**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ**
**ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**
**КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

# КВАЛІФІКАЦІЙНА РОБОТА

першого (бакалаврського) рівня вищої освіти

на тему: **«Розробка інформаційної системи управління даними з обробки та доставки замовлень»**

Виконав: студент 4 курсу групи Кн-41сп

Спеціальності 122 «Комп'ютерні науки»
(шифр і назва)

Хомищак Євген Богданович
(Прізвище та ініціали)

Керівник: к.т.н., доцент Желєзняк А.М.
(Прізвище та ініціали)

**ДУБЛЯНИ-2024**

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА
ІНФОРМАЦІЙНИХ ТЕХНЕОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Перший (бакалаврський) рівень вищої освіти
Спеціальність 122 «Комп'ютерні науки»

«ЗАТВЕРДЖУЮ»

Завідувач кафедри_____

д.т.н., проф. А. М. Тригуба
«____» _____2023 р.

# ЗАВДАННЯ

на кваліфікаційну роботу студенту

Хомищаку Євгену Богдановичу

1. Тема роботи: «Розробка інформаційної системи управління даними з обробки та доставки замовлень»

Керівник роботи Желєзняк Алла Михайлівна, доцент
затверджені наказом по університету від 27.11.2023 року № 641/к-с.

2. Строк подання студентом роботи 10.06.2024 р.
3. Вихідні дані до роботи: *вимоги до проектування інформаційних систем; методика проектування інформаційних систем; технічне завдання на проектування інформаційної системи управління даними з обробки та доставки замовлень.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)
*Вступ.*
*1. Аналіз предметної області.*
*2. Постановка задачі та планування проекту*
*3. Проектування інформаційної системи управління даними з обробки та доставки замовлень*
*4. Охорона праці та безпека в надзвичайних ситуаціях*
*Висновки та пропозиції.*
*Список використаної літератури.*
5. Перелік ілюстраційного матеріалу (з точним зазначенням обов'язкових креслень): *Огляд ринку служб доставки в Україні, аналіз особливостей використання API служб доставки замовлень, вибір засобів реалізації проекту та середовища розробки, формування бази даних для збереження та управління даними, особливості*

*розробки інформаційної системи управління даними з обробки та доставки замовлень, сценарії використання розробленої інформаційної системи*

6. Консультанти з розділів:

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| 1, 2, 3 | *Желєзняк А.М., доцент кафедри ІТ* | | |
| 4 | *Городецький І.М., доцент кафедри фізики, інженерної механіки та безпеки виробництва* | | |

7. Дата видачі завдання 30 листопада 2023 р.

Календарний план

| № з/п | Назва етапів дипломного проекту | Терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | *Написання першого розділу* | *30.11.23-02.02.24* | |
| 2 | *Виконання другого розділу та аркушів ілюстраційного матеріалу до нього* | *03-28.02.24* | |
| 3. | *Виконання третього розділу та аркушів ілюстраційного матеріалу до нього* | *01.03-30.04.24* | |
| 4. | *Написання розділу «Охорона праці»* | *01-15.05.24* | |
| 5. | *Завершення оформлення розрахунково-пояснювальної записки та аркушів ілюстраційного матеріалу* | *15-30.05.24* | |
| 6. | *Завершення роботи вцілому* | *01 - 10.06.24* | |

Студент _____ Хомищак Є.Б.
(підпис)

Керівник роботи _____Желєзняк А.М.
(підпис)

УДК 004.9 : 631.1

Розробка інформаційної системи управління даними з обробки та доставки замовлень. Хомищак Є.Б. Кафедра ІТ – Дубляни, Львівський НУП, 2024. Кваліфікаційна робота: 59 с. текст. част., 24 рис., 10 табл., 10 арк. ілюстраційного матеріалу, 23 джерел.

Проведено загальний огляд ринку служб доставки в Україні, встановлено основні переваги і недоліки найпопулярніших служб доставки. Проведено детальний аналіз особливостей використання API служб доставки замовлень.

Обґрунтовано вибір засобів реалізації проекту та середовища розробки. На основі проведеного аналізу було обрано відповідні технології та інструменти для інформаційної системи управління даними.

Здійснено формування бази даних для збереження та управління даними користувачів та розроблено інформаційну систему управління даними з обробки та доставки замовлень. Описано різні сценарії використання розробленої інформаційної системи управління даними з обробки замовлень.

Розроблено заходи щодо охорони праці та безпеки в надзвичайних ситуаціях.

# ЗМІСТ

# ВСТУП

Зараз майже неможливо уявити наш світ вільним від впливу інформаційних технологій, оскільки вони проникли в усі закутки людського існування та діяльності. І саме завдяки цьому ми можемо покращити все своє життя. Візьмемо, наприклад, системи навчання, які роблять онлайн-освіту більш керованою, спрощуючи процес: у вашому власному темпі та часі, або цифрові сервіси, які допомагають полегшити та зробити щоденні завдання зручнішими, як-от перегляд маршрутів наземного транспорту, перш ніж вибрати вид транспорту чи виклик таксі та можливість перегляду фільмів у дорозі — це лише кілька прикладів. Життя без інформаційних технологій неможливо уявити!

Сьогодні використання додатків доставки різних товарів і послуг стало ще зручнішим. Оскільки ринок цифрових послуг у кожного під рукою, можна легко замовити те, що вам потрібно, безпечно та зручно онлайн, не виходячи з дому. Як наслідок, ми побачили приплив у розширення ринку програмного забезпечення, спрямованого на досягнення цих цілей. Ситуація особливо актуальна: заявки на товари та послуги сьогодні стають все більш важливими.

Розширення онлайн-торгівлі та збільшення кількості угод підіймають потребу у швидкій та ефективній обробці даних на новий рівень. У цій грі постачальники перебувають у центрі уваги як головні актори, які повинні добре зіграти свою роль у забезпеченні швидкої та безперебійної доставки.

В епоху глобалізації та передових технологій люди бажають мати продукти на порозі з різних причин. Служби доставки дозволяють людям зручно замовляти товари та послуги в бажаному місці — додому чи в офіс. Це виявляється дуже корисним для тих, хто має обмеження в часі або обмеження фізичного доступу до магазинів або закладів харчування самостійно.

Зручний варіант доставки допомагає людям економити час, який зазвичай витрачається на дорогу до магазину чи ресторану, пошук товару та очікування в черзі. Це дозволяє людям замовляти все, що їм потрібно, у будь-який час і доставляти це прямо до порогу.

Служби доставки надають споживачам широкий вибір. Вони можуть замовляти їжу в будь-якому ресторані, товари в будь-якому інтернет-магазині, ліки в будь-якій аптеці, дозволяючи їм вибирати на основі ціни, якості та особистих уподобань.

У сфері безпеки та охорони здоров'я, особливо під час епохи COVID-19, служби доставки вийшли на новий рівень популярності, оскільки вони тримаються подалі від фізичного контакту, який може призвести до небажаних інфекцій. Ідея безконтактної доставки, коли товар просто залишають на порозі без будь-якої прямої взаємодії з кур'єром, зараз є перевагою для багатьох людей.

Сьогодні як ніколи актуальні всі онлайн-замовлення. Але основна проблема полягає в тому, що для того, щоб замовити товар в магазині, на складі або у фізичної особи, потрібно або знати їх номер і зробити замовлення по телефону, або шукати сайт або додаток конкретного магазину, якщо він є. . Така ситуація незручна ні для споживачів, ні для постачальників: у обох немає повної інформації один про одного. Це також ускладнює комунікацію між ними через відсутність універсального способу замовлення товарів, який би в іншому випадку спростив процес і зробив його більш ефективним.

Це призвело до попиту на системи обробки даних у службах доставки, які повинні мати прості та ефективні інструменти для реєстрації та відстеження замовлень, а також легко переглядати умови та спілкуватися з постачальниками.

# 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Аналіз ринку служб доставки в Україні

Зараз на українському ринку домінують дві основні служби доставки: Нова Пошта та Укрпошта. Найчастіше кожен користувач вибирає одну з них, зазвичай віддає перевагу Новій Пошті [1].

Існують також спеціальні методи доставки, такі як Delivery та Meest. У той час як Meest зосереджується на доставці товарів за кордон, Delivery все ще залишається способом доставки великих внутрішніх товарів.

«Нова Пошта» стала одним із найпопулярніших способів доставки по всій Україні та дозволяє користувачам отримувати найзручніші варіанти доставки. «Нова Пошта» — приватна компанія, яка самостійно регулює методи своєї діяльності. У результаті клієнти отримують спеціальні пропозиції та покращені послуги, хоча й за вищою ціною [1].

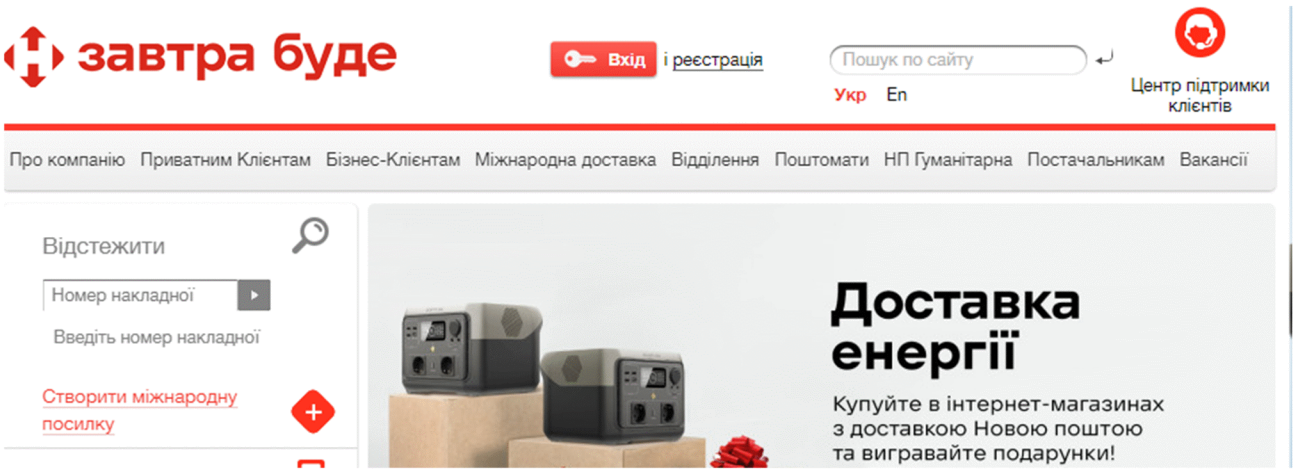



Рис. 1.1 Веб-сайт служби доставки Нова Пошта

До переваг цієї служби доставки можна віднести [ 2]:

**Швидка доставка**. У більшості випадків час від відправлення посилки до отримання не перевищує двох робочих днів.

**Розгалужена мережа відділень**. Клієнт може вибрати будь-яке зручне для нього відділення, яке розташоване в найбільш густонаселених областях України. Широкий вибір варіантів не тільки означає, що ви можете вибирати з великої кількості місць збору, але також дозволяє використовувати звичайні торгові точки, вантажні перевезення, поштові автомати або експрес-доставку.

**Зручні послуги**. Клієнти отримують миттєве сповіщення про те, що їхній пакунок уже в дорозі, а також ще одне сповіщення, коли воно прибуває. Крім того, ви можете вивести на новий рівень ефективність самого додатку «Нова Пошта». Таким чином, клієнти мають можливість сплатити вартість доставки заздалегідь, дізнатися точну дату прибуття, відстежити її за допомогою карт і штрих-кодів і забрати кілька відправлень від різних продавців одночасно.

**Доступність та інтеграція**. «Нова Пошта» — дуже популярний спосіб доставки, і коли потенційний клієнт заходить в магазин з обраним товаром, він не запитує «Чи доставляють Новою Поштою?». Саму Нову Пошту також можна знайти на більшості сайтів. Крім того, при створенні сайту на Weblium ви також можете додати цей спосіб доставки, що значно спростить вашу взаємодію з клієнтами.

Однак, варто зазначити, що вартість доставки Новою Поштою вища, ніж іншими поштовими службами в Україні. Так, базова ціна доставки по містах становить 70 гривень, а по селищах сільського типу та селища міського типу – 95 гривень. Ціна також залежить від обсягу та ваги упаковки.

Укрпошта не менш популярна, ніж Нова Пошта і для багатьох є послугою номер один. Однак нюанси роботи цієї служби доставки свідчать про те, що вона поступається своїм «приватним» конкурентам [2].

