КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій
Кафедра технологій управління

Спеціальність 122 – Комп'ютерні науки,
освітня програма «Інформаційна аналітика та впливи»

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему:

"Автоматизація аналізу та візуалізація даних у складських процесах"

Студента 2-го курсу групи ІАВ-21

Таборовського Андрія Анатолійовича
(прізвище, ім'я, по батькові)

(підпис студента)

Науковий керівник:

доктор технічних наук, доцент
(науковий ступінь, вчене звання)

Хлевна Юлія Леонідівна
(прізвище, ім'я, по батькові)

(дата) (підпис)

Попередній захист:

(Висновок: «До захисту в Екзаменаційній комісії»)

Завідувач кафедри технологій управління _____ Морозов В. В, _____

(підпис) (прізвище, ініціали) (дата)

Київ – 2021

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра технологій управління
Освітньо-кваліфікаційний рівень Магістр
Спеціальність 122 - Комп'ютерні науки
Освітня програма Інформаційна аналітика та впливи

ЗАТВЕРДЖУЮ
Завідувач кафедри
професор Морозов В.В.
_____
«____» _____20___року

З А В Д А Н Н Я
НА ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Студент: Таборовський Андрій Анатолійович

Група: ІАВ-21

1. Тема кваліфікаційної роботи

Автоматизація аналізу та візуалізація даних у складських процесах

Затверджена наказом по від «9» листопада 2020 р. № 4.

2. Строк подання студентом готової роботи – «7» травня 2021 р.

3. Цільова установка та вихідні дані до роботи.

Сервіс аналізу візуальної інформації для класифікації наліпок на коробках, що завозяться на автоматизований склад за допомогою технології комп'ютерного бачення а також обраного під час дослідження стеку технологій; система візуалізації інформації, технологічних процесів складу, реалізована у вигляді гібридного застосунку, який може бути встановлений на терміналах автоматизованого складу.

4. Зміст роботи.

Визначення впливу автоматизації процесу класифікації наліпок на коробках, що завозяться на автоматизований склад, а також впливу застосування технологій комп’ютерного бачення на ефективність процесу введення коробок.

Виокремлення підходів та методів до вирішення такого завдання та аналіз даних, які належать процесам, що потребують візуалізації для операторів автоматизованого складу.

Обґрунтування та вибір стеку технологій для програмної реалізації такої моделі з урахуванням технічних та архітектурних особливостей автоматизованого складу.

Пояснення доцільності створення гібридних застосунків для встановлення їх на автоматизованому складі, а також аналіз існуючих рішень та стеків технологій для створення таких застосунків.

Опис методити розробки гібридного застосунку для візуалізації інформації для операторів автоматизованого складу, архітектура якого буде дозволяти додати його до існуючих компонентів програмної системи складу.

5. Перелік графічного матеріалу (слайдів).

Робота налічує 11 таблиць та 20 рисунків, які позначають математичні формули для побудови моделей класифікації таких наліпок, структури процесу введення коробок на автоматизований склад, процеси класифікації зображення обраними методами, архітектуру програмної системи автоматихованого складу, огляд користувацького інтерфесу гібридного застосунку для візуалізації процесів на автоматизованому складу, що був розроблений у ході дослідження.

6. Календарний план виконання роботи:

№ п/п	Назва частин роботи	%	Виконання роботи	
			За планом	Фактично
1.	Вибір теми дипломної роботи	3	01.10.20	01.10.20
2.	Протокол кафедри ТУ про затвердження тем дипломних робіт та призначення наукових керівників	2	09.11.20	09.11.20
3.	Формування переліку нормативних матеріалів, літератури з проблематики дипломної роботи	10	08.01.21	07.01.21
4.	Складання розгорнутого плану кваліфікаційної роботи	5	18.01.21	18.01.21
5.	Ознайомлення наукового керівника з розгорнутим планом кваліфікаційної роботи. Внесення змін.	5	19.01.21 - 20.01.21	20.11.21
6.	Підготовка розділу 1 «Аналіз теоретико-методологічних основ використання інформаційної аналітики у складських процесах»	10	12.02.21	13.02.21
7.	Підготовка розділу 2 «Використання теорії нечіткої логіки для аналізу візуальної інформації у складських процесах»	14	08.03.21	08.03.21
8.	Підготовка розділу 3 «Технології реалізації моделі аналізу інформації у складських процесах»	14	01.04.21	01.04.21
9.	Підготовка розділу 4 «Методики розробки та застосування гібридних застосунків для візуалізації інформації у складських процесах»	13	20.04.21	20.04.21
10.	Оформлення кваліфікаційної роботи. Підготовка висновків і пропозицій	15	03.05.21	03.05.21
11.	Передача кваліфікаційної роботи науковому керівникові	2	04.05.21	04.05.21
12.	Передача кваліфікаційної роботи рецензенту для рецензування	2	07.05.21	07.05.21
13.	Попередній захист кваліфікаційної роботи	5	11.05.21	11.05.21

Дата видачі завдання «____» _____ 20__ р.

