставки

Рис 2.2 Діаграма використання для користувача з підприємства

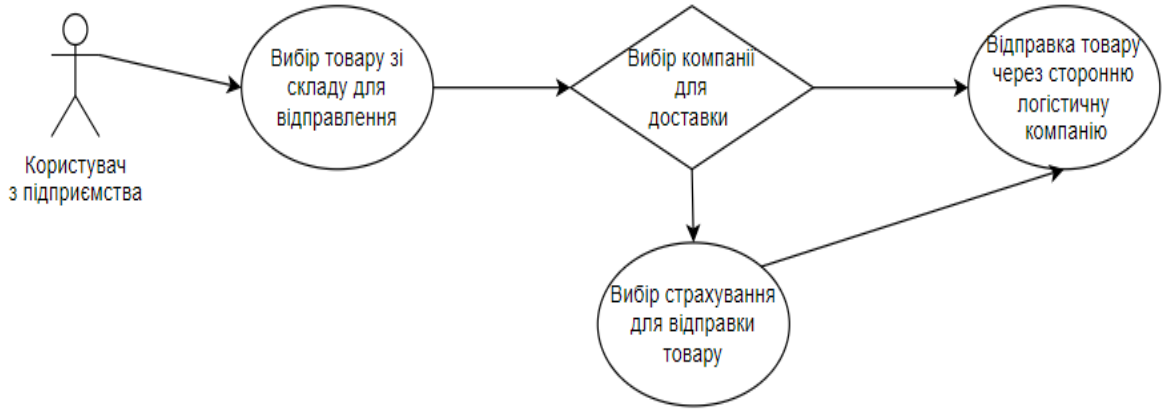

Рис 2.3 Діаграма використання для користувача підприємства, опція відпраки зі страхуванням

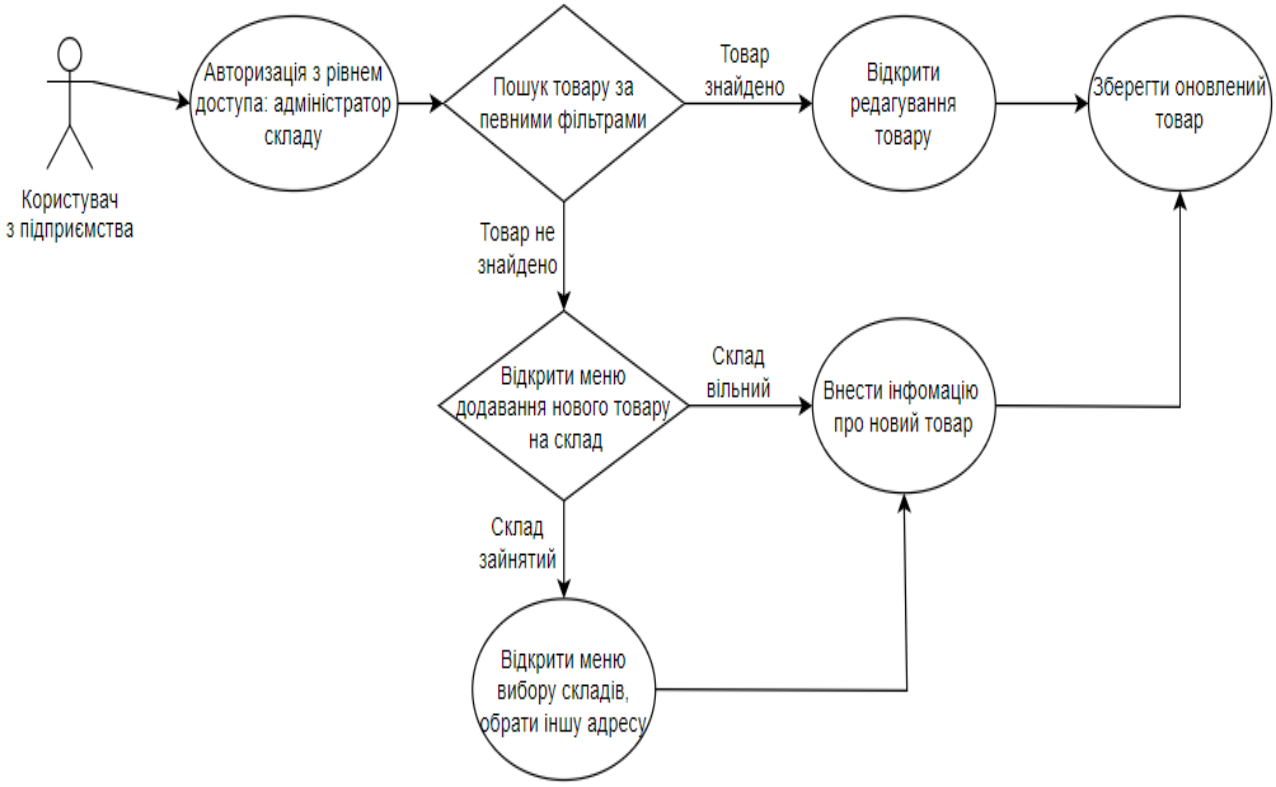

Рис 2.4 Діаграма використання для користувача підприємства, опція додавання/редагування товару

2.3 Не функціональні вимоги до проекту

Не функціональні вимоги до програмного забезпечення є важливою частиною процесу визначення та проектування, який визначає якість, надійність, продуктивність та інші атрибути системи, не тільки за її функціональністю, але й за якістю її функціонального виконання. Давайте заглибимося в різні аспекти нефункціональних вимог.

Головна програмна вимога до інформаційної системи– це мають бути WEB API, написані мовою C# у програмі JetBrains Rider. Користувацький інтерфейс повинен бути зроблений на веб фреймворку React. Для зберігання інформації повинні використовуватися реляційні бази даних.

1. Продуктивність

Швидкість: система повинна реагувати на дії користувача майже миттєво.Максимально допустимий час відгуку не повинен перевищувати заданих меж.

Пропускна здатність: система повинна забезпечувати достатню пропускну здатність для обробки очікуваного робочого навантаження. Пропускна здатність повинна бути масштабована по мірі збільшення навантаження.

Ефективність використання ресурсів: слід оптимально використовувати пам'ять, процесорний час та інші ресурси.

2. Безпека:

Безпека даних: конфіденційність, цілісність та доступність даних повинні бути гарантовані. Шифруйте конфіденційні дані та використовуйте захищені канали зв'язку.

Ідентифікація та автентифікація: система повинна забезпечити надійні методи ідентифікації та автентифікації користувачів.

Стійкість до атак: система повинна виявляйти та запобігати різним типам атак, наприклад, SQL-ін'єкціям, перехопленням сеансів тощо.

3. Надійність:

Стійкість до помилок: забезпечення того, що система виявляє та обробляє помилки належним чином, щоб гарантувати безперебійну роботу.

Стійкість до катастроф: забезпечення механізмів для швидкого відновлення після збоїв і відновлення стану системи.

Тестування: система повинна легко піддаватися тестуванню, а тести повинні бути надійними та відтворюваними.

4. Сумісність:

Сумісність з іншими системами: взаємодія та обмін даними з іншими додатками та платформами.

Сумісність з різними браузерами та пристроями: система працює коректно на різних браузерах і пристроях.

5. Простота використання та зручність:

Інтуїтивно зрозумілий інтерфейс: розробка інтуїтивно зрозумілого та простого у використанні інтерфейсу.

Доступність для людей з обмеженими можливостями: система доступна для всіх користувачів, незалежно від їхніх здібностей.

6. Розширюваність:

Вертикальна та горизонтальна масштабованість: здатність системи покращувати продуктивність за рахунок вертикального та горизонтального масштабування.

Модифікація та розширення вимог: легкість, з якою система може бути модифікована та розширена її функціональність.

7. Підтримка та обслуговування:

Легкість впровадження змін: можливість впроваджувати зміни та оновлення без значних витрат часу та ресурсів.

Простота усунення несправностей: швидке виявлення та усунення помилок.

8. Продуктивність:

Простота встановлення та оновлення: легке встановлення нових версій систем і компонентів.

Робота в різних операційних середовищах: забезпечує стабільну роботу на різних операційних системах та їхніх версіях.

9. Екологічні аспекти

Енергоефективність: мінімізує споживання енергії та зменшує вплив на навколишнє середовище.

Використання ресурсів: раціональне використання ресурсів з дотриманням екологічних норм.

Не функціональні вимоги до програмного забезпечення визначають характеристики та атрибути, які безпосередньо не пов'язані з функціональністю, а їх правильне визначення та реалізація визначають якість продукту та задоволеність користувачів, а також гарантують високу ефективність та конкурентоспроможність.

Висновки

У цьому розділі роботи описано специфікацію веб-додатку системи управління логістичним центром підприємства, виділено основні критерії опису програмного продукту та детально проаналізовано завдання, які необхідно виконати для створення веб-додатку логістичної системи.

У цьому розділі були представлені функціональні та нефункціональні вимоги до веб-додатку. Нефункціональні вимоги - це вимоги, які вказують на характеристики, які повинна мати система, або обмеження, яких необхідно дотримуватися, але не стосуються роботи самої системи. Функціональні вимоги - це вимоги, що описують дії, які повинен виконувати користувач під час роботи самої системи.

Складені специфікації можуть бути використані для планування подальшої розробки програмного забезпечення системи управління логістичним центром підприємства та шлях розвитку продукту.

Розділ 3. Структура системи управління логістичними центрами підприємства.

3.1 Архітектура системи управління логістичним центром підприємства

Архітектура розроблення програмного забезпечення являє собою спроектовану структуру застосунку і передбачає визначення взаємодії внутрішніх процесів та інтерфейсних компонентів застосунку. Простіше кажучи, це певний підхід, що визначає, які функції за що відповідають і як вони взаємодіють.