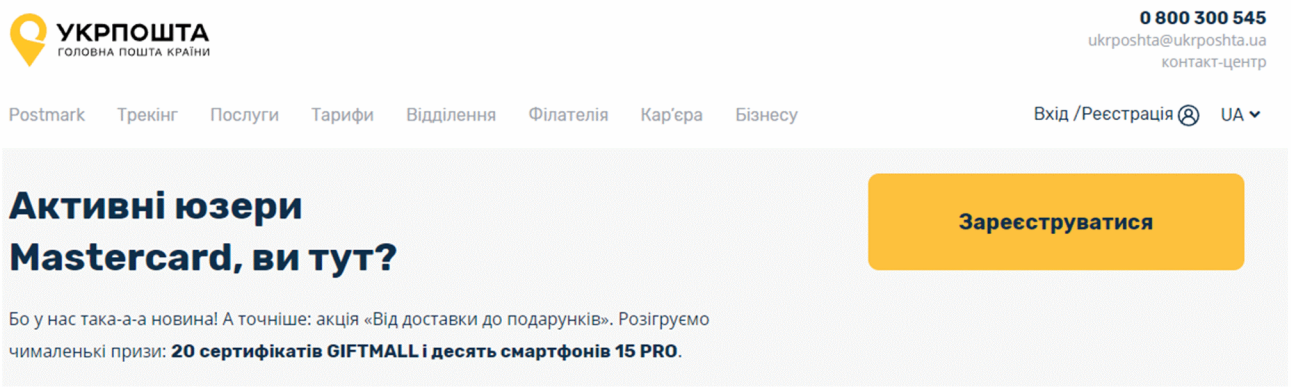



Рис. 1.2 Веб-сайт служби доставки Укрпошта

Варто зазначити, що Укрпошта є національною організацією. Це дозволяє тримати ціни на максимально низькому рівні. Так, базова вартість доставки становить 25 грн, що є ключовою перевагою вибору цієї служби доставки. Ви також можете розрахувати точну вартість доставки посилки.

Проте «Укрпошта» зазнала більше критики та значно більше недоліків у своїй роботі. Серед них можна виділити [2]:

**Тривалий термін доставки.** Відправка Укрпоштою займає 3 робочих дні з моменту відправлення посилки до отримання. Нова Пошта відправляє посилки щодня, а відділення Укрпошти мають фіксований графік доставки. Це сильно впливає на швидкість доставки.

**Погане обслуговування**. Звичайно, все залежить від окремих секторів і операторів. Однак користувачі повідомляють, що посилки Укрпошти часто губляться в дорозі, а компенсації не передбачено. Також часті затримки. Якщо мова йде про доставку швидкопсувних вантажів, то Укрпошта буде не найоперативнішим сервісом.

**Працювати онлайн незручно**. Робота в Інтернеті не є інтуїтивно зрозумілою, сама програма часто дає збої, а інформація про пакет може відображатися некоректно.

Укрпошта залишається практичним і актуальним способом доставки для тих, хто хоче заощадити. Також можна розглянути логістику цієї компанії, оскільки для деяких клієнтів туди зручніше і швидше поїхати, ніж у відділення Нової Пошти.

Служба доставки Meest — міжнародна компанія, яка в основному зосереджена на доставці в країни Європи, США та Канаду. Але можлива доставка по Україні і через Міст [2].

Базова вартість упаковок вагою до 1 кг – 50 грн. Клієнти можуть обрати як доставку невеликих посилок (документи, посилки вагою до 2 кг), так і великих посилок. Основна спеціалізація компанії – велика упаковка.

Незалежно від того, чи планується продавати товари за кордон, чидоставляти продукцію з Європи та США в Україну, цей спосіб доставки стане чудовим рішенням.

Служба доставки, яка створена Delivery Group спеціалізується на доставці великогабаритних вантажів по Україні. Цей метод розповсюдження більше спрямований на розповсюдження будівельних матеріалів і запасних частин, таких як автомобілі та техніка.

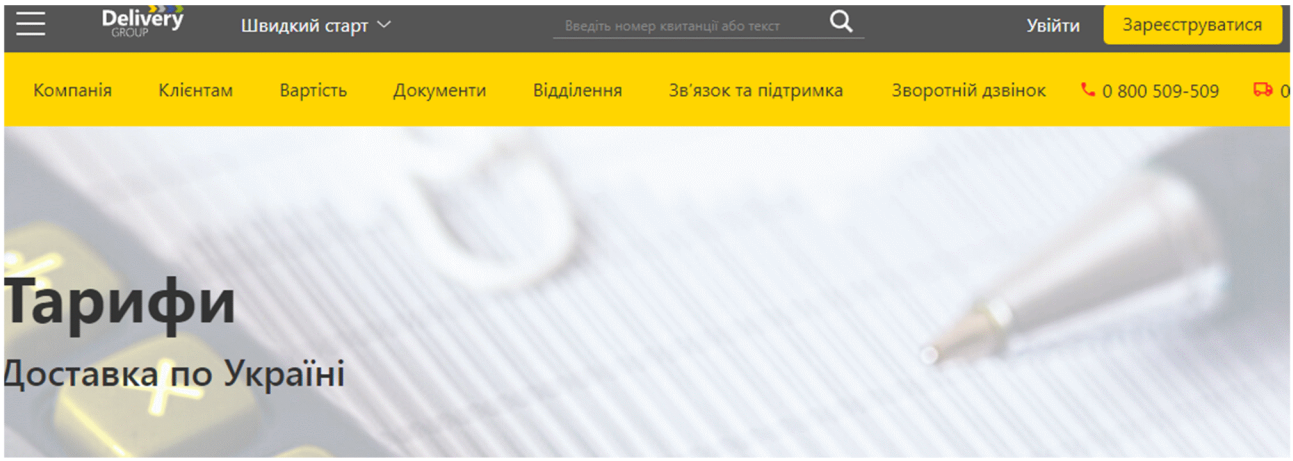



Рис. 1.3 Веб-сайт служби доставки Delivery

Сервіс має багато переваг, серед яких[1,2]:

**Поліпшена логістика.** Можна обрати найкращий час доставки. Так, на сайті доступні 24 години, 48 годин і 72 години. Це дозволяє заощадити гроші, коли доставка не термінова.

**Страхування вантажу.** Усі товари застраховані, і клієнти мають можливість отримати компенсацію від служби доставки, якщо вони будуть пошкоджені під час транспортування.

**Широка мережа складів.** Delivery має досить розгалужену складську мережу, що включає понад 460 представництв, регіональних відділень і 27 сортувальних центрів. Це дозволяє вибрати найбільш зручний спосіб доставки та покращити логістику для ваших клієнтів.

**Доставка до відділення.** Існують спеціальні галузеві тарифи, які дозволяють спеціалізуватися на певних видах доставки. Так, на сайті ви можете скористатися тарифами «Агропрайс», «Будівельна ціна», «Велопрайс» та ін. Детальніше про цінову політику компанії ви можете прочитати на сайті доставки.

Вузька спеціалізація компанії дозволяє краще зосередитися на покращенні сервісу та забезпеченні конкурентної переваги. Тому Delivery спеціалізується на доставці великих відправлень.

## 1.2 Особливості використання API служб доставки замовлень

Щоб організувати відвантаження з інтернет-магазину разом із замовленням необхідно підписати договір з сервісом. Нова пошта пропонує можливість автоматизувати логістичні процеси: відправлення можна обробляти та контролювати за допомогою особистого бізнес-облікового запису на веб-сайті сервісу або через API.

Ще одне корисне рішення, яке «Нова Пошта» пропонує для інтернет-магазинів, — це віджет, який можна просто «вбудувати» на сайт, що дає користувачам можливість самостійно відстежувати товари, розраховувати вартість та отримувати інформацію про умови доставки [1].

API Нової Пошти дозволяє виконувати різні дії, такі як створення відправлень, відстеження посилок, розрахунок вартості доставки тощо. Запити виконуються за допомогою HTTP POST методів, де кожен запит повинен містити

певні обов'язкові поля та слідувати певному формату. Запити до API виконуються на URL: https://api.novaposhta.ua/v2.0/json/. Кожен запит містить JSON-об'єкт із необхідними параметрами.

Основні параметри запиту для створення відпралення [1]:

- apiKey: API-ключ для доступу до API Нової Пошти.
- modelName: Назва моделі, з якою ви працюєте. Для створення відправлення це InternetDocument.
- calledMethod: Метод, який викликається. Для створення відправлення це save.
- methodProperties: Об'єкт, який містить властивості для методу. Для створення відправлення це деталі про відправлення, такі як тип платника, метод оплати, дата та час відправлення, тип вантажу, вага, опис, адреси відправника та одержувача, контактні дані тощо.

Основні параметри запиту для відстеження відправлення:

- modelName: Назва моделі для відстеження документів - TrackingDocument.
- calledMethod: Метод для отримання статусу документів – getStatusDocuments
- methodProperties: Об'єкт, який містить властивості для методу. У цьому випадку це масив документів із номерами та телефонами.

Усі відповіді API також повертаються у форматі JSON і містять поле success для індикації успішності запиту та масив data, який містить результати запиту.

Щоб оформити доставку з інтернет-магазину за допомогою Укрпошти, корпоративним клієнтам необхідно зателефонувати в контакт-центр, уточнити інформацію про тарифи, знижки та умови співпраці, укласти договір з Укрпоштою, узгодити місце відправлення пошти та заповнити відповідні форми та декларації або в Особистому кабінеті або через API. Також варто відзначити, що користувачам API надається знижка 10% на відправлення (за умови укладення договору).

API Укрпошти надає широкий спектр функцій для інтеграції поштових послуг у бізнес-процеси. Всі запити до API виконуються за допомогою HTTP POST методів і передаються у форматі JSON. Запити до API виконуються на URL: https://www.ukrposhta.ua/ecom/0.0.1/. Кожен запит має містити JSON-об'єкт з необхідними параметрами.

Основні параметри запиту для створення відпралення [1]:

- apiKey: API-ключ для доступу до API Укрпошти.
- sender: Інформація про відправника, включаючи ім'я, адресу, телефон та email.
- recipient: Інформація про одержувача, включаючи ім'я, адресу, телефон та email.
- parcel: Інформація про посилку, включаючи вагу, розміри (довжина, ширина, висота), вартість та опис.
- serviceType: Тип послуги (наприклад, "STANDARD").
- paymentMethod: Метод оплати (наприклад, "CASH").

Відповіді API повертаються у форматі JSON і містять поля для індикації успішності запиту та дані, що запитуються. Наприклад, відповідь на запит відстеження може містити інформацію про поточний статус відправлення, його місцезнаходження та історію переміщень.

API Meest надає широкий спектр функцій, які спрощують інтеграцію логістичних послуг з бізнес-процесами. Однією з ключових особливостей є можливість створення та управління відправленнями, що включає генерацію унікальних трекінг-номерів, створення лейблів та розрахунок вартості доставки. API дозволяє інтегрувати ці функції безпосередньо в інтернет-магазини та інші платформи, автоматизуючи процеси оформлення та обробки замовлень. Крім цього, API забезпечує відстеження відправлень у режимі реального часу, надаючи детальну інформацію про поточний статус і місцезнаходження посилки, що підвищує рівень прозорості та клієнтського сервісу.

API Meest дозволяє інтегрувати функції логістичних послуг з різними бізнес-процесами. Запити до API виконуються через HTTP POST та GET методи і передаються у форматі JSON. Запити до API виконуються на базовий URL: https://api.meest.com/v2/json/. Кожен запит містить необхідні параметри в JSON-форматі.

Основні параметри запиту для створення відпралення:

- apiKey: API-ключ для доступу до API Meest.
- modelName: Назва моделі, з якою ви працюєте. Для створення відправлення це InternetDocument.
- calledMethod: Метод, який викликається. Для створення відправлення це save.
- methodProperties: Об'єкт, який містить властивості для методу. Для створення відправлення це деталі про відправника, одержувача, деталі посилки (тип вантажу, вага, опис, вартість), адреси відправника та одержувача, контактні дані тощо.

Основні параметри запиту для відстеження відпралення:

- modelName: Назва моделі для відстеження документів - TrackingDocument.
- calledMethod: Метод для отримання статусу документів - getStatusDocuments.
- methodProperties: Об'єкт, який містить властивості для методу. У цьому випадку це масив документів із номерами для відстеження.

Всі відповіді API повертаються у форматі JSON і містять поле success для індикації успішності запиту та масив data, який містить результати запиту, наприклад, статус відправлення, його місцезнаходження та іншу релевантну інформацію.

Ще один відомий сервіс **ShipStation** — це веб-рішення для керування замовленнями та доставкою. За допомогою ShipStation API можна отримувати тарифи доставки від різних перевізників і отримувати мітки для своїх посилок в єдиному інтерфейсі на основі заданих параметрів.

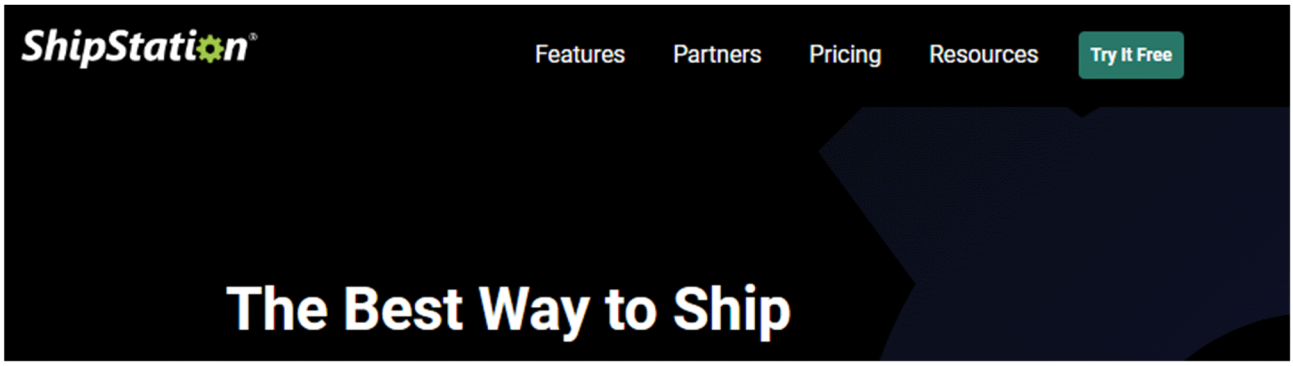



Рисунок. 1.4 Веб-сайт служби сервісу ShipStation.