Керівник роботи:

доктор технічних наук, доцент
Хлевна Юлія Леонідівна
(посада, прізвище, ім’я, по батькові)

_____
(підпис)

Завдання прийняв до виконання студент групи ІАВ-21:

Таборовський Андрій Анатолійович

(прізвище, ім’я, по батькові)

_____
(підпис)

ЗМІСТ

АНОТАЦІЯ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра технологій управління
Спеціальність 122 - Комп'ютерні науки,
освітня програма "Інформаційна аналітика та впливи"

Дипломна робота магістра Таборовського Андрія Анатолійовича.

Тема роботи – «Автоматизація аналізу та візуалізація даних у складських процесах».

Мета дипломної роботи магістра – підвищити ефективність процесу введення коробок на автоматизований склад шляхом впровадження використання аналізу інформації та візуалізації даних у складських процесах.

Об'єкт дослідження – процес аналізу візуальної інформації для класифікації наліпок на коробках, що вводяться на автоматизований склад, вплив автоматизації цього процесу на його загальні обсяги та на ефективність.

Предмет дослідження – засоби, принципи, методології аналізу візуальної інформації та систем підтримки прийняття рішень для класифікації наліпок на коробках, що вводяться на автоматизований склад.

Наукова новизна роботи – процес аналізу візуальної інформації на складі стає повністю автоматизованим та готовим до вживлення в існуючу архітектуру програмної частини складу. Реалізується сервіс для аналізу візуальної інформації та візуалізації процесів на автоматизованому складі.

Дипломна робота складається зі вступу, основної частини, яка включає чотири розділи, висновків та списку використаних джерел. Всього налічує 84 сторінки основного тексту та перелік посилань з 51 джерела на 4 сторінках.

Ключові слова: проект, методика, ланцюг постачання, візуалізація інформації, автоматизація складів.

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API (прикладний програмний інтерфейс) – набір готових класів, процедур, функцій, структур і констант, що надаються застосунком (бібліотекою, сервісом) чи операційною системою для використання у зовнішніх програмних продуктах.

AR (Augmented Reality) – доповнена реальність.

ARM – це архітектура 32-бітових RISC-процесорів, розроблених компанією ARM Limited для мобільних пристроїв.

CLI (Command Line Interface) – інтерфейс командного рядка, консоль.

Cocoa – об’єктно-орієнтований прикладний програмний інтерфейс для ОС macOS.

DI (Dependency injection) – шаблон проектування програмного забезпечення (ПЗ), що передбачає надання зовнішньої залежності програмному компоненту. Працює у парі з IoC.

ECMAScript – стандарт мови програмування, затверджений міжнародною організацією ECMA згідно зі специфікацією ECMA-262.

FreeBSD – сучасна ОС на базі UNIX для серверів, десктопів і вбудованих комп’ютерних платформ.

GNU – вільна Unix-подібна операційна система.

GNUstep – вільна реалізація Cocoa.

HTML (HyperText Markup Language) – стандартна мова розмітки веб-сторінок у мережі Інтернет.

HTTP – протокол передачі даних, що використовується у інфокомунікаційних мережах.

IDE (Integrated Development Environment) – інтегроване середовище розробки.

Inbound – процес завезення товару на склад.

IoC Container (інверсія управління) – абстрактний принцип, набір

рекомендацій для написання слабозв’язаного коду. Його суть полягає в тому, що кожний компонент системи повинен бути якомога більше ізольованим від інших, не покладаючись у своїй роботі на деталі конкретної реалізації інших компонентів.

JavaScript (JS) – мова програмування високого рівня.

JSX – це об’єктно-орієнтована мова програмування, призначена для сучасних веб-браузерах.

Linux – це сімейство Unix-подібних операційних систем на базі ядра Linux, яка охоплює набір утиліт і програм проекту GNU та інші компоненти.

LLVM (Low Level Virtual Machine) – універсальна система аналізу, трансформації і оптимізації програм, що реалізує віртуальну машину з RISC-подібними інструкціями.

LPN – (License plate number) – унікальний ідентифікатор кожної палети з товаром.

NeXTSTEP – об’єктно-орієнтована, багатозадачна ОС, розроблена компанією NeXT Computer для власних комп’ютерів.

ОС – операційна система.

Outbound – процес вивезення товару зі складу.