Точного розуміння і чіткого формулювання цього процесу не існує. Основне завдання - створити логічну структуру застосунку і спростити взаємодію між розробниками. Це дає змогу в майбутньому вносити зміни в додаток, зачіпаючи конкретні аспекти, а не переробляти все програмне забезпечення. Архітектура і дизайн програмного забезпечення забезпечують виконання додатком своїх завдань і дотримання цілей, визначених на ранній стадії розробки.

Основна ідея архітектури полягає в зниженні складності системи за рахунок розподілу відповідальності та створення чіткої структури. Архітектура і дизайн ПЗ дають змогу програмістам створити чітку структуру, у межах якої їм зручно працювати. Від її якості залежить легкість супроводу, модифікації, доповнення та підтримки програмного забезпечення.

Структура програмного забезпечення. Внутрішня частина може бути запущена на сервері під управлінням операційної системи Windows/Linux. Фронтенд-частина є веб-додатком, тому користувачі можуть відкрити його на будь-якій операційній системі за допомогою браузера.

Керівник. Вовк В.Г

Нормконтролер Радішевський М.Ф

ПІ-222М

Для написання системи буде використовуватися об’єктно орієнтований стиль програмування. Об'єктно-орієнтоване програмування зробило революцію в написанні коду. Від великих корпоративних програм до невеликих додатків, ООП забезпечує потужний і гнучкий підхід до розробки. Завдяки об'єктній орієнтації розробники мислять в термінах об'єктів, класів і методів, що робить код більш інтуїтивно зрозумілим і легшим в обслуговуванні. Крім того, ООП дозволяє розробникам створювати ефективний і багаторазовий код, скорочуючи час і витрати на розробку.

Завдяки основним принципам ООП (поліморфізм, інкапсуляція, успадкування та абстрагування), ООП полегшує життя розробникам. Один і той самий інтерфейс можна використовувати для роботи з різними спеціальними типами даних, для кожного типу даних можна запровадити окремі функції та окремі дії, а завдяки успадкуванню зміни в рядках коду можна замінити новими розширеннями або встановити нові значення параметрів для конкретних об'єктів.

1. Це скорочує час розробки, який можна використати для інших завдань.
2. Повторно використовувані компоненти, як правило, набагато менш схильні до помилок, ніж новостворені, оскільки вони вже були протестовані багато разів.
3. Якщо компонент використовується декількома клієнтами одночасно, поліпшення його коду матиме позитивний вплив на багато програм, що працюють на ньому одночасно.
4. Якщо додаток базується на стандартизованому компоненті, його структура та користувацький інтерфейс будуть більш одноманітними, легшими для розуміння та використання.

Сама архітектура системи буде **мікросервісною.**

Мікросервіси - це найпростіші одиниці, це сервіси, які приймають вхідні запити на виконання дій; це можуть бути бекенд-сервіси, доступні 24/7 або функції, які викликаються при настанні певної події. Простіше кажучи, це функція або набір функцій, до яких можна отримати доступ через мережу за допомогою певного API. Іншими словами, це бекенд-сервіс, розгорнутий на сервері. У певному сенсі це монолітний додаток. Однак він несе не всю функціональність системи, а лише невелику частину логіки. На відміну від монолітів, результуючий додаток будується як набір відносно невеликих, незалежних сервісів, які називаються "мікросервісами", що взаємодіють

через комп'ютерну мережу. Мікросервіси можна описати як ті ж самі логічні модулі, що і монолітні додатки, але замість того, щоб працювати в рамках одного процесу (пристрою), вони розподілені по комп'ютерній мережі.

Мікросервіси зазвичай структуровані ярусами. Не існує правил щодо того, скільки і яких ярусів має бути, однак, є найкращі практики. На діаграмі вище можна побачити найпоширеніші яруси, що використовуються в подібних архітектурах. Назви можуть відрізнятися, але структура дуже схожа на класичну багатоярусну архітектуру. Логічні яруси, які ми бачимо:

- Front-End — Client Application, Static Content
- Back-End — API Gateway, Experience Microservices, Domain Microservices
- Data & Integration — Data Stores, Integration Interfaces / Connectors

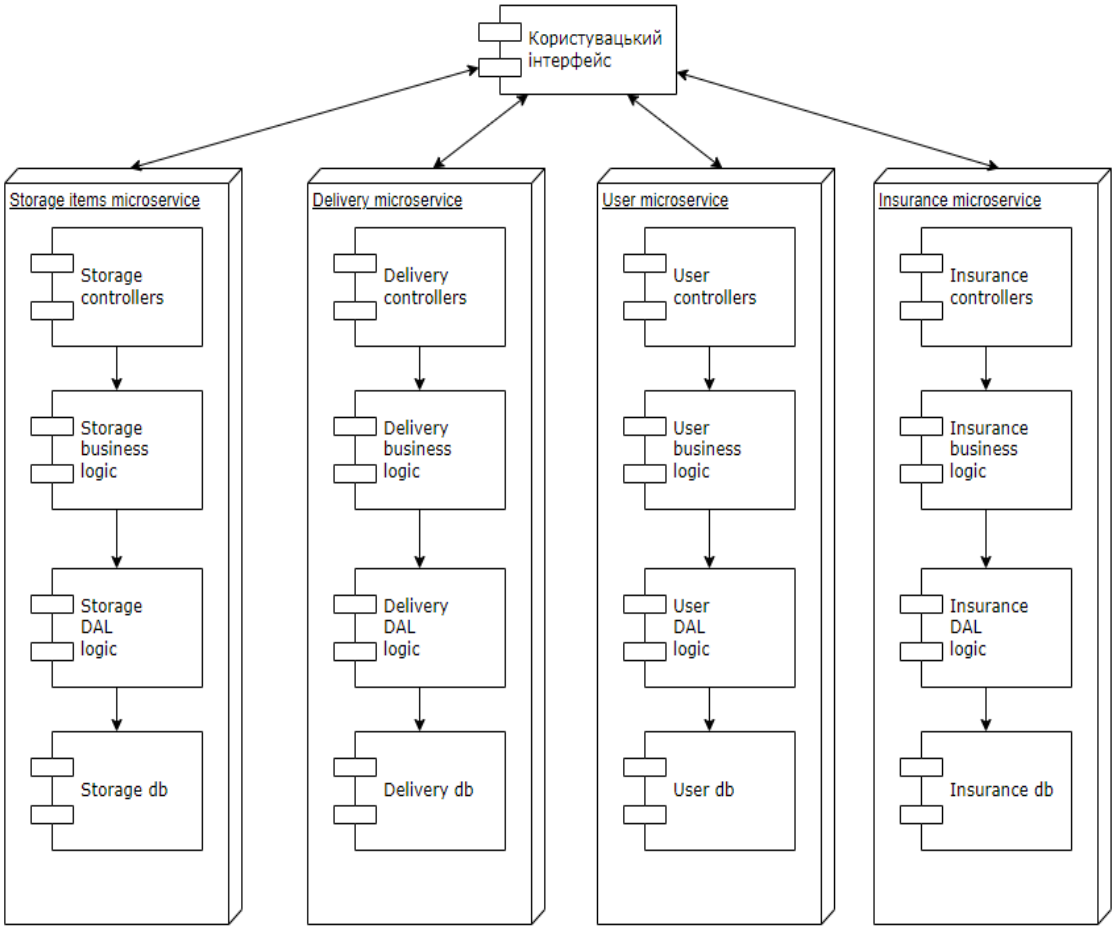

Рис 3.1 Діаграма компонентів системи управління логістичним центром підприємства

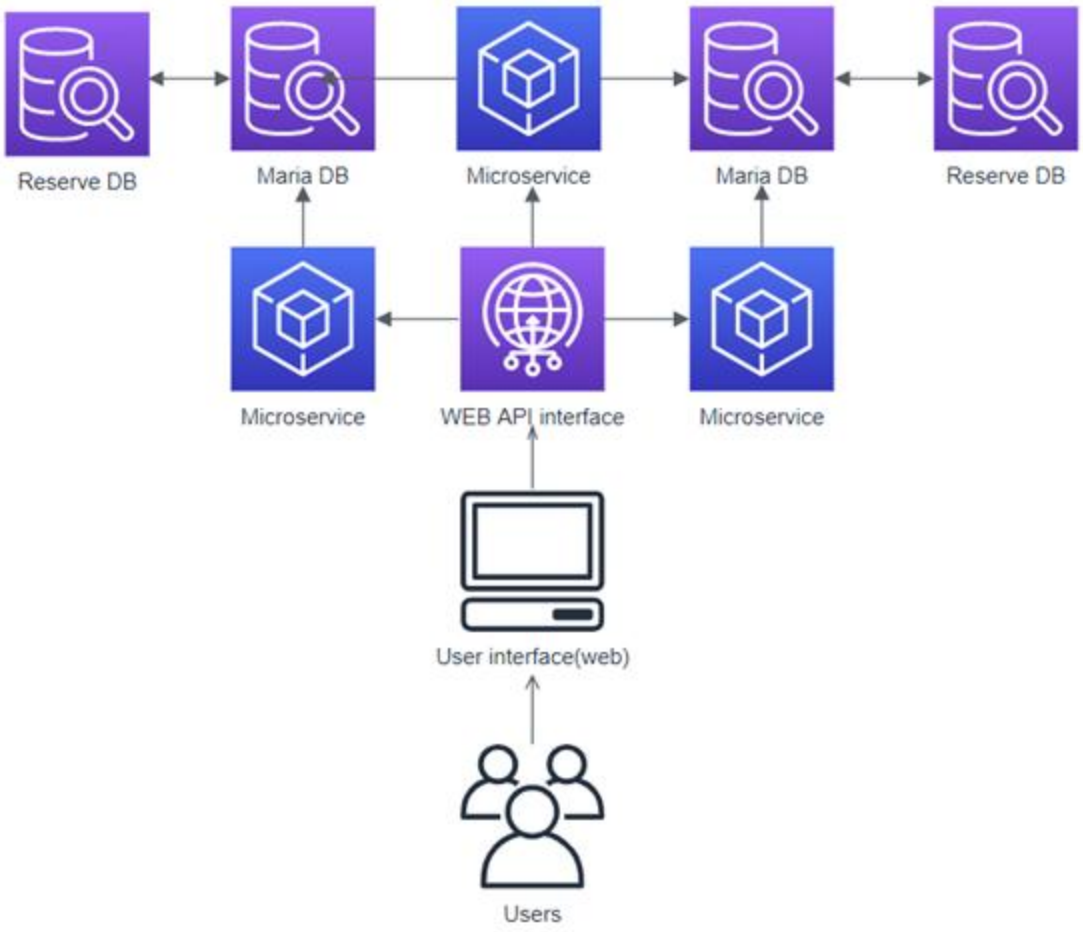

Рис 3.2 Діаграмма розгортання системи

Використання мікросервісів принесе наступні переваги:

1. Гнучкість і швидкість розгортання:

Гнучкість: Сервіси можна розгортати та розширювати незалежно один від одного. Різні технології та мови програмування можуть використовуватися для різних послуг.