Насправді він діє як посередник між такими перевізниками, як FedEx, USPS, DHL, APC та іншими підприємствами, які потребують подібних логістичних послуг. Цей спосіб дозволяє обробити десятки або сотні посилок і негайно передати їх перевізнику.

Як і Payoneer, ShipStation використовує JSON API і популярний метод підписання запитів за допомогою заголовка «Authorization» зі значенням «Basic». У цьому випадку ідентифікатор ($id) і ключ ($key) надаються розробнику ShipStation.

Сервіс працює в двох вимірах розміру (см і дюйми) і в трьох вимірах ваги (грами, унції і фунти). Ви можете надіслати параметри посилки та отримати вартість доставки для різних способів доставки з усіх доступних служб і вибрати найбільш підходящий. Крім того, враховуються різні варіанти упаковки, що робить вибір послуг і способів доставки більш гнучкими.

Усі сутності API, такі як користувачі, замовлення, пакети тощо, розділені, що дозволяє зручно робити запити, що неможливо з усіма API. Кожна служба доставки має свої нюанси, і в певному сенсі ці нюанси також відображені в ShipStation API. Це можна побачити, наприклад, при роботі з типами пакетів при виборі FedEx, але ці особливості стають очевидними лише на етапі підключення до самого API. Тому деякі служби доставки можуть вимагати додаткового налаштування для обробки даних. Але з іншого боку, ShipStation є хорошою універсальною службою, і її підключення, як правило, економить ваш час

розробки, усуваючи необхідність підключатися та інтегрувати API з різних служб.

Зручна система повернення помилок, яка враховує статус коду HTTP, може допомогти розробникам правильно обробляти несподівані помилки. Для одного облікового запису API ShipStation API обмежено 40 запитами на хвилину. Для більш частого використання потрібна окрема угода з ShipStation, що може стати перешкодою для великих проектів із використанням цієї послуги.

# 2. ПОСТАНОВКА ЗАДАЧІ ТА ПЛАНУВАННЯ ПРОЕКТУ

## 2.1. Мета та задачі проекту

Основна мета створення інформаційної системи управління даними з обробки та доставки замовлень полягає в автоматизації та оптимізації процесів, пов'язаних з прийомом, обробкою та доставкою замовлень. Така система покликана забезпечити ефективне управління замовленнями, скорочення часу обробки замовлень, зменшення людського фактора в обробці даних та підвищення точності виконання замовлень. Впровадження цієї системи сприятиме покращенню обслуговування клієнтів, збільшенню задоволеності користувачів та, відповідно, підвищенню конкурентоспроможності підприємства.

Однією з ключових задач системи є забезпечення централізованого зберігання та управління інформацією про замовлення та клієнтів. Це включає в себе збереження всіх необхідних даних про клієнтів, історію їхніх замовлень, стан поточних замовлень та інформацію про доставку. Завдяки централізованій базі даних та інтеграції з іншими системами підприємства, інформаційна система дозволить оперативно отримувати актуальну інформацію, що сприятиме прийняттю обґрунтованих управлінських рішень.

Крім того, інформаційна система має бути оснащена функціональністю для відстеження стану замовлень у реальному часі, що дозволить клієнтам отримувати актуальну інформацію про етапи обробки та доставки їхніх замовлень. Це не тільки покращує прозорість процесів для клієнтів, але й дозволяє підприємству швидко реагувати на будь-які проблеми, що виникають під час обробки або доставки замовлень. В результаті, система забезпечує високий рівень контролю за виконанням замовлень та підвищує загальну ефективність бізнес-процесів.

Основні задачі для розробки інформаційної системи управління даними з обробки та доставки замовлень включають:

- Забезпечення автоматизованої обробки замовлень від прийому до відправлення, включаючи збирання, підготовку до доставки та відстеження статусу замовлення;
- Централізоване зберігання та управління даними про клієнтів, їх замовленнями, статусами доставок та іншою важливою інформацією.
- Можливість інтеграції інформаційної системи з іншими бізнес-системами, такими як системи обліку, фінансові системи, складський облік тощо, для забезпечення єдиної точки доступу до даних та покращення управлінської звітності.

## 2.2. Вибір засобів реалізації проекту

**Мова програмування.** C# (вимовляється як "сі Шарп") — це об'єктно-орієнтована мова програмування загального призначення від Microsoft. Він був випущений в 2001 році. Він працює на платформі .NET. Саме після випуску платформи .NET Core ця мова започаткувала нові області та сфери застосування. Вивчення C# є хорошим вибором для початківців, оскільки процес програмування відносно простий і зрозумілий. Освоївши основи мови, ви можете пов'язати свою професійну діяльність з веб-розробкою.

Основні напрямки використання C#:

- Розробка веб-сайтів на платформі .NET або програмному забезпеченні з відкритим кодом;
- Розробляти програми та додатки для Windows;
- Реалізація мобільних і настільних ігор;
- Створення хмарних сервісів.

Більшість програм і веб-сервісів, пов’язаних із продуктами Microsoft, написані мовою C#. Відомий веб-сайт stackoverflow.com написаний на C# з

використанням .NET framework. Веб-сайти Microsoft і Dell створені за подібним підходом.

C# використовується для розробки настільних програм Windows. Наприклад, Microsoft Visual Studio, Paint.NET написані на C#. Платформа .NET, а точніше Windows Forms, є основою для Skype, Microsoft Office і Photoshop.

Розробники Gamedev розуміють механізм Unity. Використовується для створення 2D і 3D комп'ютерних ігор. Програмування в Unity дозволяє вам зосередитися на вмісті вашої гри, не вдаючись у багато технічних деталей. В основному виконується на C#. Rust, Hearthstone, Fall Guys та інші створені в Unity за допомогою C#. Крім того, він дозволяє взаємодіяти з DirectX, набором компонентів, які використовуються для графіки та звуку в іграх.

C# також використовується для розробки мобільних додатків. Деякі платформи, такі як Xamarin, дозволяють запускати код C# у різних операційних системах, включаючи мобільні операційні системи Android та iOS. Нижче наведено список найпопулярніших програм, написаних за допомогою Xamarin.

У 2018 році бібліотека ML.NET була додана в середовище .NET, що дозволяє працювати з моделями машинного навчання. Вона не так добре задокументована, як бібліотека Python, але зручна і іноді використовується у виробництві.

Основні переваги мови програмування C#:

- Використання надійної платформи .NET, що постійно розвивається;
- Сувора статична типізація запобігає помилкам і робить код безпечнішим;
- Має велику спільноту розробників, можна швидко отримати відповіді на свої професійні запитання;
- Використовує широкий функціонал Visual Studio для створення додатків, консольних програм, програм з графічним інтерфейсом;
- Надає структуровану документацію;

- Універсальність додатків: Стандартні процедури та розробка сучасних додатків.

Варто відзначити, що C# поєднує в собі переваги C++ і Java, що робить його популярним серед розробників. Він також найбільш сумісний зі стандартом Common Language Infrastructure (CLI – інтерфейс для взаємодії користувачів з операційною системою).

**N- Tire архітектура.** Розробка систем обробки даних про замовлення передбачає роботу зі складною та заплутаною архітектурою. Для створення нашої інформаційної системи в якості основи було використано архітектурний стиль N-Tire. Цей особливий архітектурний стиль поділяє програму на окремі логічний і фізичний рівні [3].

Концепція рівнів використовується для ефективного розподілу завдань і обробки взаємозалежностей. Кожен рівень несе певні обов'язки, причому вищі рівні можуть використовувати послуги нижчих рівнів, але не навпаки.

Звичайна трирівнева програма складається з трьох рівнів: рівня презентації, середнього рівня та рівня бази даних [3]. Варто відзначити, що середній шар необхідний не завжди. У разі більш складних програм можуть бути додаткові рівні, крім стандартних трьох.

Presentation Layer (PL) є обличчям програми, оскільки саме цей рівень взаємодіє з користувачами. Він відповідає за відображення даних, зібраних з інших рівнів, та прийом введених користувачем даних для подальшої обробки. У веб-додатках PL зазвичай включає HTML, CSS, JavaScript, а також фреймворки для побудови інтерфейсів користувача, такі як React, Angular або Vue.js. У десктопних додатках це можуть бути бібліотеки та фреймворки, такі як WPF для .NET або JavaFX для Java.

Основна функція PL полягає в наданні інтуїтивно зрозумілого і зручного інтерфейсу для кінцевих користувачів. Він відповідає за відображення інформації та обробку подій, таких як кліки миші, натискання клавіш або введення даних у форми. PL відправляє запити до Business Logic Layer (BLL) для обробки даних і отримання необхідної інформації, після чого відображає

результати користувачеві. У випадку веб-додатків, він також може взаємодіяти з серверними API через AJAX-запити або WebSockets для динамічного оновлення сторінок без повного перезавантаження.

Business Logic Layer (BLL) є ядром додатку, де реалізована вся бізнес-логіка. Цей рівень забезпечує виконання бізнес-правил, обробку даних та виконання обчислень. BLL часто реалізується у вигляді сервісів або фасадів, які приймають запити від Presentation Layer і взаємодіють з Data Access Layer (DAL) для отримання та збереження даних. На цьому рівні здійснюються перевірки прав доступу, валідація введених даних і логічна обробка запитів.

BLL відповідає за виконання всіх правил і логіки, що визначають, як саме повинні оброблятися дані в системі. Наприклад, якщо система керує замовленнями, то BLL забезпечує обробку замовлення, обчислення вартості, перевірку наявності товарів на складі, обробку знижок і акцій, а також валідацію даних замовлення. Цей рівень також відповідальний за управління транзакціями, забезпечуючи цілісність даних під час складних операцій. BLL ізолює Presentation Layer від специфіки реалізації даних, забезпечуючи стандартизований інтерфейс для взаємодії [3].

Data Access Layer (DAL) відповідає за взаємодію з базами даних або іншими системами зберігання даних. Цей рівень реалізує методи для створення, читання, оновлення та видалення (CRUD) даних, використовуючи SQL-запити або інші механізми доступу до даних. DAL також може використовувати ORM (Object-Relational Mapping) інструменти, такі як Entity Framework для .NET або Hibernate для Java, для зручнішої роботи з об'єктами даних [3].

Основна функція DAL полягає в ізоляції Business Logic Layer від деталей реалізації доступу до даних. Це означає, що BLL не потребує знання про те, як саме і де зберігаються дані, він лише взаємодіє з DAL через чітко визначені інтерфейси. DAL забезпечує виконання SQL-запитів, управління підключеннями до бази даних, обробку результатів запитів і перетворення їх у об'єкти, які розуміє BLL. Такий підхід дозволяє легше змінювати або оптимізувати шари зберігання даних без впливу на бізнес-логіку [3].

**Загальні вигоди N-Tier архітектури.** Ця архітектура забезпечує чітке розділення обов'язків між різними рівнями, що дозволяє розробникам зосередитися на конкретних аспектах додатка під час розробки і тестування. Вона також сприяє кращій масштабованості, оскільки кожен рівень може бути масштабований незалежно від інших. Наприклад, ви можете збільшити кількість серверів для Presentation Layer для обробки великої кількості користувачів без необхідності змінювати Business Logic Layer або Data Access Layer.

**REST-архітектура.** Основним засобом написання PL-рівня для нашої інформаційної системи є архітертура REST (Representational State Transfer) яка є архітектурним стилем для побудови масштабованих веб-сервісів. Основою REST-архітектури є ресурси, кожен з яких ідентифікується унікальним URI (Uniform Resource Identifier). Ресурси можуть бути будь-чим, що представляє дані: користувачі, замовлення, товари і т.д. Клієнти взаємодіють з ресурсами через чітко визначені HTTP методи: GET, POST, PUT, DELETE, які відповідають CRUD-операціям (створення, читання, оновлення, видалення). Наприклад, GET /users отримує список користувачів, POST /users створює нового користувача, PUT /users/{id} оновлює існуючого користувача, а DELETE /users/{id} видаляє користувача з певним ID [18].

Однією з ключових особливостей REST є концепція представлення (representation). Ресурс може мати різні представлення, такі як JSON, XML, HTML тощо, в залежності від того, який формат потрібен клієнту. Клієнти та сервери обмінюються представленнями ресурсів за допомогою стандартних протоколів і заголовків HTTP, таких як Content-Type для вказівки формату даних. Це дозволяє гнучко взаємодіяти з ресурсами, використовуючи різні формати представлення, без зміни самого ресурсу [18].