PLC (Programmable logic controller) – апаратно-програмна система, яку можливо запрограмувати для подальшого використання в цілях автоматизації технологічних процесів.

Rabbit MQ - платформа, що реалізує систему обміну повідомленнями між компонентами програмної системи на основі стандарту AMQP (Advanced Message Queuing Protocol).

Redis – мережеве сховище даних типу ключ-значення з відкритим кодом.

Redux – JavaScript бібліотека з відкритим вихідним кодом для керування станом застосунку.

RISC (reduced instruction set computer) – архітектура процесора, в якій швидкість роботи збільшується за рахунок спрощення інструкцій, щоб їх декодування було простішим, а час виконання меншим.

Supply chain – проміжок часу, який проходить продукт від стадії його виробництва до кінцевого користувача.

SKU – (Stock keeping unit) – унікальний ідентифікатор товару. Містить в собі короткий опис категорії товару.

UNIX – операційна система, яку розробляли співробітники підрозділу Bell Labs корпорації AT&T.

WCF – (Windows Communication Foundation) — програмний фрумворк, що використовується для обміну даними між застосунками та входить у склад .NET Framework.

Xamarin - фреймворк для розробки мобільних застосунків (iOS/macOs/watchOS, Android, Windows Phone) з використанням мови C#.

Xcode – IDE, розроблений компанією Apple.

XNU – ядро операційної системи, розроблений компанією Apple. Є акронімом для X is Not Unix (X не є UNIX).

XPath, або XML Path Language — це мова запитів, для вибору вузлів з XML документів, або для обчислення величин (наприклад, рядкових, числових або булевих) на основі вмісту XML документа. XPath був розроблений World Wide Web Consortium (W3C).

ВСТУП

Із плином часу продуктові та виробничі компанії дедалі більше роблять акцент на важливості складського господарства у процесах своєї логістики. Головне завдання – це збереження та забезпечення транспортування запасів сировини і матеріалів, а також готових кінцевих продуктів. Це набуває важливої ролі у дистрибуції матеріальних цінностей, сировини, обладнання, nf інших виробів, продуктів харчування, напівфабрикатів, тощо.

Supply chain – шлях, який проходить товар з моменту його виробництва до споживання кінцевим користувачем. Найчастіше цей шлях складається з таких ланок: фабрика, склад, дистриб'ютори, кінцевий користувач. Дохід компанії за визначений термін залежить від обсягів товару, який пройде цей шлях. Не малу роль відіграє склад та його пропускна здатність – кількість товару, який склад може прийняти та віддати за короткий термін. Автоматизація та роботизація складу допоможе збільшити ці показники.

Автоматизація - напрям науково-технічного прогресу, який спрямовано на застосування саморегульованих технічних засобів, економіко-математичних методів і систем керування, що звільняють людину від участі у процесах отримання, перетворення, передачі і використання енергії, матеріалів чи інформації.

Для процесу Inbound автоматизованого складу (введення товарів) вхідною є інформація про коробки, які заїжджають на склад, про палети, в яких містяться ці коробки, про автомобілі, які привезли ці палети. Дані вводяться вручну, або скануються автоматично. Вихідним результатом є розміщення усіх коробок на полицях самим складом і збереження даних щодо місця призначення кожної коробки у сховищі (базі даних).

В процесі Inbound беруть участь не лише валідні для складу коробки. Вони можуть бути механічно пошкоджені, або ж мати стікери з позначенням типу товарів, правил зберігання чи транспортування, наявності алергенів тощо.

Ця інформація повинна бути оброблена складом на етапі отримання коробок. Рішення щодо прийняття цих коробок базується на візуальній інформації та повинне буде прийняте автоматично.

Швидкість та точність вирішення цього завдання і лежить в основі даного дослідження. Використання технологій комп'ютерного бачення та автоматизація цього процесу загалом допомагає підвищити пропускну здатність автоматизованого складу а також зменшити кількість ресурсів для його обслуговування. Як результат, обсяги Supply Chain процесу компанії чи холдингу збільшаться. Це і пояснює **актуальність** цього **дослідження**.

Методології для прийняття такого рішення та для візуалізації виконаних дії для оператора буде розглянуто у даній роботі.

Мета дослідження – підвищити ефективність процесу введення коробок на автоматизований склад шляхом впровадження використання аналізу інформації та візуалізації даних у складських процесах.

Було поставлено такі **завдання дослідження**:

1) покращити процес введення коробок на автоматизованому складі;
2) автоматизувати зчитування візуальних даних коробки, що заїжджає;
3) проаналізувати та обрати технологію для парсингу наліпок на коробках;
4) побудувати модель аналізу візуальної інформації за допомогою технологій комп'ютерного бачення;
5) розробити модуль аналізу візуальної інформації у форматі окремого сервісу;
6) створити систему сповіщень та звітності для візуалізації процесів на складі.