Швидкість розгортання: окремі сервіси можна швидко розгортати, не впливаючи на інші частини системи. Нові функції та зміни можуть бути впроваджені негайно.

2. Незалежність і масштабованість:

Незалежність сервісів: незалежність розробки, тестування та розгортання кожного окремого сервісу. Тільки необхідні сервіси можуть бути оновлені та розширені.

Розширюваність: розширювати можна лише ті сервіси, які потребують додаткових ресурсів. Лише певні сервіси можуть бути покращені з точки зору продуктивності та навантаження.

3. Висока доступність та безвідмовність:

Немає єдиної точки відмови: відмова або збій в одному сервісі не впливає на інші сервіси. Підвищує відмовостійкість і забезпечує високу доступність.

Спрощене відновлення: окремі сервіси можуть бути легко відновлені у разі збою або відключення. Є можливість використання механізмів балансування навантаження та резервного копіювання.

4. Технологічна різноманітність:

Використання різних технологій: можливість використовувати різні технології та мови програмування для різних сервісів у системі. Це підвищує ефективність і дозволяє використовувати найбільш підходящі інструменти для кожної конкретної задачі.

5. Простота обслуговування та розвитку:

Простота обслуговування: легкість виявлення та виправлення помилок у конкретних сервісах. Зменшує взаємозалежності та полегшує налагодження і тестування.

Простота розробки: легкість додавання нового функціоналу та оновлення окремих сервісів. Це зменшує ризик впровадження змін у великих монолітних системах.

6. Зручність віддаленої розробки:

Віддалена розробка: легкість віддаленої розробки та співпраці, оскільки розробка зосереджена на окремих сервісах. Це забезпечує робочі процеси з невеликими розподіленими командами.

Архітектура мікросервісів дозволяє створювати гнучкі, масштабовані та високопродуктивні додатки, які відповідають вимогам сучасного ринку

програмного забезпечення. Однак важливо пам'ятати, що їх впровадження також вимагає певних витрат на планування та управління.

Комунікація з інтерфейсом буде виконуватися за допомогою інтерфейсу **REST web api.**

REST (скорочення від Representational State Transfer - передача представницького стану) - це підхід до архітектури мережевих протоколів, що забезпечує доступ до інформаційних ресурсів, визначений у 2000 році Роєм Філдінгом, одним з винахідників протоколу HTTP. REST базується на принципах Всесвітньої павутини, зокрема на функціональності HTTP. Філдінг розробляв REST паралельно з HTTP 1.1, який базується на більш ранньому протоколі HTTP 1.0.

Дані повинні передаватися в невеликій кількості стандартних форматів (наприклад, HTML, XML, JSON); протоколи REST (включаючи HTTP) повинні підтримувати кешування, бути незалежними від мережевого рівня і не зберігати інформацію про стан між парами запиту і відповіді. Стверджується, що такий підхід забезпечує масштабованість системи і дозволяє еволюціонувати відповідно до нових вимог.

Аналогом REST є підхід, заснований на віддаленому виклику процедур (RPC), де велика кількість методів і складних протоколів може використовувати невелику кількість мережевих ресурсів, тоді як в підході REST кількість методів і складність протоколів строго обмежена, тому кількість окремих ресурсів повинна бути великою.

REST - це архітектурний стиль розподіленої гіпертекстової системи.

Інтерфейси прикладного програмування (**API**) - це готові конструкції мови програмування, які дозволяють розробникам створювати складну функціональність з меншими зусиллями; API "приховують" більш складний код від програміста і гарантують простоту використання.

Для кращого розуміння розглянемо аналогію з домашньою електромережею. Коли ви хочете скористатися електричним приладом, ви просто вмикаєте його в розетку, і все працює. Ніхто не намагається під'єднати дроти безпосередньо до джерела живлення. Це марно, складно і небезпечно, якщо ви не є інженером-електриком.

Аналогічно, якщо ви хочете запрограмувати, наприклад, 3D-графіку, набагато простіше використовувати API, написані мовами високого рівня, такими як JavaScript або c#.

JavaScript має низку клієнтських API, серед інших. Вони не є частиною мови, але побудовані з використанням вбудованих функцій JavaScript, щоб надати вам більше можливостей при написанні коду. Вони діляться на дві категорії:

Браузерні API вбудовані у веб-браузери і дозволяють виконувати більш складну обробку, використовуючи дані з браузера і комп'ютерного середовища. Наприклад, API веб-аудіо надає структуру JavaScript для обробки аудіо в браузері. На практиці для обробки аудіо в браузері виконується складний низькорівневий код (наприклад, C++ або Rust), але, як згадувалося вище, ці деталі приховані API.

Сторонні API не вбудовані в стандартні браузери. Такі API та інформацію про них зазвичай доводиться шукати в інтернеті. Наприклад, API Twitter дозволяє розміщувати останні твіти на веб-сайті. API визначає ряд конструкцій, які роблять запит до сервісу Twitter і повертають певні дані.

Реляційна база даних - це база даних, в якій вся інформація зберігається в таблицях, пов'язаних між собою спеціальними зв'язками. Ці зв'язки дозволяють отримувати та об'єднувати дані з однієї або декількох таблиць за допомогою одного запиту дані можуть бути отримані та об'єднані за допомогою одного запиту. Бази даних, побудовані на основі реляційної моделі. У реляційній базі даних кожен об'єкт ідентифікується відповідним записом (рядком) у таблиці. Реляційні бази даних створюються та управляються за допомогою системи управління реляційними базами даних.

Інформація, що зберігається у двовимірних таблицях. Зв'язки між таблицями можуть бути відображені в структурі даних або існувати неявно, тобто на неформатованому рівні. Вони існують на неформатованому рівні. Кожна таблиця бази даних представлена у вигляді набору рядків і стовпців, де рядки відповідають екземплярам об'єктів, певним подіям чи явищам, а стовпці - атрибутам (ознакам, властивостям, параметрам) об'єктів, подій чи явищ.

Атрибути (ознаки, властивості, параметри) відповідають атрибутам (ознакам, властивостям, параметрам) об'єктів, подій і фактів. Реляційні бази даних

полегшують доступ до звітів (зазвичай через SQL) і підвищують надійність і цілісність даних, оскільки немає надлишкової інформації.

Рис 3.3 Схема таблиці у реляційних базах даних

Реляційна база даних - це набір таблиць (сутностей). Таблиця складається зі стовпців і рядків (кортежів). Усередині таблиць можуть бути визначені обмеження, а між таблицями існують зв'язки.

При побудові інформаційної системи набір зв'язків дозволяє зберігати дані про об'єкти предметної області та моделювати зв'язки між ними.

Зв'язки є найважливішим поняттям і являють собою двовимірні таблиці, що містять певні дані.

Сутності - це об'єкти будь-якої структури, дані про які зберігаються в базі даних. Дані про сутності зберігаються у зв'язках.

Атрибути - це властивості, які характеризують сутність. У структурі таблиці кожен атрибут має ім'я і відповідає певному заголовку стовпця таблиці За допомогою SQL можна виконувати запити, які повертають набори даних, отримані з однієї або декількох таблиць Один запит об'єднує декілька таблиць для отримання даних (JOIN). У багатьох випадках

Для об'єднання використовуються ті самі стовпці, які визначають зв'язок між таблицями.

Нормалізація - це процес структурування моделі даних для забезпечення узгодженості та надлишковості даних. Метою нормалізації реляційних баз даних є усунення дефектів у структурі бази даних, які призводять до надмірності.

Надмірність може призводити до різних аномалій і порушень цілісності даних. У процесі розвитку цієї теорії теоретики реляційних баз даних визначили типові приклади надмірності та пояснили, як їх усунути. Реляційне сховище пропонує простоту, гнучкість, продуктивність і відмовостійкість.

Вони пропонують найкраще поєднання довговічності, гнучкості, продуктивності, масштабованості та інтероперабельності. Що стосується масштабованості, то реляційні бази даних добре масштабуються, тільки якщо вони знаходяться на одному сервері.

Сама архітектура бек-енд частини буде написана у «Цибулевому» стилі.

Підхід ієрархізації коду програми відповідно до функціональності та призначення відомий як цибулинна архітектура. У цій моделі навколо центральної моделі домену будуються концентричні кола або шари, кожен з яких відповідає за окреме завдання і має залежності від ядра.

Інфраструктура додатку та інтерфейс користувача представлені у зовнішніх шарах додатку, тоді як основна логіка домену додатку представлена у верхньому шарі.

Архітектура Onion має велике практичне значення, особливо при створенні великих, складних програмних систем. Шляхом пошарової побудови додатків бізнес-логіка може бути відокремлена від шарів зображення та інфраструктури, що полегшує тестування, підтримку та оновлення кодової бази.

Крім того, така модульність дозволяє розробникам змінювати частини або технології, не впливаючи на інші компоненти системи.