Однією з фундаментальних властивостей REST є безстанність (statelessness). Це означає, що кожен запит від клієнта до сервера повинен містити всю необхідну інформацію для обробки цього запиту, без залежності від будь-якого попереднього запиту. Сервер не зберігає стан клієнта між запитами. Безстанність спрощує масштабування сервісу, оскільки сервер не потребує

зберігання сесійної інформації, і будь-який сервер у кластері може обробляти будь-який запит клієнта, забезпечуючи балансування навантаження.

REST підтримує кешування, що дозволяє зменшити навантаження на сервер і покращити продуктивність. Відповіді сервера можуть містити заголовки HTTP, такі як Cache-Control, які вказують клієнтам і проміжним проксі-серверам, як довго зберігати кешовані копії ресурсів. Наприклад, заголовок Cache-Control: max-age=3600 вказує, що ресурс може кешуватися протягом години. Кешування зменшує затримки та покращує продуктивність, особливо для ресурсів, які рідко змінюються.

**Переваги REST-архітектури.** REST-архітектура забезпечує простий і зрозумілий спосіб взаємодії з веб-сервісами завдяки використанню стандартних методів і форматів HTTP. Це зменшує криву навчання для розробників і полегшує інтеграцію з іншими системами. Клієнти можуть використовувати широко доступні інструменти і бібліотеки для побудови та обробки HTTP-запитів, що спрощує розробку і тестування додатків [18].

Завдяки безстанності та підтримці кешування, REST-архітектура добре підходить для масштабованих систем, що обслуговують велику кількість одночасних запитів. Без необхідності зберігання стану сесії сервери можуть легко обробляти запити незалежно один від одного, що сприяє кращому балансуванню навантаження і підвищує продуктивність системи в цілому. Кешування зменшує навантаження на сервери і прискорює час відповіді, що особливо важливо для часто запитуваних ресурсів.

Таким чином, REST-архітектура є потужним і гнучким підходом до побудови веб-сервісів, який поєднує простоту, масштабованість і стандартизацію, що робить її популярною серед розробників і забезпечує ефективну інтеграцію різних систем.

**Фреймворк ASP.NET Core 6**. Для розробки нашої ІС також було використано популярний фреймворк ASP.NET Core 6, який побудований з урахуванням принципів модульності та розширюваності, що робить його ідеальним для розробки за N-Tier архітектурою. Фреймворк підтримує модульну

структуру, де різні частини додатку можуть бути легко відокремлені, розроблені та протестовані окремо. Це дозволяє ефективно розділяти обов'язки між Presentation Layer, Business Logic Layer і Data Access Layer, сприяючи більш чистій і зрозумілій організації коду. Dependency Injection, вбудована в ASP.NET Core, полегшує впровадження сервісів на різних рівнях, забезпечуючи низьку зв'язаність і високу тестованість [10,12].

## 2.3 Огляд можливостей середовища розробки

Підтримуючи різні операційні системи, включаючи Windows, MacOS і Linux, Visual Studio Code — це редактор вихідного коду, який забезпечує баланс між зручністю для користувача та надійною функціональністю. [13]

Visual Studio Code підтримує широкий список середовищ виконання та розширень, а також широкий спектр популярних мов програмування. Список мов програмування та фреймворків включає JavaScript, TypeScript, Node.js, C++, C#, Java, Python, PHP, Go, .NET і Unity.

Щоб забезпечити зручність для користувачів, VS Code включає різні функції, такі як вбудований налагоджувач, інструменти репозиторію Git, підсвічування синтаксису, інструменти рефакторингу та IntelliSense, який надає пропозиції щодо автозавершення на основі скорочень команд. Microsoft представила VS Code IDE на конференції Blink 28 травня 2015 року, а згодом зробила його доступним для використання за ліцензією MIT 19 жовтня 2015 року.

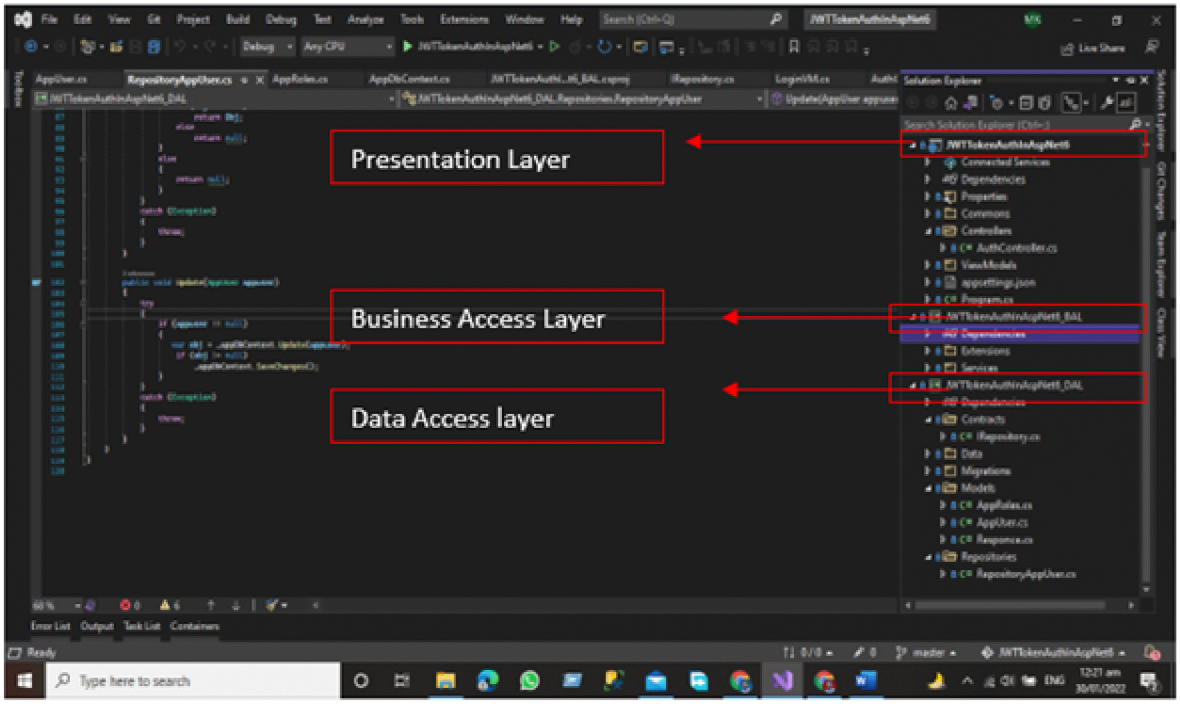



Рисунок 2.1 – Інтерфейс середовища розробки IDE Visual Studio Code.

Для роботи з трирівневою архітектурою на мові C# з використанням фреймворка ASP.NET Core 6, необхідно імпортувати низку бібліотек, які забезпечать відповідні функціональні можливості для кожного рівня архітектури[5]:

**Microsoft.AspNetCore.Mvc.** Ця бібліотека є основою для побудови веб-додатків та API в ASP.NET Core. Вона надає всі необхідні компоненти для створення контролерів, обробки HTTP запитів і формування відповідей. Microsoft.AspNetCore.Mvc підтримує атрибути маршрутизації, моделі контролерів і дій, фільтри для авторизації та валідації, а також інтеграцію з форматами представлення даних, такими як JSON. Ця бібліотека також включає підтримку Razor Pages і MVC, що дозволяє легко створювати динамічні веб-сторінки і API-інтерфейси.

**Microsoft.AspNetCore.Http.** Забезпечує доступ до об'єктів HTTP запитів і відповідей, таких як HttpRequest та HttpResponse, що необхідні для обробки і маніпулювання даними, переданими між клієнтом і сервером. Вона включає можливості для роботи з куками, сесіями, файлами та іншими HTTP-функціями.

**Microsoft.Extensions.DependencyInjection.** Бібліотека є основною для впровадження залежностей (Dependency Injection) в ASP.NET Core. Вона дозволяє реєструвати і налаштовувати сервіси, які реалізують бізнес-логіку, таким чином, що вони можуть бути легко використані в інших частинах програми. Dependency Injection сприяє зменшенню зв'язаності коду та покращенню тестованості додатка.

**FluentValidation.** Ця бібліотека забезпечує зручний і гнучкий спосіб для валідації бізнес-логіки. Вона дозволяє визначати правила валідації для моделей даних і забезпечує інтерфейси для виконання цих перевірок. FluentValidation легко інтегрується з ASP.NET Core і може бути використана для валідації вхідних даних у контролерах.

**Microsoft.EntityFrameworkCore.** Entity Framework Core (EF Core) є ORM (Object-Relational Mapping) бібліотекою для .NET. Вона дозволяє розробникам працювати з базами даних за допомогою об'єктно-орієнтованого підходу, замість написання SQL-запитів. EF Core підтримує різні провайдери баз даних, такі як SQL Server, PostgreSQL, MySQL та інші. Ця бібліотека забезпечує можливості для створення і міграції схем баз даних, роботи з транзакціями і виконання запитів LINQ.

**Microsoft.Extensions.Configuration.** Бібліотека забезпечує можливості для роботи з конфігураціями додатка, такими як рядки підключення до бази даних, налаштування служб та інші параметри. Вона підтримує різні джерела конфігурації, включаючи файли JSON, змінні середовища, параметри командного рядка та секрети користувача.

**Dapper.** Dapper є мікро-ORM бібліотекою для .NET, яка забезпечує високу продуктивність і легкість використання при виконанні SQL-запитів. На відміну від EF Core, Dapper не пропонує автоматичного відображення об'єктів і міграцій, але забезпечує швидкий і ефективний доступ до баз даних, що може бути корисним для специфічних сценаріїв або оптимізації продуктивності.

## 2.4. Обгрунтування та вибір структури бази даних

Для розробки цієї інформаційної системи було вирішено використовувати систему керування реляційними базами даних MS SQL Server. Ця СУБД має багато переваг, серед яких[4,7,19]:

- Для взаємодії з C# ASP.NET Entity Framework підтримує: MS SQL Server глибоко інтегровано з Entity Framework, популярним ORM C# (Object Relational Mapping). Це спрощує взаємодію з базою даних і дозволяє використовувати об'єктно-орієнтований підхід до обробки даних.
- Висока продуктивність: MS SQL Server відомий своєю високою продуктивністю та ефективністю обробки запитів. Це особливо важливо для веб-додатків, де швидкий доступ до даних є критичним фактором успіху.
- Інтелектуальна обробка запитів: MS SQL Server має вбудовану систему оптимізації запитів, яка використовує статистику даних, що дозволяє автоматично вибрати найкращий план виконання запиту. Це допомагає ефективно використовувати ресурси сервера та покращувати продуктивність системи.
- Гарна безпека даних: MS SQL Server надає декілька механізмів для забезпечення безпеки даних, включаючи автентифікацію, авторизацію, шифрування та аудит доступу до бази даних. Це дозволяє захистити конфіденційну інформацію від несанкціонованого доступу та зберегти цілісність даних.
- Масштабованість: MS SQL Server може обробляти великі обсяги даних і ефективно масштабуватися за допомогою розподіленої архітектури, кластеризації та розподіленого розгортання. Це забезпечує високу доступність і продуктивність системи, навіть якщо обсяги даних збільшуються.

Таким чином, використання MS SQL Server із C# ASP.NET забезпечує надійне й ефективне зберігання та обробку даних, а також розширені функції обробки даних і безпеки у веб-додатках.

# 3. ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ УПРАВЛІННЯ ДАНИМИ З ОБРОБКИ ТА ДОСТАВКИ ЗАМОВЛЕНЬ

## 3.1 Формування бази даних для збереження та управління даними користувачів інфомаційної системи

В рамках реалізації нашої інформаційної системи було розгорнуто дві бази даних з використанням MS SQL Server. В одній з них зберігаються дані реєстрації та авторизації користувача. Для цього використовується ASP.NET Core Identity.

ASP.NET Core Identity — це структура, яка забезпечує механізми автентифікації та авторизації для веб-додатків на платформі ASP.NET Core. Це дозволяє розробникам створювати безпечні системи авторизації, керувати користувачами, їхніми ролями та правами доступу[10].

Основні компоненти ASP.NET Core Identity включають [10]:

- Користувачі та ролі: керує реєстрацією користувачів і налаштуваннями їхніх ролей.
- Хешування паролів: безпечне зберігання паролів за допомогою хешів.
- Керування ролями та правами доступу: визначте ролі користувачів і надайте права доступу.
- Налаштування та розширення: налаштувати та розширити функціональність за допомогою налаштувань та інтерфейсу.

За результатами проектування розроблено базу даних яка використовується для зберігання та керування даними користувача для нашої інформаційної системи, схему якої показано на рис. 3.1.