Об’єктом дослідження є процес аналізу візуальної інформації для класифікації наліпок на коробках, що вводяться на автоматизований склад, вплив автоматизації цього процесу на його загальні обсяги та на ефективність.

Предметом дослідження є засоби, принципи, методології аналізу візуальної інформації та систем підтримки прийняття рішень для класифікації наліпок на коробках, що вводяться на автоматизований склад.

Методами дослідження є системний аналіз щодо особливостей розробки системи аналізу візуальної інформації, синтез технологій аналізу візуальної інформації, а також класифікація та опис існуючих технологій для створення системи візуалізації прийнятих рішень для автоматизованого складу.

Науковою новизною є те, що процес аналізу візуальної інформації на складі стає повністю автоматизованим та готовим до вживлення в існуючу архітектуру програмної частини складу.

Практичне значення. Отримані результати можуть бути використані при створенні системи для аналізу даних та візуалізації інформації на автоматизованому складі. Запропоновано технології реалізації моделі аналізу інформації у складських процесах.

Особистий внесок здобувача. Усі наукові результати, які відображено у кваліфікаційній роботі, отримані автором самостійно. Результати співавторів сумісних публікацій до тексту кваліфікаційної роботи не включено. У надрукованих статтях, опублікованих у співавторстві, магістранту належить наступне: проведення аналітики доцільності та значущості застосування методів та засобів інтелектуального аналізу даних у сфері складського господарства.

Апробація результатів роботи. Автор виступав доповідачем на VI Information Technology and Interactions (Satellite): Conference Proceedings (м. Київ, 2020р.)

Публікації. Основні наукові положення, висновки і результати магістерської кваліфікаційної роботи знайшли відображення у 1 тезах доповіді на конференції.

Структура та обсяг роботи. Магістерська робота складається зі вступу, 4 розділів, висновків, списку використаних джерел з 51 найменування та додатків. Загальний обсяг становить 109 сторінок, із них 84 сторінок основного тексту, який містить 20 рисунків.

РОЗДІЛ 1
АНАЛІЗ ТЕОРЕТИКО-МЕТОДОЛОГІЧНИХ ОСНОВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНОЇ АНАЛІТИКИ У СКЛАДСЬКИХ ПРОЦЕСАХ

Із плином часу продуктові та виробничі компанії дедалі більше роблять акцент на важливості складського господарства у процесах своєї логістики. Головне завдання – це збереження та забезпечення транспортування запасів сировини і матеріалів, а також готових кінцевих продуктів. Це набуває важливої ролі у дистрибуції матеріальних цінностей, сировини, обладнання, nf інших виробів, продуктів харчування, напівфабрикатів, тощо.

Склад – приміщення або комплекс приміщень, що є однією з ланок процесу постачання товарів і призначений для зберігання ресурсів чи будь-яких матеріальних цінностей. Свого роду це медіатор резервів ресурсів, необхідних для забезпечення та підтримки рівня об’єму поставок і попиту, а також підтримки балансу потужностей потоків товарів у процесі просування шляху від виробництва до кінцевих споживачів або контроль процесів технологічного виробництва.

Складське господарство – це сполучна ланка між виробничими підрозділами, цехами, що займаються випуском продукції, потужностями матеріально і технічного забезпечення, внутрішніми підрозділами потужностей, а також місцями збуту. Його діяльність впливає на безперебійну і ефективну роботу основного виробництва, на ритмічний випуск і відгрузку товарної продукції. [2]

1.1 Функції складського господарства

Складське господарство виконує такі функції:

- Прийом та організація належних умов для зберігання товарів, матеріальних цінностей, ресурсів, а також ї розвантаження, переміщення, перетарювання; перевірка упаковки згідно з

визначеними стандартами, а також оформлення документів для всіх згаданих процесів;

- забезпечення ресурсів та умов для випуску матеріальних засобів виробництва і їх транспортування між різними точками (етапами) виробництва;
- підготовка приміщень і площі, переміщення вантажів на складі та його частинах з метою раціонального використання території та площі складів;
- прийом готової продукції по визначеній кількості, типу і сортах від одного або різних виробників та постачальників, оформлення документів для прийому, зберігання та виведення такої продукції; забезпечення умов для її збереження, відповідальність за належний стан прдукції;
- відпуск продукції складом за отриманим замовленням, яке може базуватись на типові товару, на даті кінцевого терміну придатності, іншими спорідненими ознаками тощо з оформленням належної документації;
- реалізація заходів та розробка методик щодо покращення якості процесів складського господарства: введення товарів, виведення товарів, їх переміщення на складі; збільшення рівню автоматизації даних процесів;