Шари цибулинної архітектури

Цибулева архітектура базується на концепції концентричних шарів, кожен з яких має свою функцію і взаємодіє з іншими шарами у чітко визначений спосіб. Нижче наведені різні шари цибулинної архітектури та властивості, які вони містять

Рівень домену. Тут знаходиться логіка домену, що лежить в основі програми, і це найглибший рівень цибулинної архітектури. Він описує структури даних, моделі та сутності, які визначають бізнес-домен додатку.

Реалізація бізнес-правил, валідація та інші критичні функції, які складають основну функціональність додатку, є відповідальністю доменного рівня. Відокремлення логіки домену від інших рівнів полегшує тестування та обслуговування.

Прикладний рівень. Прикладний рівень знаходиться між доменним та інфраструктурним рівнями. Варіанти використання, директиви та інші елементи складають прикладну логіку, яка виконує бізнес-логіку додатку. Прикладний рівень взаємодіє з доменним рівнем для виконання своїх функцій.

Він також взаємодіє з інфраструктурним рівнем для читання і запису даних. Крім того, цей рівень надає API, який інфраструктурний рівень може використовувати для отримання бізнес-вимог, і відповідає за перетворення цих вимог у придатний для використання код.

Інфраструктурний рівень. Рівень, який взаємодіє із зовнішніми об'єктами, такими як бази даних, API та зовнішні сервіси, називається інфраструктурним. Він взаємодіє з доменним рівнем через інтерфейси і забезпечує реалізацію інтерфейсів, визначених прикладним рівнем.

Зберігання даних, мережеве підключення та безпека - це лише деякі з функцій, які цей рівень враховує при підключенні до зовнішніх ресурсів. Інфраструктурний рівень можна модифікувати або додавати нові функції, не впливаючи на решту програми.

Рівень представлення. Інтерфейс користувача програми складається з представлень і контролерів, а рівень представлення відповідає за управління ними. Рівень представлення взаємодіє з прикладним рівнем для отримання та налаштування даних, а також для керування введенням і виведенням даних користувачем.

Рівень представлення співпрацює з прикладним рівнем для виконання завдань і представлення даних у спосіб, зрозумілий кінцевому користувачеві. Рівень представлення повинен бути відокремлений від інших рівнів, щоб користувацький інтерфейс можна було легше змінювати і щоб можна було зберегти кодову базу.

Розробка програмного забезпечення базується на наборі базових ідей, які формують цибулинну архітектуру. Ці принципи гарантують, що кодова база є модульною, тестованою та підтримуваною з часом. Провідні ідеї цибулинної архітектури є наступними

1. Ізоляція проблеми: Ця ідея передбачає поділ різних функціональних компонентів програми на окремі модулі або шари. Кожен шар повинен бути незалежним від інших, оскільки кожен з них відіграє свою роль. Такий поділ полегшує тестування, підтримку та оновлення кодової бази.
2. Концентричні шари: в цибулинній архітектурі шари додатку розташовуються концентричними колами навколо центральної моделі домену. Бізнес-логіка додатку розміщується в найглибшому шарі, замінюючи модель домену. Інтерфейс користувача та інфраструктура додатку розміщуються у зовнішніх шарах.
3. Незалежність шарів: Шари цибулинної архітектури повинні бути незалежними один від одного. Іншими словами, щоб шар працював ефективно, він не повинен залежати від іншого шару. Натомість, кожен шар повинен бути незалежним від інших і мати чітко визначені інтерфейси.
4. Ін'єкція залежностей: Залежності між шарами в цибулинній архітектурі управляються за допомогою техніки проектування, відомої як ін'єкція залежностей. Це надає залежності компонентам замість того, щоб компоненти самі створювали залежності. Ця стратегія робить кодову базу більш гнучкою та адаптивною.
5. Модульне тестування: Важливою частиною архітектури Onion є модульне тестування. Кожен рівень повинен бути спроектований так, щоб полегшити тестування. Це означає, що кожен рівень повинен мати чітко визначену взаємодію з іншими рівнями і бути незалежним від зовнішніх ресурсів, таких як бази даних і API. Надійність і коректність кодової бази забезпечується юніт-тестами.

Переваги Onion-архітектури, ця архітектура має багато переваг як для компаній, так і для розробників. Деякі з основних переваг цибулинної архітектури

Масштабованість. Модульна структура, якій надає перевагу архітектура Onion, дозволяє легко розширювати додаток. Дизайн побудований навколо базового рівня домену, який містить бізнес-логіку додатку і оточений іншими рівнями, які обробляють різні частини додатку.

Завдяки модульній архітектурі додаток можна легко розширювати за рахунок додаткової функціональності та продуктивності, не впливаючи на основний рівень домену.

Підтримувати загальний дизайн також простіше, оскільки обов'язки між рівнями чітко розподілені. Це означає, що зміни на одному рівні не вимагають змін на інших рівнях.

Простота тестування. Тестування архітектури Onion є однією з її головних переваг. Оскільки архітектура заохочує поділ завдань, легко тестувати кожен рівень окремо.

Розбиваючи додаток на невеликі незалежні компоненти, розробники можуть створювати модульні тести, які перевіряють функціональність кожного компонента. Це не тільки гарантує, що додаток працює належним чином, але й полегшує пошук та виправлення помилок.

Підтримка. Модульна та декомпонована архітектура, запропонована архітектурою Onion, полегшує довгострокову підтримку додатків. Кожен рівень має незалежну функціональність і взаємодіє з іншими рівнями через чітко визначені інтерфейси, що дозволяє розробникам вносити зміни в один рівень, не впливаючи на інші.

В результаті вони можуть легко адаптуватися до мінливих потреб бізнесу без необхідності повністю переписувати прикладне програмне забезпечення.

Гнучкість. Адаптивна архітектура Onion дозволяє розробникам модифікувати додаток, не впливаючи на інші компоненти системи. Кожен рівень є незалежним сам по собі і взаємодіє з іншими рівнями тільки через чітко визначені інтерфейси, що дозволяє розробникам змінювати або оновлювати компоненти без зміни інших компонентів системи.

Це означає, що їм не потрібно турбуватися про базову технологію і вони можуть адаптуватися до мінливих ринкових умов і вимог клієнтів. Недоліки

Архітектура Onion - це потужне програмне забезпечення, яке пропонує багато переваг, але воно не позбавлене недоліків. Деякі обмеження архітектури Onion перераховані нижче:

Підвищена складність: Цибулева архітектура може збільшити складність додатків, що є одним з її недоліків. Розробникам доводиться керувати більшою кількістю коду і мати справу з підвищеною складністю організації взаємодії між рівнями в результаті розбиття програми на менші, більш модульні компоненти.

Крута крива навчання: Архітектура Onion може бути складною для освоєння розробниками, які не знайомі з рекомендаціями та кращими практиками проектування. Розробники повинні знати, як правильно реалізувати шари та інтерфейси архітектури, щоб зробити додатки надійними, керованими та масштабованими.

Накладні витрати на продуктивність: Додаткові шари та інтерфейси в архітектурі Onion можуть погіршити продуктивність програми. Додатковий код і взаємодія між рівнями можуть погіршити продуктивність програми.

Надмірне проектування: Використання цибулинної архітектури збільшує ймовірність того, що розробники будуть надмірно проектувати додаток. Розробники ризикують створювати надто складні та заплутані проекти через акцент на модульність та поділ завдань.

Збільшення часу розробки: Реалізація цибулинної архітектури може зайняти більше часу та зусиль, ніж інші проекти, з точки зору часу та зусиль на розробку. Затримки в циклі розробки можуть виникати через те, що шари та інтерфейси в архітектурі повинні бути належним чином сплановані та спроектовані розробником.

3.2 Вибір інструментів для створення веб додатка

Для побудови даної системи буде використаний наступний стек технологій. База даних – mariaDB, мова програмування – c#, середовище розробки – JetBrains Rider, веб-фреймворк – ASP.NET, користувацький інтерфейс – React.

C# - це мова програмування високого рівня, яка працює в середовищі Common Language Runtime (CLR). Вона має строгу статичну типізацію і підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класу, атрибути, події, властивості, винятки та коментарі у форматі XML. C# відрізняє C++, Object Pascal, Modules, Smalltalk та багато інших моделей від своїх попередників і, виходячи з практики використання, усунула деякі моделі, які виявилися проблематичними при розробці програмних систем. Наприклад, C#, на відміну від C++, не забезпечує множинного успадкування класів .

ASP.NET - програмний інструмент Microsoft, що дозволяє створювати веб-додатки та веб-сервіси. Він є частиною фреймворку Microsoft .Net. Веб-сторінки ASP.NET, формально відомі як веб-форми, є додатками в ASP.NET Хоча веб-додатки повинні бути скомпільовані перед розгортанням, структура веб-сайту дозволяє користувачам створювати веб-додатки без необхідності попередньої компіляції файлів, існує два основних методи створення веб-форм: форма веб-додатку та форма веб-сайту, які копіюються безпосередньо на сервер. Веб-форми містяться у файлах з розширенням ".aspx". Ці файли зазвичай містять статичну HTML-розмітку. Компонентна розмітка може включати елементи керування на стороні сервера і елементи керування на стороні користувача, визначені в каркасі веб-сторінки.

.NET використовує техніку створення "відвідуваних компонентів". Під час компіляції файл шаблону (.aspx) компілюється в код ініціалізації, який створює дерево елементів керування (композит), що представляє оригінальний шаблон. Фактичний текст перетворюється в екземпляри фактичних класів елементів управління, а елементи управління сервера перетворюються в екземпляри конкретних класів елементів управління. Код ініціалізації об'єднується з кодом, написаним користувачем (зазвичай він поєднує кілька підкласів), і передається до конкретного класу сторінки.