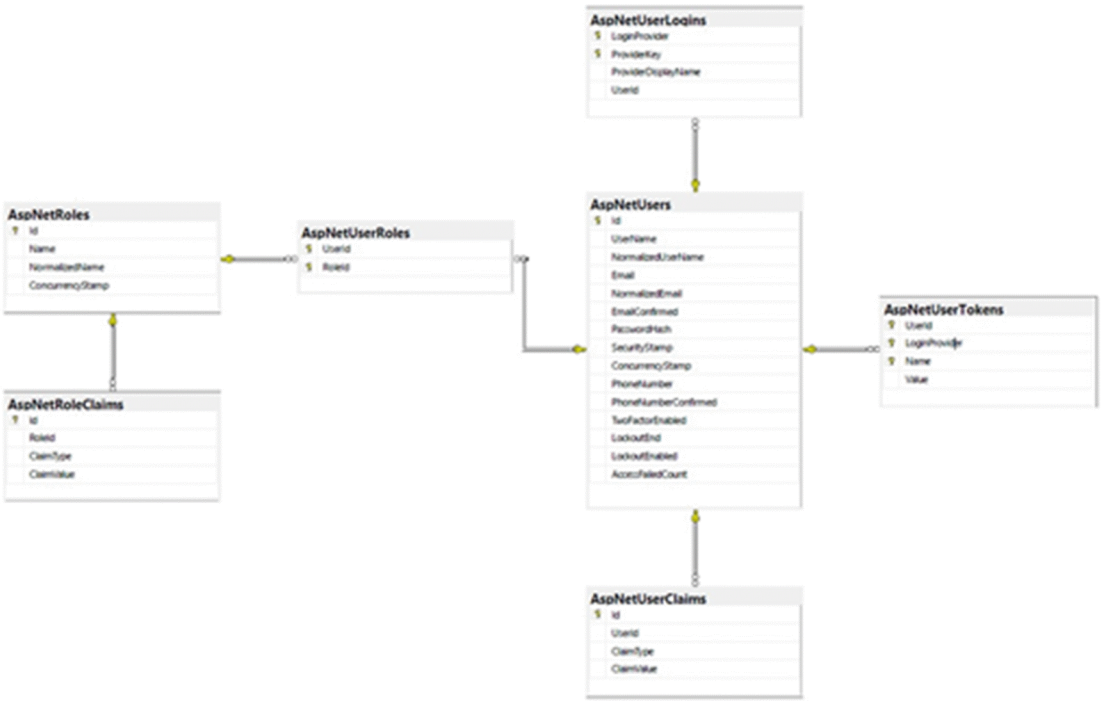



Рисунок 3.1 – Схема бази даних для збереження та управління даними користувачів ІС.

Також розроблено базу даних яка використовується використовується для обробки даних по замовленнях користувачів, схему якої показано на рис. 3.2.

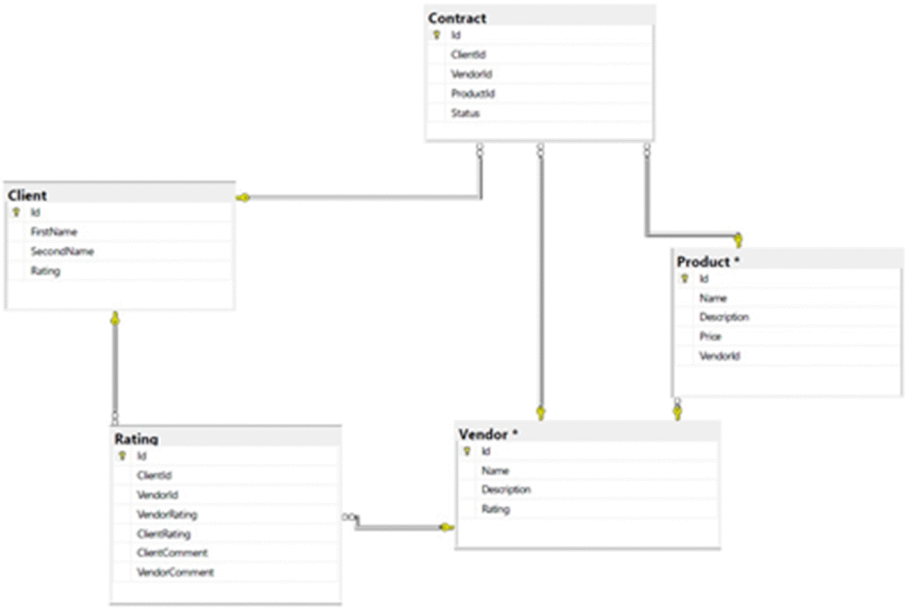



Рисунок 3.2 – Схема бази даних для обробки даних по замовленнях користувачів ІС

База даних для системи обробки даних замовлення містить кілька таблиць, як зазначено в таблиці 3.1. У цій таблиці подано назви та стислі описи кожної таблиці. Для більш повного розуміння структури бази даних таблиці 3.2-

3.6 пропонують детальний опис кожної окремої таблиці у створених БД для нашої інформаційної системи.

Таблиця 3.1 – Опис вмісту таблиць бази даних інформаційної системи

| Назва таблиці БД | Опис вмісту |
|---|---|
| Clients | Містить дані про покупців |
| Vendor | Містить дані про продавців |
| Contract | Містить дані про контракти |
| Product | Містить дані про продукти продавців |
| Rating | Містить дані про взаємний рейтинг |

Таблиця 3.2 – Структура таблиці Clients

| Стовбець | Характеристика | Тип даних | Ключ таблиці |
|---|---|---|---|
| Id | Ідентифікатор | Integer | PK |
| LastName | Імя покупця | Nvchar | |
| FirstName | Прізвище | Nvchar | |
| Rating | Рейтинг покупця | Float | |

Таблиця 3.3 – Структура таблиці Vendor

| Стовбець | Характеристика | Тип даних | Ключ таблиці |
|---|---|---|---|
| Id | Ідентифікатор | Integer | PK |
| Name | Назва продавця | Nvchar | |
| Description | Опис продавця | Nvchar | |
| Rating | Рейтинг продавця | Float | |

Таблиця 3.4 – Структура таблиці Product

| Стовбець | Характеристика | Тип даних | Ключ таблиці |
|---|---|---|---|
| Id | Ідентифікатор | Integer | PK |
| Name | Назва товару | Nvchar | |
| Description | Опис товару | Nvchar | |

Продовження табл. 3.4.

| | | | |
|---|---|---|---|
| VendorID | Ідентифікатор вендора | Integer | FK |
| Price | Ціна товару | Float | |

Таблиця 3.5 – Структура таблиці Contract

| Стовбець | Характеристика | Тип даних | Ключ таблиці |
|---|---|---|---|
| Id | Ідентифікатор | Integer | PK |
| VendorID | Ідент. вендора | Integer | FK |
| ClientID | Ідент. клієнта | Integer | FK |
| ProductID | Ідент. товару | Integer | FK |
| Status | Наявність товару | Nvchar | |

Таблиця 3.6 – Структура таблиці Rating

| Стовбець | Характеристика | Тип даних | Ключ таблиці |
|---|---|---|---|
| Id | Ідентифікатор | Integer | PK |
| VendorID | Ідент. вендора | Integer | FK |
| ClientID | Ідент. клієнта | Integer | FK |
| VendorRating | Рейтинг вендора | Integer | |
| ClientRating | Рейтинг покупця | Nvchar | |
| VendorComment | Коментар про продавця | Nvchar | |
| ClientComment | Коментар про покупця | Nvchar | |

## 3.2 Розробка інформаційної системи управління даними з обробки та доставки замовлень

Система обробки даних про замовлення заснована на архітектурі N-Tire і має 3 рівня (Presentation Layer, Business Logic Layer, Data Access Layer) [3].

Спочатку опишемо реалізацію рівня доступу до бази даних (DAL)

Для взаємодії з проектованою інформаційною системою було окремо виділено декілька сутностей (Entity)

- Замовлення(Contract);
- Покупець(Client)
- Товар(Product);
- Рейтинг(Rating);
- Продавець(Vendor).

Таблиця 3.7 перераховує репозиторії рівня DAL. Дані репозиторії безпосередньо взаємодіють з базою даних нашої інформаційної системи

Таблиця 3.7. Опис репозиторіїв Data Access Layer

| Назва репозитарію | Призначення та функції |
|---|---|
| Product Repository | Додавання товару<br>Видалення товару<br>Оновлення інформації про товар<br>Отримання товару<br>Отримання списку всіх товарів від певного продавця |
| Client Repository | Додавання покупця<br>Видалення покупця<br>Оновлення інформації про покупця<br>Отримання покупця<br>Отримання списку всіх покупців |

Продовження табл. 3.7

| | |
|---|---|
| Contract Repository | Додавання замовлення<br>Видалення замовлення<br>Оновлення замовлення<br>Оновлення статусу замовлення<br>Отримання замовлення<br>Отримання замовлення за id його покупця<br>Отримання переліку замовлень за id їх продавця<br>Отримання замовлень по id товару |
| Rating Repository | Додавання показника рейтингу<br>Видалення показника рейтингу<br>Оновлення показника рейтингу<br>Одержання усіх сутностей рейтингу по id покупця<br>Одержання усіх сутностей рейтингу по id продавця |
| Vendor Repository | Додавання продавця<br>Видалення продавця<br>Оновлення інформації про продавця Отримання інформації про продавця<br>Отримання списку всіх продавців |

Таблиця 3.8 перераховує репозиторії рівня BLL та надає вичерпну інформацію про призначення та функції, які вони виконують в інформаційній системі.

Таблиця 3.8. Опис репозиторіїв Business Logic Layer

| Назва репозитарію | Призначення та функції |
|---|---|
| Client Service | Створення нового покупця<br>Видалення покупця<br>Оновлення інформації про покупця<br>Одержання покупця<br>Одержання списку всіх покупців |

Продовження табл. 3.8

| | |
|---|---|
| Product Service | Додавання товару<br>Видалення товару<br>Оновлення інформації про товар<br>Одержання товару за його ідентифікатором<br>Одержання списку всіх товарів від продавця |
| Contract Service | Створення замовлення<br>Видалення замовлення<br>Оновлення замовлення<br>Оновлення статусу замовлення<br>Одержання замовлення за його id<br>Одержання списку замовлень за id покупця<br>Одержання списку замовлень за id продавця<br>Отримання списку замовлень за id товару |
| Rating Service | Видалення показника рейтингу<br>Встановлення чи оновлення рейтингу покупця<br>Встановлення чи оновлення рейтингу продавця<br>Встановлення чи оновлення відгуку від унікального покупця<br>Встановлення чи оновлення відгуку від продавця<br>Одержання показника рейтингу за id |

| | |
|---|---|
| | Одержання списку сутностей показника рейтингу за id покупця<br>Одержання списку сутностей показника рейтингу за id продавця |
| Vendor Service | Створення нового продавця<br>Видалення продавця<br>Оновлення інформації про продавця<br>Одержання інформації про продавця за його ідентифікатором<br>Одержання списку усіх продавців |

І нарешті, щоб завершити опис реалізації архітектурних рішень в проектованій нами інформаційній системі, опишемо реалізацію останнього рівня архітектури N-tire, а саме рівень PL (Presentation Layer).

Використовуючи технологію Web API, рівень PL забезпечує безперебійний зв'язок між клієнтом і сервером за допомогою запитів HTTP. Це досягається за допомогою реалізації спеціальних методів, таких як HttpGet, який дозволяє отримувати дані з сервера. Надсилаючи запит HTTP за допомогою методу GET на вказану URL-адресу сервера, клієнт може отримати потрібні дані у відповіді сервера.

Метод HTTP PUT використовується для зміни наявних даних, що зберігаються на сервері. Щоб оновити дані, клієнт може надіслати запит HTTP PUT на відповідну URL-адресу сервера, надаючи нові дані, які потрібно оновити. Після отримання сервер продовжить відповідне оновлення даних на своєму кінці.

Метод HttpPost служить для додавання свіжої інформації на сервер. Здійснюючи HTTP-запит POST до відповідної URL-адреси сервера, клієнт може передати нові дані, які потрібно включити, і сервер продовжить зберігати ці дані на своєму кінці.

Метод DELETE, відомий як HttpDelete, використовується для видалення даних із сервера. Надсилаючи запит HTTP за допомогою методу DELETE на відповідну URL-адресу сервера, клієнт ініціює видалення відповідних даних із бази даних сервера.

Використовуючи методи HTTP, користувачі можуть ефективно взаємодіяти з сервером через веб-API на рівні PL. Це дозволяє безперешкодно керувати даними, включаючи пошук, зміну, додавання та видалення за допомогою стандартних запитів HTTP. Таблиця 3.9 описує різні контролери та їхні відповідні функції, які працюють на рівні PL, надаючи короткий огляд їх призначення.

Таблиця 3.9. Опис контролерів рівня бізнес-логіки

| Назва контролера | Призначення та функції |
|---|---|
| Vendor Controller | Вміщує HTTP-запити для додавання, оновлення, видалення, одержання списку продавців. |
| Clients Controller | Вміщує HTTP-запити для додавання, оновлення, видалення, одержання списку покупців. |
| Contract Controller | Вміщує HTTP-запити для додавання, оновлення, видалення, одержання списку замовлень. |
| Rating Controller | Вміщує HTTP-запити для додавання, оновлення, видалення, одержання показників рейтингу покупців і продавців. |
| Account Controller | Містить HTTP-запити які відповідальні за реєстрацію та авторизацію покупців і продавців |

Продовження табл. 3.9

| | |
|---|---|
| Product Controller | Вміщує HTTP-запити для додавання, оновлення, видалення, одержання списку товарів |

Завдяки реалізації підходу багаторівневої архітектури було створено систему з кількома рівнями програм, які сприяють бездоганній взаємодії. При розробці програмного коду для цих модулів були застосовані широко поширені принципи SOLID, що призвело до створення кодової бази, яка є легко зрозумілою та адаптованою. Підходи RESTful були використані для розробки API для всіх служб. Для інтерфейсу користувача було використано широко поширене поєднання HTML, CSS і JavaScript, що призвело до створення динамічного та надійного веб-інтерфейсу, який також реагує на екрани різних розмірів.