Функції, що здійснюються на складах, можна узагальнити наступними головними процесами: прийом ресурсів, їх розміщення, забезпечення умов для належного зберігання, підготовка до видачі з урахуванням умов замовлення, що запрошується; забезпечення документації для усіх цих процесів. [2]

1.2 Зберігання товарів на складі

Продукти та ресурси, що вводяться на склад, проходять перевірку на відповідність та валідацію. Цей процес полягає у перевірці відповідності фактичної якості та стані матеріалів до того, який вказаний у документах, технічних особливостях товарів.

Попередня перевірка вантажів, що надійшли на склад, здійснюється представником підприємства на спеціальній валідаційній станції, яка визначається за правилами споживачів. Тут перевіряється кількість фактичних місць, що прибули, цілісність, правильність упакування, наявність необхідного маркування порівняння фактичної ваги з допустимою. Якщо знайдено розбіжність між фактичною інформацією і тим, що вказано в супровідних документах, то на станції складається комерційний акт про повернення товарів та спецальна форма для пред'явлення претензій винуватцеві неточностей. Це може бути або виробник, або транспортер. [4]

Навіть якщо кількість та якість матеріалу, що надійшов, по зовнішньому вигляді не викликає сумнівів, його вага та маркування на станції прибуття все одно перевіряється. З моменту отримання товару складом відповідальність за його стан несе виключно сам склад.

Разом із кількісною перевіркою на складах має проводитись якісне приймання. Здійснюється вона, як правило, органами технічного контролю із залученням лабораторій (за необхідністю). Якісною перевіркою встановлюється відповідність стандартам та технічним умовам отриманих товарів. За умов невідповідності товарів ти матеріалів стандартам або технічним умовам викликається представництво постачальника й складається акт про непридатність матеріалу.

В окремих випадках акт про непридатність складається самою комісією: наприклад, якщо представник постачальника не може прибути, або ж відсоток

невалідних лежить в межах допустимої похибки. При кладанні такого акту обов ’ яково залучається представник незацікавленої організації. Акт направляється постачальникові разом із запитом, як діяти із забракованим матеріалом. До вказівки власника торав перебуває в споживача на окремому зберіганні в спецально відведеному місці. Якщо це зберігання завдає збитків складові, виробник також несе відповідальність.

Іноді якісна перевірка матеріалів і продуктів не проводиться. Для цього необхідно, щоб виробник мав список документів, що підтверджують валідність товарів та знімають відповідальність із складу за їх некоректне зберігання. Склад у такому випадку може нести покарання тільки за самостійне завдавання шкоди продуктам чи сировині.

Товари та матеріали, що прийняті на склад, розміщаються з дотриманням вимог зберігання та обліку. Для цього кожний матеріал повинен бути розташованим на складі з рахуванням того, щоб кількість та якість матеріалів збереження була забезпечена. Матеріали спільного призначення та ідентичного типу розміщаються поруч, важкі та великі товари повинні розташовуватись ближче до місця виведення. [4]

Є три способи зберігання та розташування ресурсів чи матеріалів на складах підприємства:

- сортове розміщення - передбачає закріплення постійного місця за кожним видом товарів;

- партіонне розміження – передбачає закріплення місця за кожною партією матеріалів, що надійшла на підприємство;

- комплектне розміщення – передбачає закріплення місця за кожною групою товарів, що випускатиметься.

Більшість промислових підприємств із матеріальними складами містять спеціальні території та потужності для підготовки матеріалів до виробництва.

Це надає можливості більш ощадливо використовувати матеріали, застосовуючи методи комбінованого розкрою, використовуючи відходи для переробки чи виробництва дрібніших деталей і т.д.

Комплектування товарів та ресурсів є одним з видів підготовки матеріалів до виробництва перед відпусткою їхнім виробником. Відпустка матеріалу цехам виконується з урахуванням встановлених обмежень кожного цеху. Залежно від типу виробництва й роду матеріалів процес відпустки матеріалів складається з різних етапів.

Більшість матеріалів масового й крупносерійного виробництва відпускаються за планкартами. Планкарта - це документ, що визначаєтсья та заповнюється відділом постачання чи планово-виробничим відділом, у якому вказуються строки та партії подачі, місячний ліміт по кожному виді матеріалу для кожного окремого цеху. Відповідно до таких планкарт склад планує свої процеси діялності таким чином, щоб своїми транспортними засобами забезпечити кожному цеху подачу партії ресурсів у встановлений термін. Відпустка матеріалів оформляється приймально-здавальними накладними. [5]

На підприємствах одиничного чи серійного виробництва основні й допоміжні ресурси, а також продукти в масштабному й крупносерійном виробництві випускаються за вимогами відповідно до згаданих раніше лімітних карт і відомостях. Відпустка оформляється накладними або розписками, які належать одержувачу.