Власне запит до сторінки обробляється в кілька кроків. По-перше, під час ініціалізації створюється екземпляр класу сторінки і виконується код

ініціалізації. Це створює дерево керування ініціалізацією, яким маніпулює метод page на наступному кроці. Оскільки кожен вузол дерева є елементом управління, представленим як екземпляр класу, код може не тільки змінювати структуру дерева, але й маніпулювати властивостями та методами окремих вузлів. Нарешті, на етапі рендерингу користувач (відвідувач) повинен відвідати кожен вузол дерева, і кожен вузол повинен використати метод відвідувача. Вихідний HTML надсилається клієнту.

Після обробки запиту екземпляр класу сторінки знищується, а разом з ним знищується і все дерево елементів управління. Це створює труднощі для початківців ASP.NET програмістів, які мають доступ до членів знищеного екземпляру класу в кожному циклі запиту відповіді сторінки.

Entity Framework (EF) — структура об'єктно-реляційного відображення (ORM) з відкритим кодом для ADO.NET. Це набір технологій ADO.NET, які підтримують розробку програмних додатків, керованих даними. Архітектори та розробники додатків на основі даних часто стикаються з необхідністю досягнення двох дуже різних цілей. Необхідно змоделювати сутності, взаємозв'язки і логіку бізнес-проблеми, яку вони намагаються вирішити, а також працювати з механізмом даних, який використовується для зберігання і вилучення даних. Дані можуть охоплювати декілька систем зберігання, кожна з яких має власні протоколи. Навіть додатки, що працюють на одній системі зберігання даних, повинні балансувати між вимогами системи зберігання даних і вимогами написання ефективного і зручного для підтримки коду програми.

Архітектура структури сутностей ADO.NET складається з наступних елементів, починаючи з самого низу:

1. **Постачальники джерел даних**, які абстрагують ADO.NET. Провайдери джерел даних, які абстрагують інтерфейси .NET.
2. **Провайдер відображення** (Mapping Provider) - це провайдер даних, який перетворює дерева команд Entity SQL у спеціальні запити до бази даних SQL. Це компонент, який перетворює загальне дерево команд у дерево команд конкретного сховища.
3. **Синтаксичний аналізатор** і переглядач EDM дозволяє програмувати на основі концептуальної моделі, використовуючи властивості моделі даних SDL і те, як ця модель даних відображається на базову реляційну

модель. Створює подання даних, які відповідають концептуальній моделі, використовуючи реляційну схему. Об'єднує інформацію з кількох таблиць для створення сутностей і розділяє оновлення сутності на кілька оновлень таблиць, які сприяють створенню цієї сутності. Конвеєр запитів і оновлень обробляє запит, фільтрує і оновлює запит, перетворюючи його на звичайне дерево команд.

4. **Служби** метаданих для обробки всіх метаданих, пов'язаних з сутностями, зв'язками і відображеннями.
5. **Транзакції** для інтеграції з транзакційною функціональністю базового сховища. Якщо базове сховище не підтримує транзакції, підтримка реалізується на цьому рівні. Виконання для отримання програмної моделі для кодування відповідно до концептуальної схеми. Використовує об'єкти ADO.Connection для посилання на провайдерів відображення, використовує командні об'єкти для надсилання запитів і повертає EntityResultSets або EntitySets, що містять результати. Компонент, який локально кешує набори даних і набори сутностей для використання ADO.NET Entity Framework. Компоненти, які локально кешують набори даних та набори сутностей для використання у середовищах, від яких залежить .NET Entity Framework.
6. **Вбудовані**: ADO.NET Entity Framework включає легку вбудовану базу даних для кешування на стороні клієнта і запитів до реляційних даних.
7. **Інструменти проектування**, такі як Mapping Designer, також включені в ADO.NET Entity Framework. Це спрощує завдання відображення концептуальних діаграм на реляційні діаграми та визначення того, які властивості типу сутності відповідають яким таблицям у базі даних.
8. **Рівень програмування** представляє ЕЦП як програмну конструкцію, яку можна використовувати в мовах програмування. Служби об'єктів автоматично генерують код для класів CLR, які надають ті самі властивості, що й сутності. Екземпляри сутностей можна створювати як об'єкти .NET.
9. **Веб-сервіс**, високорівневі служби, такі як служби звітності, які працюють з сутностями, а не з реляційними даними.

React (раніше відомий як React.js та ReactJS) - це бібліотека JavaScript з відкритим вихідним кодом для побудови користувацьких інтерфейсів,

призначена для вирішення проблеми часткового оновлення вмісту веб-сторінок, з якою стикаються при розробці односторінкових додатків. Розроблена компанією Meta (раніше Facebook) та спільнотою індивідуальних розробників".

React дозволяє розробникам створювати великі веб-додатки, які використовують дані, що змінюються з часом, без перезавантаження сторінок. Мета React - бути швидким, простим і масштабованим. Він відповідає моделі Model-View-Controller (MVC) і може використовуватися з іншими бібліотеками JavaScript та масштабними MVC-фреймворками, такими як AngularJS . Його також можна комбінувати з фреймворками на основі плагінів React, щоб подбати про безфронтендну частину створення веб-додатків. Як бібліотека користувацького інтерфейсу, React здебільшого використовується в поєднанні з іншими бібліотеками, такими як Redux.

Наразі React використовують Khan Academy, Netflix, Yahoo, Airbnb, Sony, Atlassian та інші

Можливості React:

Одностороння передача даних: Властивості передаються рендереру компонента як властивості html-тегів. Компонент не може змінювати передані властивості безпосередньо, але може змінювати їх за допомогою функцій зворотного виклику. Цей механізм називається "властивість вниз, подія вгору".

Віртуальний DOM: react підтримує віртуальний DOM замість того, щоб покладатися виключно на DOM браузера. Це дозволяє бібліотекам визначати, які частини DOM змінилися, порівнюючи їх зі збереженою версією віртуального DOM, і визначати найбільш ефективний спосіб оновлення DOM браузера . Таким чином, бібліотека визначає, які саме компоненти сторінки потрібно оновити, а програміст працює, виходячи з припущення, що оновиться вся сторінка в цілому.

JSX: компоненти React зазвичай пишуться на JSX ; код, написаний на JSX, компілюється у виклики методів у бібліотеці React. Розробники також можуть писати на чистому JavaScript; JSX схожий на XHP, іншу мову, створену Facebook для розширення PHP.

Більше, ніж просто рендеринг HTML у браузері: react не лише відображає HTML у браузері. Наприклад, Facebook має динамічну графіку,

яка рендериться в теги<canvas>; Netflix та PayPal використовують ізоморфне завантаження для рендерингу однакового HTML на сервері та клієнті .

Методи життєвого циклу: методи життєвого циклу - це різні техніки, вбудовані в ReactJS. Вони дозволяють розробникам обробляти дані на різних етапах життєвого циклу React-додатку. Наприклад

Атрибути: JSX надає набір атрибутів елементів, призначених для відображення атрибутів, наданих в HTML. Користувацькі атрибути також можуть бути передані компонентам. Всі атрибути передаються компоненту як пропси.

3.3 Вибір інструментів для тестування веб-додатка

У розробці програмного забезпечення можуть використовуватися такі методи та техніки веб-тестування залежно від функції, призначення продукту та вимог замовника

Функціональне тестування - це процес, який передбачає перевірку основних елементів програми, а саме: інтерфейсу користувача, API, бази даних і рівня безпеки, а також продуктивності сервера та інших параметрів. Функціональне тестування виконується наступним чином. З використанням заздалегідь розроблених тестових кейсів;

Дослідницьке тестування - також звертається увага на правильність функціонування внутрішніх посилань, зовнішніх посилань, анкорних посилань, mailto-посилань, форм, файлів cookie та варіантів використання.

Юзабіліті-тестування. Цей тип тестування перевіряє, чи відповідає структура інтерфейсу бізнес-цілям, чи справляє продукт правильне враження на цільову аудиторію і на які елементи додатку користувачі звертають увагу. Юзабіліті-тестування включає в себе виявлення проблем і помилок, пов'язаних з юзабіліті додатку, і пропозиції щодо поліпшення.

Тестування користувацького інтерфейсу. Під час тестування перевіряються основні елементи інтерфейсу меню, панелі інструментів, кнопки, зображення, діалогові вікна, поля введення, інструменти управління списками тощо. Тестування може проводитися вручну або за допомогою різних засобів автоматизації.

Тестування на сумісність. Цей тест виявляє сумісність програми з різними операційними системами, серверним програмним забезпеченням, пропускною здатністю мережі, браузерами, роздільною здатністю екрану, конфігураціями клієнтів, базами даних тощо. Основна мета цього типу тестування - забезпечити хороший користувацький досвід при створенні та взаємодії з універсальними додатками.

Тестування продуктивності. Завдання тесту - визначити максимальне навантаження, яке може витримати продукт. Також перевіряється стабільність роботи програми, оцінюються її можливості при різних конфігураціях середовища і можливості масштабування, визначаються межі при пікових навантаженнях і т.д.

Сам проект має два типи тестів юніт і інтеграційні тести. Самі тести показані на скріншоті нижче.

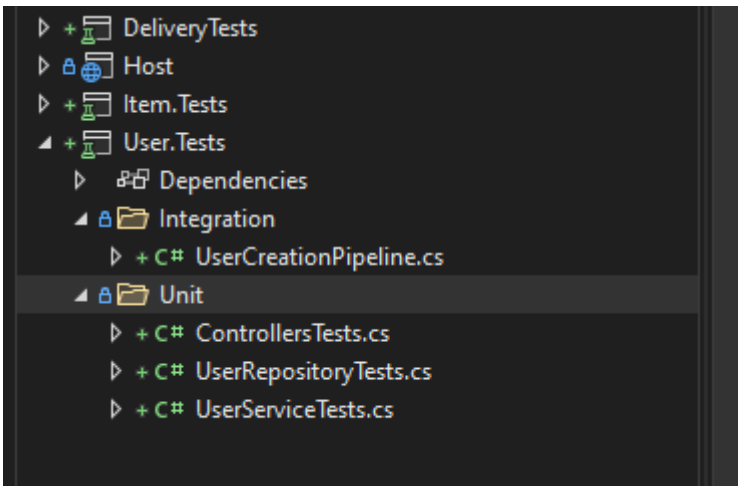

Рис 3.3.1 Наявні тести у проекті

Висновки

У цьому розділі проаналізовано методи та інструменти, використані для створення веб-додатку.