### 3.3 Опис різних сценаріїв використання розробленої інформаційної системи управління даними з обробки замовлень.

Почнемо з розгляду сценарію, коли користувач може бути зареєстрований як покупець. Цей конкретний сценарій відбувається за посиланням /api/account/register. На малюнку 3.9 представлено візуальне представлення запиту на реєстрацію покупця.

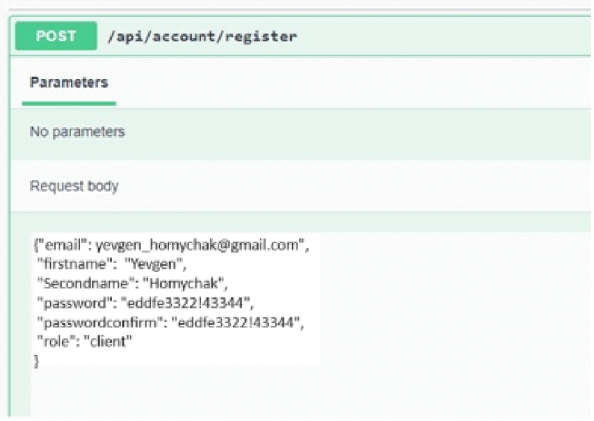



Рисунок 3.9 – Запит на реєстрацію покупця в системі

Новий запис генерується в базі даних як частина процесу виконання сценарію. Це можна спостерігати на малюнку 3.9, де показано включення нового запису в клієнтську базу даних.

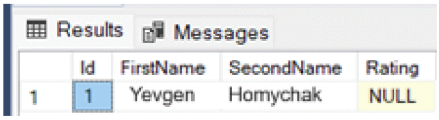



Рисунок 3.9 – Відображення нового запису в базі даних покупців

Наступний сценарій показує можливість покупця переглядати весь вміст товарів в наявності. Це відбувається за наступним посиланням /api/Products/{id}, де id — це ідентифікатор продавця. На малюнку 3.10 показано запит на одержання списку всіх продуктів продавця.

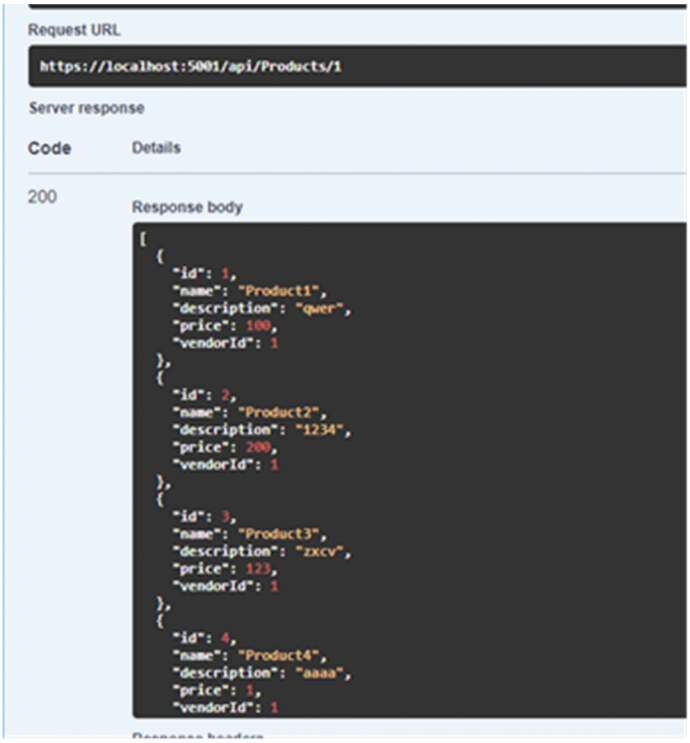



Рисунок 3.10. – Запит на одержання  списку усіх товарів продавця

Ще однією особливістю виконання вищеописаного сценарію є отримання всіх записів з бази даних продавця. На малюнку 3.11 показано базу даних товарів продавця.

| | Id | Name | Description | Price | VendorId |
|---|---|---|---|---|---|
| 1 | 1 | Product1 | qwer | 100.00 | 1 |
| 2 | 2 | Product2 | 1234 | 200.00 | 1 |
| 3 | 3 | Product3 | zxcv | 123.00 | 1 |
| 4 | 4 | Product4 | aaaa | 1.00 | 1 |

Рисунок 3.11 – Результат виводу бази даних товарів продавця

Наступний сценарій дозволяє продавцям додавати новий товар. Це відбувається за наступним посиланням /api/Products/. На малюнку 3.12 показано запит на створення нового товару.

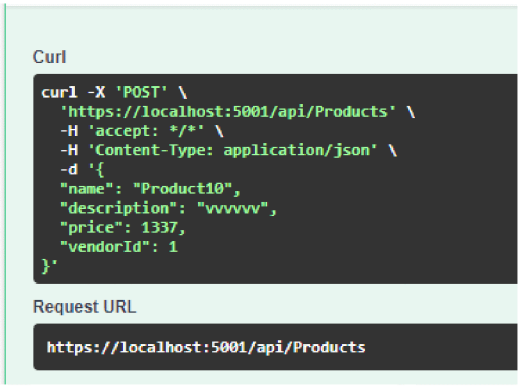



Рисунок 3.11 – Запит на створення нового товару продавцем

Реалізація згаданого сценарію також передбачає включення нового запису в базу даних. На рис 3.12 наочно демонструється результат включення нового товару в базу даних товарів.

| | Id | Name | Description | Price | VendorId |
|---|---|---|---|---|---|
| 1 | 1 | Product1 | qwer | 100.00 | 1 |
| 2 | 2 | Product2 | 1234 | 200.00 | 1 |
| 3 | 3 | Product3 | zxcv | 123.00 | 1 |
| 4 | 4 | Product4 | aaaa | 1.00 | 1 |
| 5 | 5 | Product10 | vvvvv | 1337.00 | 1 |

Рисунок 3.12 – Результат включення нового товару в базу даних товарів

Четвертий сценарій передбачає можливість для продавця вилучити продукт. Ця дія виконується за певною URL-адресою: /api/Products/{id}, де {id} представляє унікальний ідентифікатор продукту що видаляється.

Видалення продукту позначається запитом до бази даних на видалення продукту, який показано на рис. 3.13.

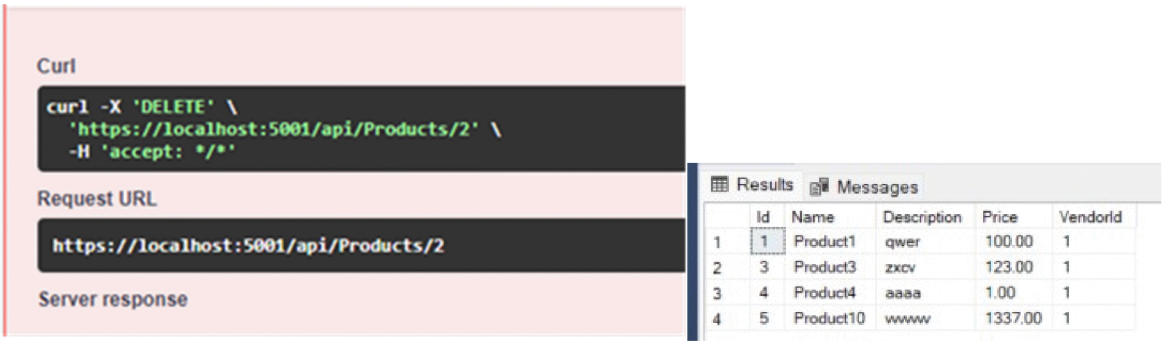



Рисунок 3.13 – Запит на видалення даних про продукт та результ виводу в базі даних

Наступний сценарій демонструє можливість створювати замовлення. Це відбувається за наступним посиланням /api/Contracts/. На малюнку 3.14 зображено запит на створення замовлення.


Рисунок 3.14 – Запит до ІС на створення замовлення

Виконання описаного вище сценарію також характеризується оновленням записів у базі даних замовлень. На малюнку 3.15 показано створення даних замовлення в базі даних замовлень.

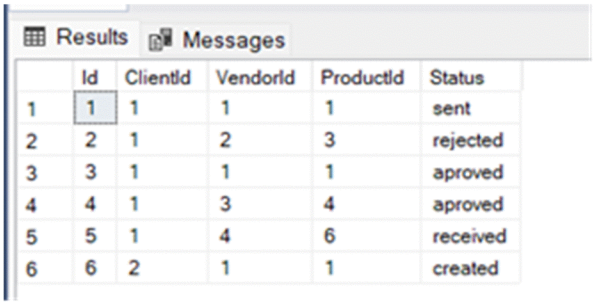
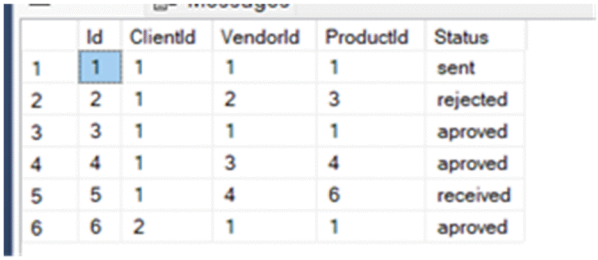



Рисунок 3.15 – Створення даних замовлення в базі даних замовлень.

Шостий сценарій показує можливість оновлення статусу замовлення. Це відбувається за наступним посиланням /api/Contracts/. На малюнку 3.16 показано запит на оновлення даних замовлення.


Рисунок 3.16 – Запит до БД на оновлення даних по замовленню

Виконання описаного вище сценарію також характеризується оновленням записів у базі даних замовлень. На рисунку 3.16 показано результат оновлення даних про товар у базі даних замовлень.



| | Id | ClientId | VendorId | ProductId | Status |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | sent |
| 2 | 2 | 1 | 2 | 3 | rejected |
| 3 | 3 | 1 | 1 | 1 | aproved |
| 4 | 4 | 1 | 3 | 4 | aproved |
| 5 | 5 | 1 | 4 | 6 | received |
| 6 | 6 | 2 | 1 | 1 | aproved |

Рисунок 3.16 – Оновлення даних замовлення в базі даних замовлень

У сьомому сценарії замовлення можна отримати на основі ідентифікатора покупця за допомогою спеціального посилання, -/api/Contract/ByClientId. Цю функцію показано на рис 3.17, який демонструє запит на отримання замовлень, пов’язаних із певним ідентифікатором клієнта.

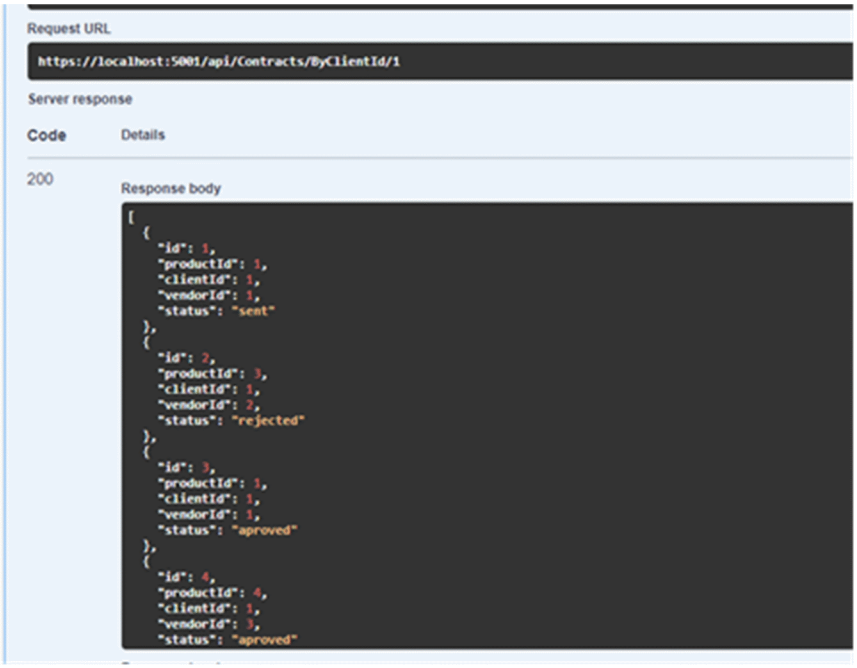



Рисунок 3.17 – Запит на отримання замовлень за ідентифікатором покупця

Реалізація описаного вище сценарію передбачає отримання замовлень з бази даних на основі ідентифікатора покупця. На рис. 3.18 показано візуальне представлення бази даних замовлень.

| | Id | ClientId | VendorId | ProductId | Status |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | sent |
| 2 | 2 | 1 | 2 | 3 | rejected |
| 3 | 3 | 1 | 1 | 1 | aproved |
| 4 | 4 | 1 | 3 | 4 | aproved |
| 5 | 5 | 1 | 4 | 6 | received |
| 6 | 6 | 2 | 1 | 1 | aproved |

Рисунок 3.18 – Візуальне представлення бази даних замовлень.