Облік переміщень матеріальних ресурсів ведеться за допомогою картотеки як на складах підприємства, так і в бухгалтерії постачаника. Картка кожного товару містить номенклатурний номер матеріалу, його назву, марку, сорт, одиницю виміру і ціну, особливості транспортуваннння та зберігання, а також фіксуються всі надходження та видачі товару. За картотекою розраховуються залишки матеріалів, які порівнюються з нормами запасу зберігання й лімітами.

Для забезпечення та підтримки ефективної роботи підприємства важливо організувати оперативне регулювання системи керування запасами. З цією метою встановлюється контроль над станом запасів на складі. Якщо кількість запасів, що видається не співпадає із заплановою кількістю, то це служить сигналом того, що роцес виробництва може бути порушений. Про це повідомляються органи матеріало-технічного забезпечення. [7]

Таким чином, склади виконують не лише функції зберігання й підготовки матеріалів до видачі у подальше виробництво, але й допомагають слідкувати за об’ємами цього процесу, порівнювати їх із очікуваннями.

1.3 Лацюг постачання

Supply Chain (ланцюг постачання) - це сукупність процесів та інформації із збереженням послідовності, які визначають шлях виробництав, який проходить продукт чи послуга: від етапу збору сировини до дистрибуції кінцевому користувачу. Логістика будь-якої продуктової чи виробничої компанії базується саме на процесі ланцюга постачання. Існує зовнішній та внутрішній ланцюг постачання. Рівень товаророзподілу – ланка посередників, які беруть участь у переміщенні товару між потужностями і передають права власності на нього черговому елементу ланцюжка у напрямку кінцевого споживача.

У 1982 році Кейт Олівер, консультант Booz Allen Hamilton, в інтерв'ю Financial Times вніс термін "управління ланцюгами поставок" в суспільне надбання.

У середині 1990-х років, більш ніж через десять років, термін "управління ланцюгами поставок" набув значення, коли на цю тему вийшов шквал статей і книг. Спочатку визначалися ланцюги поставок, які охоплювали будь-які види діяльності, пов'язані з потоком, обробкою і перетворенням товарів від сировини до кінцевого споживача, а також пов'язані з ними інформаційні потоки та процеси.

Управління ланцюгми поставок далі було визначено як інтеграцію діяльності ланцюга постачання за рахунок поліпшення відносин ланцюжка постачання для досягнення конкурентних переваг.

Наприкінці 1990-х років значення управління ланцюгами поставок зросло, і керівники операцій почали використовувати його у своїх титулах із зростаючою регулярністю. Інші загальноприйняті визначення управління ланцюгами постачання включають:

- Управління потоками доданої вартості матеріалів, кінцевих товарів та супутньої інформації між постачальниками, компаніями, торговими посередниками та кінцевими споживачами.
- Систематичне, стратегічне узгодження традиційних бізнес-функцій і тактик у всіх бізнес-функціях в межах конкретної компанії та між підприємствами в межах ланцюга постачання з метою поліпшення довгострокової діяльності окремих компаній та ланцюга поставок в цілому.
- Інтеграція ключових бізнес-процесів по ланцюжку поставок з метою створення цінності для клієнтів і зацікавлених сторін. [8]

За даними Ради фахівців з управління ланцюгами поставок, управління ланцюгами постачання включає планування та управління всіма видами діяльності, пов'язаними з постачанням, закупівлями, переробкою та управлінням логістикою. Вона також включає координацію та співпрацю з партнерами по каналах, які можуть бути постачальниками, посередниками, постачальниками послуг третіх сторін або клієнтами. Управління ланцюгами постачання об'єднує управління попитом та пропозицією всередині компанії та між ними. Віднедавна мережа компаній, які співпрацюють з метою надання пропозицій на продукцію та послуги, називається розширеним підприємством. [9]

Ланцюжок постачання, на відміну від управління ланцюгами поставок, являє собою набір організацій, безпосередньо пов'язаних одним або кількома потоками продуктів, послуг, фінансів або інформації від джерела до клієнта.

Програмне забезпечення для управління ланцюгами постачання включає в себе інструменти або модулі, які використовуються для виконання операцій у ланцюжку поставок, управління відносинами з постачальниками та управління пов'язаними з ними бізнес-процесами.