Для створення архітектури були використані патерни проектування синглтонів та спостерігачів. Патерни - це описи взаємодії між об'єктами та класами, адаптовані для вирішення конкретного завдання в певному контексті.

Патерни проектування дають назви структурам, абстрагують їх і визначають важливі аспекти структури, щоб їх можна було використовувати для створення багаторазових проектних рішень.

Для розробки веб-додатку були використані наступні технології

- C# - мова програмування.

- ASP.NET web api - програмна платформа для створення веб-додатків.

- Maria DB - реляційна база даних.

- React - фреймворк для побудови користувацьких інтерфейсів.

Цей набір технологій та архітектури дозволяє швидко та надійно розробляти архітектуру та реалізовувати веб-додатки. Мікросервісна архітектура робить додатки легко масштабованими, адаптованими до мінливих вимог та легко реорганізованими.

РОЗДІЛ 4. Реалізація системи управління логістичними центрами підприємства.

4.1 Розробка логістичної системи

Розробка складних інформаційних систем починається зі створення прототипу. Чим більша система, що розробляється, тим більше уваги слід приділяти її прототипу та дизайну.

Прототипи відрізняються за типом, рівнем візуалізації та інтерактивності. Від одного аркуша паперу до багатосторінкової структури, кожен прототип виконує одне завдання.

Навіть професіонали часто починають створення прототипу з ескізу на папері або офісній дошці. За допомогою грубого ескізу можна створити базове бачення перед початком роботи над кінцевим продуктом. Це можна зробити за короткий час і не вимагає спеціальних знань або інвестицій в робочі інструменти.

Прототипування - це перевірений метод проектування продуктів. Його суть полягає у створенні першої моделі кінцевого результату, яка називається прототипом.

Прототипування використовується на ранніх стадіях проектування для виявлення відсутніх, неправильно сформульованих вимог і необґрунтованих припущень, показуючи, як продукт буде виглядати і функціонувати.

Прототипи включають неробочі моделі, робочі представлення та цифрові представлення рішень або пропонованих продуктів. Прототипи можна використовувати як макети веб-сайтів, частково робочі проекти продуктів або для опису процесів за допомогою серії діаграм (наприклад, робочих процесів).

Прототипи бізнес-правил і даних можна використовувати для опису бажаного потоку процесів і бізнес-правил. Прототип даних можна використовувати для очищення і трансформації даних.

Підходи до створення прототипів. Існує два загальних підходи до створення прототипів:

Прототипи створюються за допомогою простих інструментів (наприклад, папір і олівець, дошка, програмне забезпечення) для визначення і уточнення вимог. Прототипи можуть бути оновлені та вдосконалені під час обговорення та розробки, але вони не стають робочим кодом і не зберігаються в результаті при впровадженні остаточної системи або процесу. Цей підхід корисний для визначення функцій або процесів, які не є легкодоступними, мають суперечливі точки зору або які важко зрозуміти за допомогою інших методів. Ці прототипи є недорогим засобом визначення або перевірки вимог, що виходять за межі інтерфейсу, наприклад, вимог, пов'язаних з процесами, даними або бізнес-правилами.

Еволюційні або функціональні: Прототипи створюються для перетворення початкових вимог у функціонуюче рішення в міру того, як вимоги розробляються за участю зацікавлених сторін. Цей підхід часто вимагає спеціалізованих інструментів або мов для створення прототипу, який може бути використаний в остаточному рішенні. Спеціалізоване програмне забезпечення можна використовувати для моделювання бізнес-процесів, правил і даних, щоб оцінити вплив змін і визначити бажані результати.

Сьогодні прототипування здійснюється багатьма способами.

Кожен з наведених нижче способів можна вважати різновидом прототипування

Перевірка принципу або перевірка концепції: модель, створена для перевірки дизайну системи без імітації зовнішнього вигляду кінцевого продукту, який буде використовуватися зацікавленими особами, матеріалів, використаних для виконання роботи, або процесу/робочого циклу Моделі, створені для перевірки дизайну системи без імітації зовнішнього вигляду кінцевого продукту, який буде використовуватися кінцевими зацікавленими особами, матеріалів, використаних для виконання роботи, і процесів/робочих циклів.

Прототип дослідження форми: Використовується для вивчення основних розмірів, зовнішнього вигляду та відчуття продукту, який буде вироблятися, без створення будь-яких реальних функцій. Використовується для оцінки ергономічних і візуальних елементів за допомогою скульптурних зображень продуктів, виготовлених з недорогих матеріалів. Цей тип прототипу також можна використовувати для моделювання високорівневих робочих процесів і навігації, а також для виявлення прогалин і невідповідностей у можливих рішеннях для функцій (наприклад, зовнішнього вигляду і конфігурації).

Юзабіліті-прототипи: моделі продукту, створені для перевірки того, як кінцеві користувачі взаємодіють з системою без включення функцій (зовнішній вигляд, конфігурація тощо).

Візуальні прототипи: моделі продукту, створені для тестування візуальних аспектів рішення без моделювання повної функціональності.

Функціональні прототипи: моделі, створені для тестування функціональності, користувацького досвіду (наприклад, зовнішнього вигляду) та робочого процесу програмного забезпечення. Ці моделі, які також називають робочими моделями, використовуються як для імітації бізнес-процесів і бізнес-правил, так і для оцінки функціональності програмного забезпечення.

Нижче наведено список найпоширеніших методів створення прототипів

1. **Розкадровка:** використовується для детального візуального та текстового опису низки дій, узагальнюючи різні взаємодії користувачів з рішенням або компанією.
2. **Паперове прототипування**: використовує папір та олівець для створення дизайну інтерфейсу та процесів.
3. **Моделювання робочого процесу**: описує послідовність операцій, які необхідно виконати, зазвичай фокусуючись лише на людському аспекті.
4. **Симуляція:** використовується для демонстрації рішень і компонентів рішень. Можна тестувати різні процеси, сценарії, бізнес-правила, дані та вхідні дані.

Прототипування забезпечує візуальне представлення майбутнього стану.

Зацікавлені сторони можуть внести свої пропозиції та надати зворотній зв'язок на ранніх стадіях процесу проектування.

При використанні методів швидкого або паперового прототипування користувачі можуть неохоче критикувати макети, оскільки вони не відшліфовані і не готові до виробництва.

Вузькі, але глибокі вертикальні прототипи можна використовувати для техніко-економічного обґрунтування, підтвердження концепції або для виявлення прогалин у технології чи процесах.

Прототипування також має певні обмеження. Якщо система або процес дуже складні, процес прототипування може перетворитися на обговорення "як", а не "що", що призведе до процесу, який потребує багато часу, зусиль і навичок фасилітації.

Щоб почати створення прототипу, може знадобитися розуміння або гіпотеза про технологію, що лежить в основі прототипу.

Якщо прототип глибоко розроблений і деталізований, зацікавлені сторони можуть мати нереалістичні очікування щодо кінцевого рішення. Ці очікування можуть варіюватися від передбачуваних дат завершення до надмірних очікувань щодо продуктивності, надійності та доступності.

4.2 Реалізація веб-додатку

Веб-додаток поділяється на дві частини бек-енд і фронт-енд. Точкою входу у ASP.NET додаток починаються з класу Program метод Main, який запускає IIS сервер на якому будуть працювати наші WEB API.

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllersWithViews();
builder.Services.AddControllers();
builder.Services.AddHttpClient();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller}/{action=Index}/{id?}");

app.MapFallbackToFile("index.html"); ;

app.Run();
```

Рис 4.1 Program.cs

Після старту програми до ендпоінтів Web API додаються ендпоінти які є в інших мікросервісах, після реєстрації і запуску IIS сервісу ми можемо робити реквести на них через браузерний рядок URL або за допомогою Swagger або Postman.

```
// This method gets called by the runtime. Use this method to add services to
the                                                                  container.
public        void        ConfigureServices(IServiceCollection        services)
{
    services.AddCors();
    services.AddControllersWithViews();

    services.RegisterUserDomain();
    services.RegisterUserPersistence(configuration);
```

```
    services.RegisterItemDomain();
    services.RegisterItemPersistence(configuration);

    services.RegisterDeliveryDomain();
    services.RegisterDeliveryPersistence(configuration);
}
```

Рис 4.2 Сервіси для Web API

Адреса де запускається IIS веб сервер - https://localhost:5001/. Запущений сервер реляційної бази даних MariaDB стартує на порту 3007. Конфігурація підключення до DB знаходиться у файлі Host - appsettings.json.