За вказаним посиланням /api/Rating/ восьмий сценарій передбачає створення рейтингу. Рисунок 3.19 ілюструє процес створення запиту на створення рейтингу .

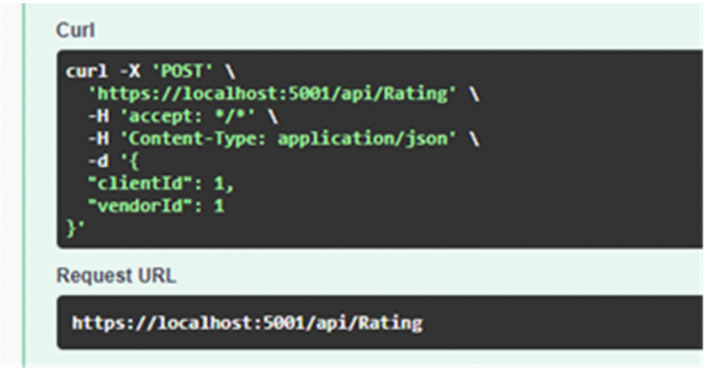



Рисунок 3.19 – Запит до ІС на створення рейтингу

Реалізація описаного сценарію також передбачає оновлення запису в рейтинговій базі. На рисунку 3.20 зображено процес генерації нового запису рейтингу в базі даних.

Results Messages

| | Id | ClientId | VendorId | VendorRating | ClientRating | ClientComment | VendorComment |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 2 | 5 | 5 | a | b |
| 2 | 2 | 1 | 1 | 4 | 4 | aa | bb |
| 3 | 3 | 1 | 1 | 5 | 2 | q | q |
| 4 | 4 | 2 | 2 | 3 | 3 | asdasd | asdasda |
| 5 | 5 | 1 | 1 | NULL | NULL | NULL | NULL |

Рисунок 3.20 – Процес генерації нового запису рейтингу в базі даних

Останні два розглянуті нами сценарії роботи проектованої інформаційної системи відображають можливість отримання рейтингу за ідентифікатором, відповідно покупця або продавця. Вони відбуваються за такими посиланнями як /api/Rating/ByClientId або /api/Rating/ByVendorId. На рисунках 3.21 та 3.22 зображено запит отримання рейтингу за ідентифікатором покупця та продавця відповідно.

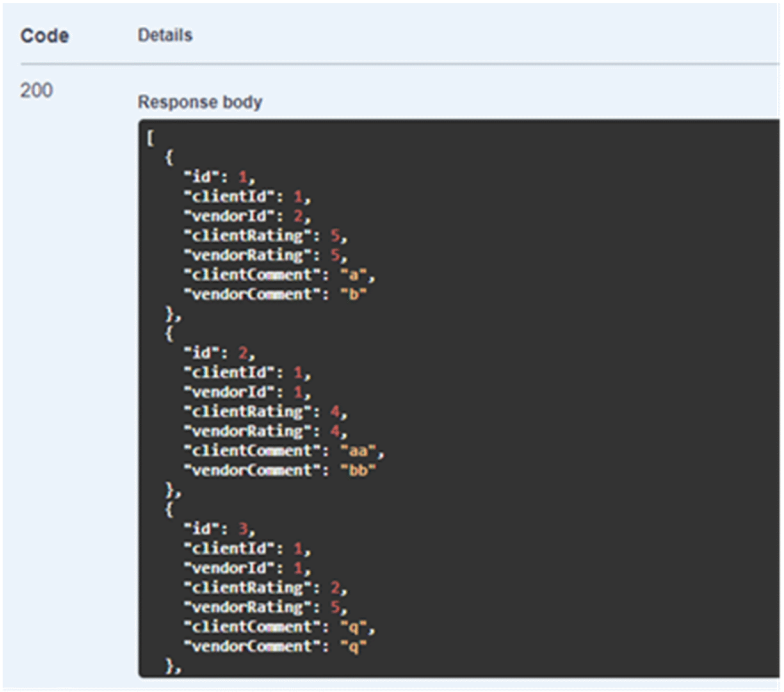



Рисунок 3.21 – Запит до БД на отримання рейтингу за ідентифікатором покупця

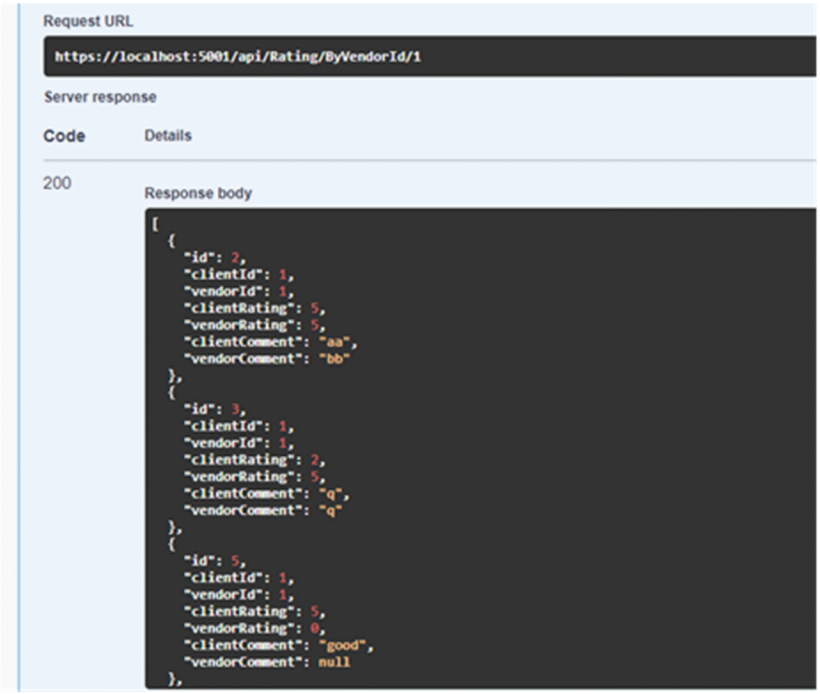



Рисунок 3.22 – Запит до БД на отримання рейтингу за ідентифікатором продавця

Для кращого розуміння взаємодії між контролерами зображень і результатами запитів у базі даних деталі сценаріїв представлені на скріншотах. Цей метод було обрано через його здатність спростити розуміння взаємозв’язку цих двох компонентів.

# 4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

## 4.1. Структурно-функціональний аналіз виробничого процесу та розроблення моделі травмонебезпечних ситуацій

У зображеннях процесів формування, виникнення аварій та виробничих травм усі випадкові події, що утворюють конкретну аварійну ситуацію, пов'язані між собою причинно-наслідковими зв'язками.

Метод логічного моделювання потенційних аварій, травм та катастроф відкриває можливість розробити досконалу систему управління ОП виробництва, яка базується на оперативному пошуку виробничих небезпек, їх глибокому аналізі й терміновому прийнятті заходів для усунення потенційних небезпек ще до виникнення травмонебезпечних та катастрофічних ситуацій. Деякі небезпечні ситуації в табл. 4.1.

Працівники, що обслуговують електрообладнання вениляційної системи, зобов'язані знати Правила безпечної експлуатації електроустановок споживачів відповідно до займаної посади або роботи, як вони виконують, і мати відповідну групу з електробезпеки [22,23].

Працівники, що порушили вимоги Правил безпечної експлуатації електроустановок, усуваються від роботи і несуть відповідальність (дисциплінарну, адміністративну, кримінальну) згідно з чинним законодавством. Такі працівники не допускаються до робіт в електроустановках без позачергової перевірки знань вимог правил безпечної експлуатації електроустановок.

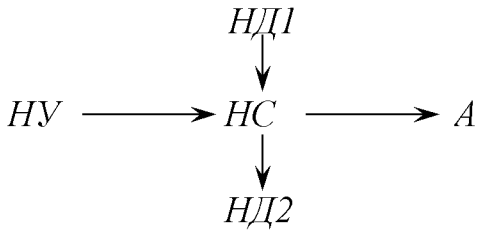
Забороняється допускати до роботи в електроустановках осіб, які не пройшли навчання і перевірку знань Правил безпечної експлуатації електроустановок.

Працівнику, який пройшов перевірку знань Правил безпечної експлуатації електроустановок, видається посвідчення встановленої форми.

Таблиця 4.1. Моделювання процесів формування та виникнення травмонебезпечних і аварійних ситуацій

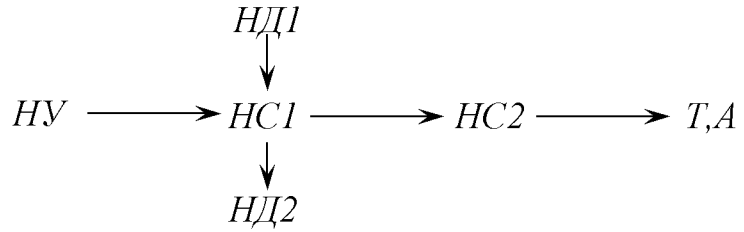
| Вид робіт | Виробнича небезпека | | | Можливі наслідки | Заходи запобігання небезпечним ситуаціям |
|---|---|---|---|---|---|
| | Небезпечна умова (НУ) | Небезпечна дія (НД) | Небезпечна ситуація (НС) | | |
| Використання механічної вентиляції | Оператор не перевірив обладнання НУ | Пошкоджений трубопровід мережі НД1<br>Закупорений трубопровід шланга НД2 | Відмова вентиляційної системи (двигуна) НС | Аварія | Розвісити плакати, провести інструктажі із експлуатації обладнання системи |

Модель процесу:




| | | | | | |
|---|---|---|---|---|---|
| Використання електронних пристроїв регулювання | Пошкоджена ізоляція провідників з'єднання НУ | Пробій на корпус НД1<br>Коротке замикання НД2 | Ураження людини електричним струмом НС1<br>Виведення обладнання із ладу НС2 | Травма<br>Аварія | Заміна провідників, установлення захисного обладнання (запобіжників, захист від ураження людини струмом) тощо |

Модель процесу:



Посвідчення про перевірку знань працівника є документом, який засвідчує право на самостійну роботу в електроустановках на зазначеній посаді за фахом.

## 4.2. Вимоги техніки безпеки під час роботи обладнання та протипожежні заходи

**Вимоги правил техніки безпеки перед початком роботи.** Для початку роботи пов'язаної з вентиляцією вимикають рубильники або автоматичні вимикачі щита низької напруги, запирають шафу і вивішують попереджувальні плакати. Також повинні бути основні захисні засоби до яких належать такі, ізоляція яких надійно захищає від робочої напруги мережі і за допомогою яких можна дотикатися до струмопровідних частин, що перебувають під напругою, без небезпеки ураження електричним струмом (інструмент з ізольованими ручками, ізолюючі струмовимірювальні кліщі, діелектричні рукавиці ).

**Вимоги правил техніки безпеки під час роботи.** Виконавши ці операції, надівають діелектричні рукавиці і за допомогою покажчика напруги перевіряють відсутність напруги на всіх фазах. Потім, приєднавши один кінець переносного заземлення до заземлюючого пристрою, накладають його на струмоведучі частини. Після цього остаточно приступають до роботи.

**Вимоги правил техніки після закінчення роботи.** Після закінчення роботи системи перед її вимиканням необхідно виконати такі технічні операції: перевірити надійність кріплення, зняти переносні тимчасові заземлення, відімкнути щит низької напруги і зняти плакати з техніки безпеки; якщо тимчасове переносне заземлення встановлене на лінії, його також треба зняти тощо [22].

**Протипожежні заходи на об'єкті.** Для запобігання пожеж на об'єкті розроблено організаційні, експлуатаційні, технічні режимного характеру, пожежно-евакуаційні, профілактичні заходи. До організаційних заходів

відносяться правила розміщення машин, що обслуговують приміщення, обладнання, матеріалів з дотримання певних проходів, не допускається захаращення приміщень, проходів і т.д.

### 4.3. Розрахунок штучного заземлення

Вибір штучного заземлення проводиться в залежності від характеру ґрунту і способу забивання стержнів [23]. Розраховуємо заземлюючий контур підстанції напругою 10/0,4 кВ з глухозаземленою нейтраллю. Характер ґрунту – чорнозем з $\rho=2\cdot10^4$ Ом·см. Кліматична зона – IV ($K_c$ – 1,2, $K_n$ – 1,5 ). Струм замикання на землю в мережі становить 50 А.

В відповідності з діючими правилами, опір заземлюючого пристрою повинен становити

$$R = \frac{125}{I_з} = \frac{125}{50} = 2,5 Ом, \qquad (4.1)$$

де $I_з$ – струм замикання на землю, А.

Приймаємо 3 Ом. Контур заземлення розміщуємо в ряд з а = 5 м, l = 2,5 м. В якості стержневого заземлювача приймаємо кутникові сталь 50х50х5 мм, а протяжного – пластинчасту сталь 40х4 мм.