У багатьох випадках ланцюг постачання включає в себе збір товарів після споживчого використання для переробки. Включення сторонньої логістики або інших агентств зі збору коштів як частину процесу повторного патентування є способом продемонструвати нову стратегію завершення гри.

Менеджмент ланцюга постачання – дизайн, планування, виконання, контроль, моніторинг процесів для побудови конкурентноспроможної інфраструктури, створення масштабованої логістики, синхронізація постачання з попитом та вимірювання продуктивності вцілому. Це крос-функціональний підхід, що включає менеджмент руху сировини між організаціями, визначення процесів внутрішньої переробки сировини у завершені продукти, рух цих продуктів від організації до кінцевого користувача. [10]

Оскільки організації прагнуть зосередитись на ключових компетенціях для того, щоб бути гнучкими у плані збуту, вони зменшують власність їх каналів сировини та дистрибуції. Тоді ці функції передаються в аутсорсинг іншим організаціям, які можуть виконувати діяльність краще або ефективніше.

Ефект полягає у збільшенні кількості організацій, що займаються задоволенням попиту клієнтів, при одночасному скороченні управлінського контролю щоденних логістичних операцій. Менший контроль і більша кількість партнерів у ланцюжку постачання призводять до створення концепції управління ланцюгами поставок.

Метою управління ланцюгами постачання є поліпшення довіри та співпраці між партнерами в ланцюгу постачання, що сприяє підвищенню видимості запасів і швидкості руху запасів.

Складові менеджменту ланцюга постачання: індустріальна інженерія, системна інженерія, операційний менеджмент, логістика, звітність, маркетинг. Канали маркетингу відіграють важливу роль у менеджменті ланцюга постачання. Головне завдання – збереження стійкості та балансування ризиків при русі товарів між ланками ланцюга постачання.

Організації все частіше виявляють, що вони повинні покладатися на ефективні ланцюги постачання або мережі, щоб конкурувати на світовому ринку та мережевій економіці. У новій управлінській парадигмі Пітера Драккера (1998) ця концепція ділових відносин виходить за межі традиційних кордонів підприємства і прагне організувати цілі бізнес-процеси по всьому ланцюжку створення цінностей декількох компаній. [10]

В останні десятиліття глобалізація, аутсорсинг та інформаційні технології дозволили багатьом організаціям, таким як Dell та Hewlett Packard, успішно керуватися мережами спільного постачання, в яких кожен спеціалізований діловий партнер зосереджується лише на декількох ключових стратегічних заходах. Ця міжорганізаційна мережа постачання може бути визнана як нова форма організації.

Проте, з ускладненою взаємодією між гравцями, мережна структура не відповідає ні "ринкові", ні "ієрархічні" категорії. Незрозуміло, яким може бути вплив різних структур постачання-мережі на фірми, і мало відомо про умови координації та компроміси, які можуть існувати між гравцями. З точки зору систем, складну мережеву структуру можна розкласти на окремі фірми-компоненти. Традиційно компанії в мережах постачання концентруються на входах і виходах процесів, не турбуючись про внутрішнє управління роботою

інших окремих гравців. Тому, як відомо, вибір структури внутрішнього контролю управління впливає на продуктивність місцевої фірми.

У 21-му столітті зміни в бізнес-середовищі сприяли розвитку мереж постачального ланцюга. По-перше, як результат глобалізації та розповсюдження багатонаціональних компаній, спільних підприємств, стратегічних альянсів і ділових партнерств, були визначені значні фактори успіху, що доповнюють раніше "just-in-time", бережливе виробництво і гнучку виробничу практику. По-друге, технологічні зміни, зокрема різке падіння комунікаційних витрат (важлива складова трансакційних витрат), призвели до змін в координації між членами мережі постачальників. [11]

Багато дослідників визнали структури мереж постачання як нову організаційну форму, використовуючи такі терміни, як "Keiretsu", "Розширене підприємство", "Віртуальна корпорація", "Глобальна виробнича мережа" і "Система наступного покоління". Загалом, таку структуру можна визначити як "групу напівнезалежних організацій, кожен зі своїми можливостями, які співпрацюють, щоб обслуговувати один або більше ринків для досягнення певної бізнес-цілі, характерної для цієї співпраці".

Управління ланцюгами постачання також важливо для організаційного навчання. Фірми з географічно більш широкими ланцюжками поставок, що з'єднують різноманітні торгові кліки, як правило, стають більш інноваційними та продуктивними.

Система управління безпекою для ланцюгів постачання описана в ISO / IEC 28000 та ISO / IEC 28001 та відповідних стандартах, опублікованих спільно ISO та IEC. Управління ланцюгами постачання в значній мірі залучає сфери управління операціями, логістики, закупівель та інформаційних технологій, а також прагне до комплексного підходу.