```
"ConnectionStrings":                                                    {
  "DBConnectionString":
"Server=127.0.0.1;Port=3307;Database=logistics;User=root;Password=qwerty;"
},
```

Рис 4.3 Рядок підключення до бд

Всі мікросервіси знаходяться в окремих папках Delivery, Item, User відповідно. Кожен мікросервіс містить рівень доступу до даних – DAL, рівень бізнес логіки – BLL, рівень комунікації - API

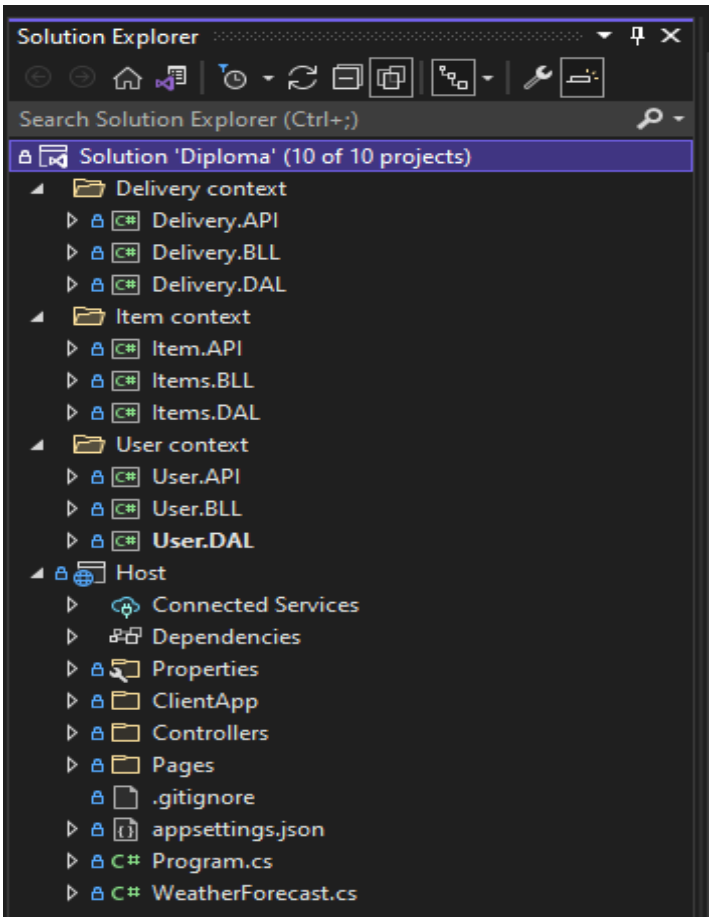

Рис 4.4 загальна схема проекта.

Інтерфейсна частина знаходиться в проекті – Host. В директорії client App знаходиться користувацький інтерфейс створений за допомогою React. Там знаходяться компоненти інтерфейса та ресурси.

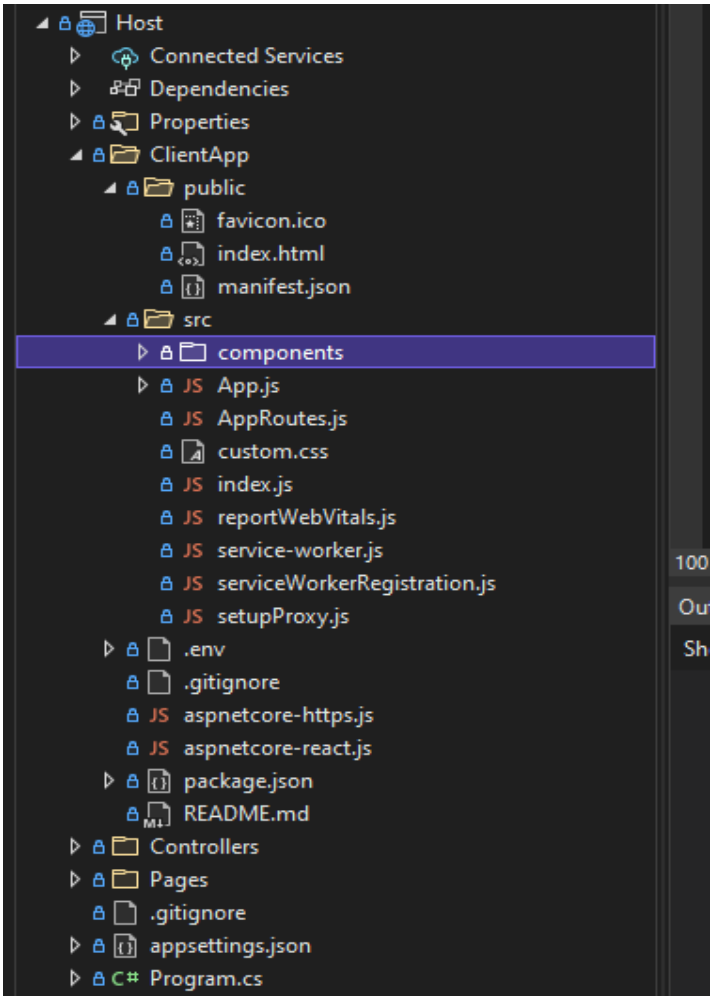

Рис 4.5 Загальна схема користувацького інтерфейса

Рівень доступу до бази даних створений за допомогою паттерна репозиторій. За допомогою інтерфейсу репозиторія можна одними й тими же викликами робити запити у різні репозиторії.

```
public class UserRepository : IUserRepository
    {
        public readonly UserDbContext _context;

        public UserRepository(UserDbContext context)
        {
            _context = context;
```

```
        }

        public async Task AddUser(User user)
        {
            _context.UserModels.Add(user);
            await _context.SaveChangesAsync();
        }

        public async Task DeleteUser(User user)
        {
            _context.UserModels.Remove(user);
            await _context.SaveChangesAsync();
        }

        public async Task<User> GetUserById(Guid userId)
        {
              return await _context.UserModels.FirstOrDefaultAsync(x => x.Id ==
userId);
        }

        public async Task UpdateUser(User user)
        {
            _context.UserModels.Update(user);
            await _context.SaveChangesAsync();
        }

        public async Task<IReadOnlyCollection<User>> GetAllUsers()
        {
            return await _context.UserModels.ToListAsync();
        }
    }
```

Рис 4.6 Репозиторій для мікросервіса з юзерами

Комунікаця з користувацьким інтерфейсом відбувається за допомогою контроллера web API. Цей контроллер приймає HTTP запити по URL та відповідає в залежності від запиту. Всі методи контроллера є асинхронними.

```
private readonly IUserRepository _userService;

        public UserController(IUserRepository repo)
        {
            _userService = repo;
        }

        /// <summary>
```

```
        /// Get all useres
        /// </summary>
        /// <returns>List of useres with dishes and useres information</returns>
        [HttpGet]
        public async Task<ActionResult<IReadOnlyCollection<User>>> GetAllAsync()
        {
            try
            {
                                IReadOnlyCollection<User>   userModels   =   await
_userService.GetAllUsers();
                return new OkObjectResult(userModels);
            }
            catch (Exception ex)
            {
                return BadRequest(new { message = ex.Message });
            }
        }

        /// <summary>
        /// Get user by user id
        /// </summary>
        /// <param name="id">Unique user id</param>
        /// <returns>User with informatio and dishes</returns>
        [HttpGet("{userId}")]
        public async Task<ActionResult<User>> GetAsync(Guid userId)
        {
            try
            {
                User res = await _userService.GetUserById(userId);
                return res;
            }
            catch (Exception ex)
            {
                return BadRequest(new { message = ex.Message });
            }
        }

        [HttpPost]
             public   async   Task<ActionResult<User>>   CreateUserAsync([FromBody]
UserRequest request)
        {
            try
            {
                var model = request.MapFrom();
                var res = await _userService.Create(model);
```

```
                return res.MapTo();
            }
            catch (Exception ex)
            {
                return BadRequest(new { message = ex.Message });
            }
        }

        [HttpPut("{userId}")]
            public  async  Task<ActionResult<UserResponse>>  UpdateUserAsync(Guid
userId, [FromBody] UserRequest request)
        {
            try
            {
                User model = request.MapFrom();
                User res = await _userService.Update(userId, model);
                return res.MapTo();
            }
            catch (Exception ex)
            {
                return BadRequest(new { message = ex.Message });
            }


            [HttpDelete("{userId}")]
            public async Task<ActionResult> DeleteAsync(Guid userId)
            {
                try
                {
                    await _userService.DeleteAsync(userId);
                    return Ok();
                }
                catch (UserDomainException ex)
                {
                    return BadRequest(new { message = ex.Message });
                }
            }
```

Рис 4.5 Загальний вигляд контроллера користувачів

Для створення проекцій одних сутностей на інші, а саме на DTO моделі були розроблені спеціалізовані мапери. Вони використовуються контролерами WEB API для проектування сутностей на рівні бази даних на сутності на рівні користувача, якими вже обмінювалися між контролером і користувацьким інтерфейсом.

```
public static class UserMapper
{
    public static User MapFrom(this UserRequest request)
    {
                return   new   User(request.UserstartDate,   request.UserEndDate,
request.Address.AddressId, request.Item);
    }

    public static UserResponse MapTo(this User model)
    {
        UserResponse response = new UserResponse()
        {
            GuidId = model.GuidId,
            UserstartDate = model.UserStartDate,
            UserEndDate = model.UserEndDate,
            Item = model.Item
        };

        return response;
    }

              public      static      GetAllUsersResponse      MapToCollection(this
IReadOnlyCollection<User> UserModels)
    {
        return new GetAllUsersResponse(UserModels.Select(cm => cm.MapTo())
            .ToList());
    }
}
```

Рис 4.8 Маппер користувачів

Авторизація користувача відбувається за допомогою JWT токенів. Коли відбуваються HTTP запити на контроллери на користувацькому інтерфейсі в загаловки HTTP запитів додається веб токен. Для створення токенів використовується бібліотека Microsoft.IdentityModel. Час життя токена 15 хвилин.