Опір одиночного стержня становить:

$$R_o = 0.00318\rho \cdot K_c, Ом \qquad (4.2)$$

де $K_c$ – коефіцієнт сезонності для стержневого заземлювача ($K_c$ – 1,2).

$$R_o = 0.00318 \cdot 2 \cdot 10^4 \cdot 1.2 = 76.32 Ом.$$

Число стержнів приймаємо 15. При цьому коефіцієнт використання стержневих заземлювачів становить $\eta_c$ = 0,7. Опір всіх стержнів розтікання струму становить:

$$R_c = \frac{R_o}{n \cdot \eta_c}, Ом, \qquad (4.3)$$

де $n$ – число стержнів, шт.

$$Rc = \frac{76.32}{15 \cdot 0.7} = 7.3Ом.$$

Довжина протяжного заземлювача становить $l$ = 35 м (3500 см); приймаємо $t$ = 50 см, $b$ = 0,4 см. Опір протяжного заземлювача становить:

$$R_{пр} = \frac{0,366}{l} \cdot \rho \cdot 2 \cdot \lg \frac{2 \cdot l^2}{t \cdot b}, Ом \qquad (4.4)$$

$$R_{пр} = \frac{0,366}{3500} \cdot 1,2 \cdot 10^4 \cdot 2 \cdot \lg \frac{2 \cdot 3500^2}{0,4 \cdot 50} = 3,2Ом$$

Коефіцієнт використання протяжного заземлювача $\eta_n$ = 0,71. Дійсний опір протяжного заземлення становить:

$$R_n = \frac{R_{пр}}{\eta_n} = \frac{3,2}{0,71} = 4,5Ом \qquad (4.5)$$

Опір всього заземлюю чого пристрою становить:

$$R_u = \frac{R_c \cdot R_n}{R_c + R_n} = \frac{4.5 \cdot 7.3}{4.5 + 7.3} 2.78 < 3Ом \qquad (4.6)$$

Отже, число стержнів вибрано вірно.

### 4.4. Захист цивільного населення

Забезпечення захисту населення і території у разі загрози та виникнення надзвичайних ситуацій є одним з найважливіших завдань не лише підприємства, але й цілої держави. Актуальність проблеми забезпечення природо-техногенної безпеки населення і території зумовлена тенденціями зростання втрат людей і шкоди територіям, що спричиняються небезпечними природними явищами, промисловими аваріями і катастрофами.

**Інженерний захист** проводиться з метою виконання вимог ІТЗ із питань забудови міст, розміщення ПНО, будівлі будинків, інженерних споруд та інше.

**Медичний захист** проводиться для зменшення ступеня ураження людей, своєчасного надання допомоги постраждалим та їх лікування, забезпечення епідеміологічного благополуччя в районах надзвичайних ситуацій.

**Біологічний захист** включає своєчасне виявлення чинників біологічного зараження, їх характеру і масштабів, проведення комплексу адміністративно-господарських, режимно-обмежувальних і спеціальних протиепідемічних та медичних заходів.

**Радіаційний і хімічний захист** включає заходи щодо виявлення і оцінки радіаційної та хімічної обстановки, організацію і здійснення дозиметричного та хімічного контролю, розроблення типових режимів радіаційного захисту, забезпечення засобами індивідуального захисту, організацію і проведення спеціальної обробки.

# ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Встановлено, що в Україні на ринку доставки домінують «Нова пошта» та «Укрпошта», причому Новій пошті віддають перевагу завдяки швидкій доставці та розгалуженій мережі відділень. Нова Пошта пропонує такі зручні послуги, як миттєві сповіщення та розширене відстеження, але за вищою ціною. Укрпошта доступніша, але має менші терміни доставки та нижчу якість обслуговування. Кожна послуга задовольняє різні потреби клієнтів.

Дослідження особливостей використання API різних служб доставки показало, що за їх допомогою можливо створювати відправлення та керувати ними, відстежувати посилки та інтегрувати послуги логістики. API забезпечують ефективні рішення доставки для компаній. Проте розробка нових інформаційних систем управління даними у цій сфері дозволить значно покращити сервіс обробки цих даних.

C# — це універсальна мова, яка використовується для веб-розробки, програм Windows, ігор і хмарних служб. Вона пропонує надійну платформу .NET, статичний тип для безпечнішого коду та підтримку великої спільноти розробників. Архітектура N-Tier і архітектура REST покращують організацію додатків і взаємодію веб-служб. ASP.NET Core 6 забезпечує модульну розробку для чистої організації коду, а Microsoft SQL Server є кращим за його продуктивність і безпеку у веб-додатках, розроблених на C#.

Реалізація інформаційної системи включає дві бази даних MS SQL Server для реєстрації користувачів і обробки замовлень, використовуючи ASP.NET Core Identity для автентифікації. N-рівнева архітектура з рівнями DAL, BLL і PL забезпечує ефективне керування даними. RESTful API і принципи SOLID покращують функціональність системи. Система дозволяє різноманітні взаємодії з базою даних клієнтів, як-от перегляд продукту, додавання, видалення, створення замовлення та рейтинг.

Проведено структурний і функціональний аналіз виробничих процесів для розробки моделі поводження з травматичними ситуаціями в промислових

умовах. Підкреслюється важливість заходів безпеки під час експлуатації обладнання, запобігання пожежі, розрахунку штучного заземлення. Крім того, висвітлюється важливість захисту цивільного населення під час надзвичайних ситуацій за допомогою заходів інженерного, медичного, біологічного, радіаційного та хімічного захисту. Основна увага приділяється забезпеченню безпеки та мінімізації ризиків у різних промислових та надзвичайних ситуаціях.

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Поза конкуренцією: Нова пошта в три рази популярніше інших служб доставки (інфографіка). URL: https://rau.ua/novyni/novini-kompanij/rating-delivery/ (дата звернення 25.04.2024)

2. Служби доставки в Україні: короткий огляд. URL: https://ua.weblium.com/blog/sluzhbi-dostavki-v-ukrayini (дата звернення 25.04.2024)

3. Wikipedia: N-tire architecture URL: https://en.wikipedia.org/wiki/Multitier_architecture (дата звернення 25.04.2024)

4. LearnSQL: Microsoft SQL Server Pros and Cons URL: https://learnsql.com/blog/microsoft-sql-server-pros-and-cons/ (дата звернення 25.04.2024)

5. LearnMicrosoft: Introduction to Identity on ASP.NET Core URL: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-7.0&tabs=visual-studio (дата звернення 26.04.2024)

6. FastAPI URL: https://fastapi.tiangolo.com (дата звернення 27.05.2024)

7. Ormuco: Most Popular Databases in 2020 URL: https://ormuco.com/blog/most-popular-databases (дата звернення 27.04.2024)

8. Wikipedia: Database URL: https://en.wikipedia.org/wiki/Database (дата звернення 27.04.2024)

9. Wikipedia: API URL: https://en.wikipedia.org/wiki/API (дата звернення 27.04.2024)

10. LearnMicrosoft: ASP.NET Core web API documentation with Swagger / OpenAPI URL: https://learn.microsoft.com/uk-ua/aspnet/core/tutorials/web-api-help-pages-using-swagger?view=aspnetcore-7.0 (дата звернення 24.04.2024)

11. LearnMicrosoft: Handle requests with controllers in ASP.NET Core MVC URL: https://learn.microsoft.com/uk-ua/aspnet/core/mvc/controllers/actions?view=aspnetcore-7.0 (дата звернення 30.04.2024)

12. LearnMicrosoft: Get started with ASP.NET Core MVC URL:

https://learn.microsoft.com/uk-ua/aspnet/core/tutorials/first-mvc-app/start-mvc?view=aspnetcore-7.0&tabs=visual-studio (дата звернення 30.03.2024)

13. LearnMicrosoft: Quickstart: Install and use a NuGet package in Visual Studio URL: https://learn.microsoft.com/uk-ua/nuget/quickstart/install-and-use-a-package-in-visual-studio (дата звернення 30.04.2024)

14. LearnMicrosoft: Work with Language-Integrated Query (LINQ) URL: https://learn.microsoft.com/uk-ua/dotnet/csharp/tutorials/working-with-linq

15. Wikipedia: Model-view-controller URL: https://en.wikipedia.org/wiki/Model–view–controller (дата звернення 30.05.2023)

16. SCand: .NET vs Java: Unbiased Comparison URL: https://scand.com/company/blog/net-vs-java-comparison/ (дата звернення 30.04.2024)

17. LearnMicrosoft: Create a database URL: https://learn.microsoft.com/en- us/sql/relational-databases/databases/create-a-database?view=sql-server-ver16 (дата звернення 30.04.2024)

18. Codecademy: What is REST? URL: https://www.codecademy.com/article/what-is-rest (дата звернення 30.04.2024)

19. Fivetran: PostgreSQL vs. MySQL URL: https://www.fivetran.com/blog/postgresql-vs-mysql (дата звернення 30.04.2024)

20. Swagger: API Documentation URL: https://swagger.io/solutions/api-documentation/ (дата звернення 30.04.2024)

21. Огляд протоколу HTTP : [Електронний ресурс]. – Режим доступу: https:// developer. mozilla.org /ru/docs/ Web/HTTP/Overview (дата звернення 30.04.2024)

22. Пістун І. П., Тимочко В.О., Городецький І. М.. Березовецький А. П. Охорона праці (гігієна праці та виробнича санітарія): навч. посібн. / за ред. І.П. Пістуна. Ч. ІІ. Львів: Тріада плюс, 2011. 224 с.

23. Пістун І. П., Тимочко В.О., Городецький І. М.. Березовецький А. П. Охорона праці (гігієна праці та виробнича санітарія): навч. посібн. / за ред. І.П. Пістуна. Ч. ІІ. Львів: Тріада плюс, 2011. 224

# ДОДАТКИ
## Код створення контролерів

OrdersController:

```
[ApiController]
[Route("api/[controller]")]
public class OrdersController : ControllerBase
{
    private readonly IOrderService _orderService;

    public OrdersController(IOrderService orderService)
    {
        _orderService = orderService;
    }

    // GET: api/orders
    [HttpGet]
    public async Task<ActionResult<IEnumerable<Order>>> GetOrders()
    {
        var orders = await _orderService.GetAllOrders();
        return Ok(orders);
    }

    // GET: api/orders/{id}
    [HttpGet("{id}")]
    public async Task<ActionResult<Order>> GetOrder(int id)
    {
        var order = await _orderService.GetOrderById(id);
        if (order == null)
        {
            return NotFound();
        }
        return Ok(order);
    }

    // POST: api/orders
    [HttpPost]
    public async Task<ActionResult<Order>> CreateOrder(Order order)
    {
        await _orderService.CreateOrder(order);
        return CreatedAtAction(nameof(GetOrder), new { id = order.OrderId }, order);
    }

    // PUT: api/orders/{id}
    [HttpPut("{id}")]
    public async Task<IActionResult> UpdateOrder(int id, Order order)
    {
        if (id != order.OrderId)
        {
            return BadRequest();
        }
```

```
        try
        {
            await _orderService.UpdateOrder(order);
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!await OrderExists(id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }

        return NoContent();
    }

    // DELETE: api/orders/{id}
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteOrder(int id)
    {
        var order = await _orderService.GetOrderById(id);
        if (order == null)
        {
            return NotFound();
        }

        await _orderService.DeleteOrder(id);
        return NoContent();
    }

    private async Task<bool> OrderExists(int id)
    {
        var order = await _orderService.GetOrderById(id);
        return order != null;
    }
}
```

2. CustomersController:

```
[ApiController]
[Route("api/[controller]")]
public class CustomersController : ControllerBase
{
    private readonly ICustomerService _customerService;

    public CustomersController(ICustomerService customerService)
    {
        _customerService = customerService;
    }
```

```
// GET: api/customers
[HttpGet]
public async Task<ActionResult<IEnumerable<Customer>>> GetCustomers()
{
    var customers = await _customerService.GetAllCustomers();
    return Ok(customers);
}

// GET: api/customers/{id}
[HttpGet("{id}")]
public async Task<ActionResult<Customer>> GetCustomer(int id)
{
    var customer = await _customerService.GetCustomerById(id);
    if (customer == null)
    {
        return NotFound();
    }
    return Ok(customer);
}

// POST: api/customers
[HttpPost]
public async Task<ActionResult<Customer>> CreateCustomer(Customer customer)
{
    await _customerService.CreateCustomer(customer);
    return CreatedAtAction(nameof(GetCustomer), new { id = customer.CustomerId }, customer);
}

// PUT: api/customers/{id}
[HttpPut("{id}")]
public async Task<IActionResult> UpdateCustomer(int id, Customer customer)
{
    if (id != customer.CustomerId)
    {
        return BadRequest();
    }

    try
    {
        await _customerService.UpdateCustomer(customer);
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!await CustomerExists(id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
```

```
        }

        return NoContent();
    }

    // DELETE: api/customers/{id}
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteCustomer(int id)
    {
        var customer = await _customerService.GetCustomerById(id);
        if (customer == null)
        {
            return NotFound();
        }

        await _customerService.DeleteCustomer(id);
        return NoContent();
    }

    private async Task<bool> CustomerExists(int id)
    {
        var customer = await _customerService.GetCustomerById(id);
        return customer != null;
    }
}
```