```
{
    public class (IConfiguration config)
    {
         _config = config;
    }

    private string GenerateJWTToken(User user)
        {
```

```
                                        var      securityKey      =      new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Jwt:Key"]));
                    var   credentials   =   new   SigningCredentials(securityKey,
SecurityAlgorithms.HmacSha256);
            var claims = new[]
            {
                new Claim(ClaimTypes.NameIdentifier,user.Username),
                new Claim(ClaimTypes.Role,user.Role)
            };
            var token = new JwtSecurityToken(_config["Jwt:Issuer"],
                _config["Jwt:Audience"],
                claims,
                expires: DateTime.Now.AddMinutes(15),
                signingCredentials: credentials);


            return new JwtSecurityTokenHandler().WriteToken(token);

        }
}
```

Рис 4.9 JWTManager для створення токенів

4.3Скріншоти роботи веб-додатку

Коли користувач заходить у веб додаток перше, що він побачить це буде сторінка авторизації. Користувач повенен ввести свій логін та пароль.

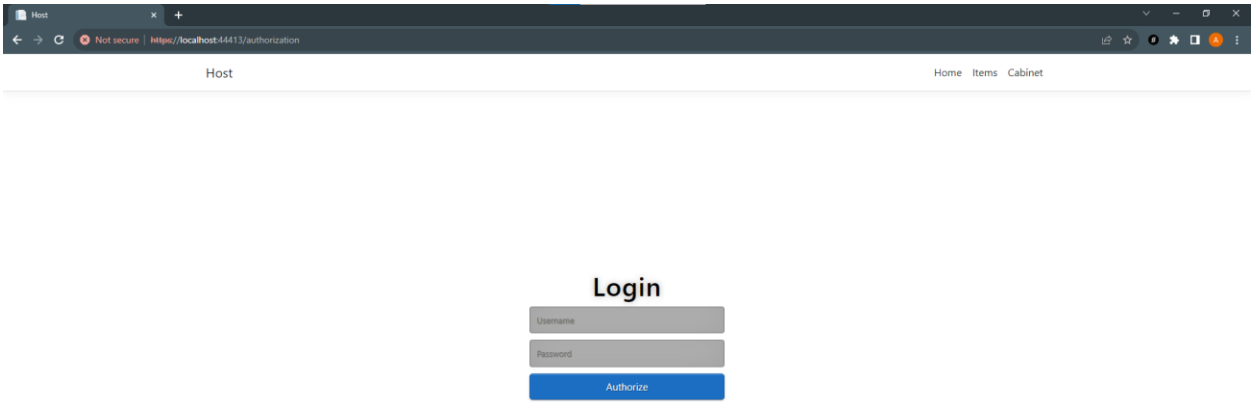

Рис 4.10 Форма авторизації

Користувач увійшовши до системи, має можливість зайти у свій кабінет та подивитися на свої дані, свої доступи та роль доступу.

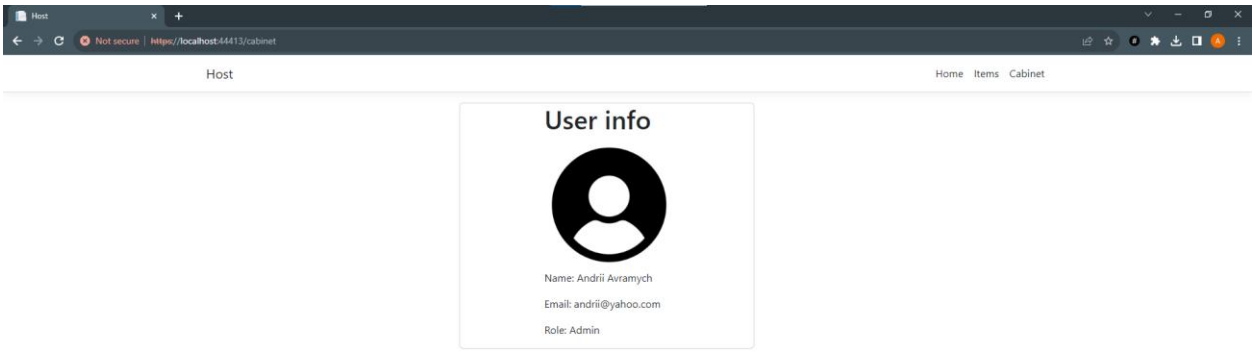

Рис 4.11 Інформація у кабінеті користувача.

Загальне меню товарів, які зараз знаходяться на складі, користувач з відповідним рівнем доступа, може замовити їх доставку на інший склад або видалити їх зі складу.

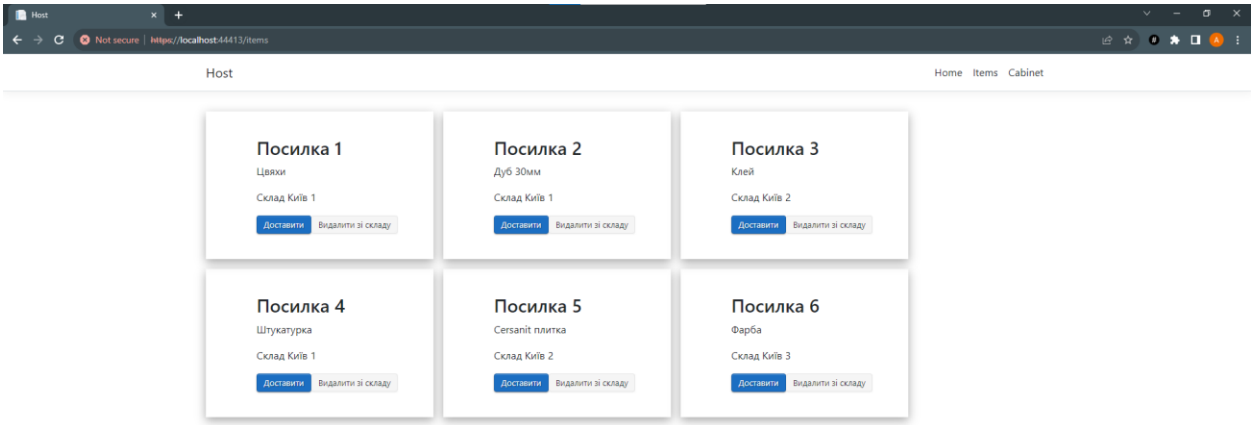

Рис 4.11 Загальна кількість товарів на складі

Коли користувач вибирає доставку на певний склад йому стає доступна опція вибору певного складу.

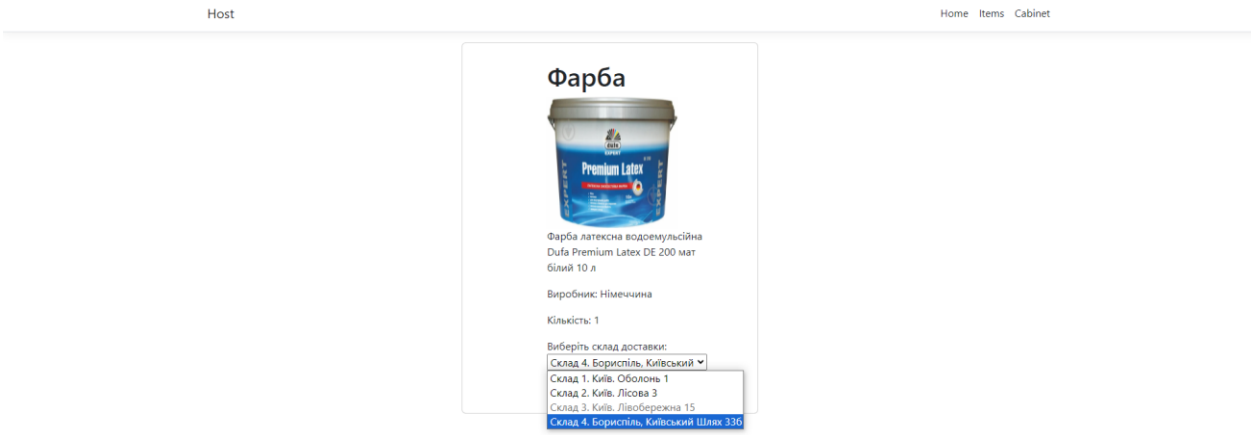

Рис 4.12 Вибір доставки фарби зі складу

ВИСНОВКИ

Метою даної дипломної роботи було створення система управління логістичним центром підприємства. Цей продукт повинен спростити процеси контролю та роботи логістичних операцій підприємства.

Для створення веб додатку система управління логістичним центром підприємства були проаналізовані існуючі системи конкурентів, та надана всебічна оцінка галузі логістичних систем.

В процесі розробки я використовував технології вивчені під час навчання в університеті: .NET/C#, MSSQL, ASP.NET, React. Комунікація бекенд частини з користувацьким інтерфейсом здійснюється за допомогою WEB API. Комунікація працює за допомогою методології REST API

В майбутньому планується розробити десктопні та мобільні версії клієнтів цієї системи за допомогою технології MAUI. Це дозволить більш зручно використовувати цю систему на більшій кількості пристроїв, що збільшить задоволеність клієнтів.

Отже, основна мета даної роботи була виконана – була розроблена система управління логістичним центром підприємства

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sam Newman. Building Microservices: Designing fine-grained systems: O`Rilley, 2021 – 258 с.

2. Martin Kleppmann. Designing Data-Intensive Applications: The big ideas behind reliable, scalable and maintainable systems: O`Rilley, 2019 – 590 с.

3. Jeffrey Richter. CLR via C#, 4th edition: Microsoft press, 2017 – 876 с.

4. Mark Richards, Neal Ford. Fundamentals of Software Architecture: O`Rilley, 2016 – 234 с.

5. Alan Beaulieu. Learning SQL, 3rd edition: O`Rilley, 2015 – 346 с.

6. Vlad Khononov. Learning Domain-Driven Design: O`Rilley media, - 469 с.

7. Патерн Observer (спостерігач). 2016. [Електронний ресурс]. – Режим доступу : https://refactoring.guru/design-patterns/observer

8. Патерн Singleton (Одиночка). 2016. [Електронний ресурс]. – Режим доступу : https://refactoring.guru/design-patterns/singleton

9. Ларман, Крэг. Застосування UML 2.0 та шаблонів проектування / Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. — [3-е вид.] 2006. — 736с.

10. Введення в ASP.NET 5. 2021. [Електронний ресурс]. – Режим доступу : https://habr.com/ru/post/243667/

11. Оптимізація ASP.NET – практичні поради по роботі з IIS. 2015. [Електронний ресурс]. – Режим доступу : https://habr.com/ru/post/250881/

12. Eric Freeman, Elisabeth Robson. Head First Design Patterns: O'Reilly Media, Inc. – 2021, 454 с.

13. Sean P. Kane, Karl Matthias. Docker: Up & Running, 3rd Edition: O'Reilly Media, Inc. – 2020, 394 с.