1.4 Методи комп’ютерного бачення – використання нечіткої логіки

Комп’ютерний зір — комплексна та широка галузь, що включає в себе багато завдань різного спрямування, серед яких сегментація, класифікація, фільтрація, реконструкція сцени, оцінка положення об’єкта, виявлення об’єктів, відеоспостереження та багато інших.

Комп’ютерне бачення є важливою складовою розвитку штучного інтелекту та інтелектуальних інформаційних технологій. Комп’ютерне бачення використовується в десятках галузей, наприклад, при побудові «розумних» магазинів, ідентифікації клієнтів за допомогою біологічних характеристик, автоматизації сільськогосподарських процесів з використанням дронів, автоматичній інспекції на виробництвах, відеоспостереженні, покращенні якості фото- та відеоданих (фільтрація), автоматичній доставці посилок безпілотними літальними апаратами. [12]

Коло застосувань цієї технології розширюється, адже потреба у системах штучного інтелекту зростає, а зір — це один з найбільш інформативних сенсорів, який може використовуватись в таких системах. І хоча значного прогресу в цій галузі вже досягнуто, залишається багато нерозв’язаних задач. Існуючим алгоритмам бракує загальності, а збільшення швидкодії зазвичай викликає зменшення точності. Тому актуальними напрямками є: покращення швидкодії існуючих алгоритмів; застосування алгоритмів розпізнавання на вбудованих системах, які мають обмежені ресурси пам’яті і обчислювальні потужності, наприклад, безпілотні літальні апарати, роботизовані та супутникові системи; розв’язання задач в режимі реального часу в умовах обмежених ресурсів; покращення точності; зменшення затрат на навчання систем, що базуються на нейронних мережах; розширення кола об’єктів розпізнавання; розпізнавання зображень низького розширення та якості.

Вважається, що значного покращення в задачах комп’ютерного бачення можна досягнути, якщо збільшити кількість інформації, яку можна обробити за адекватний період часу. Але хоча людина не вміє обробляти величезні потоки даних в реальному часі, вона виконує задачу розпізнавання дуже і дуже успішно. Вміння підбирати потрібний рівень деталізації — ось що дозволяє людині розпізнавати об’єкти, занадто загальний опис може призвести до пропуску важливих деталей та відвести увагу від основних характеристик.

Теорія нечіткої логіки дозволяє варіювати цим рівнем узагальнення змінюючи кількість лінгвістичних змінних в системі та варіюючи вигляд функцій належності нечітких множин. Системи комп’ютерного зору повинні вміти представляти невизначеність та передбачати ефекти невизначеності для правильної інтерпретації отриманих результатів.

Невизначеність часто розглядається як результат деякого випадкового процесу, проте в комп’ютерному баченні невизначеність може виникати і з інших причин, серед яких: проекція зображень у простір меншої розмірності, зміна освітлення, дискретизація просторових чи часових (у випадку відео) координат, невідома якість зображення, неточні обчислення, перетин класів розпізнаваних об’єктів, тобто неможливість чітко сформулювати ознаки і визначення об’єктів. [14]

Теорія нечіткої логіки ідеально підходить для розв’язання проблем невизначеності такого роду. Нечітка логіка дозволяє легко переносити накопичений досвід в системи комп’ютерного бачення, використовуючи прості та зрозумілі правила.

Людина, здійснюючи класифікацію зображень, не працює з суто числовими характеристиками об’єктів, які зображені, натомість ми класифікуємо об’єкт за базою правил, які отримані з досвідом і можемо легко сформулювати. Це є великою перевагою таких систем, оскільки потенціал для їх покращення

практично невичерпний і завжди можна сформулювати правила більш жорстко, чи доповнити систему новими, уточнюючими, правилами.

Базовим поняттям нечіткої логіки є нечітка множина. Нечітка множина — це множина, елементи якої належать їй в певній мірі, на відміну від традиційних множин, де елементи або належать множині або ні. Нечітка множина — це пара: (A, f), де A — множина, а $f: A \rightarrow [0,1]$ — функція належності або характеристична функція множини.

Вважаючи, що елемент належить нечіткій множині в певній мірі, судження в нечіткій логіці є правдивими теж в певній мірі, на відміну від традиційної логіки, де висловлювання може бути або вірним, або хибним.

Основними кроками проектування систем на базі нечіткої логіки є:

1) фаззіфікація — підбір лінгвістичних змінних, визначення потрібного рівня деталізації;

2) підбір функцій належності для кожної нечіткої множини кожної лінгвістичної змінної;

3) вибір бази нечітких правил таким чином, щоб мінімізувати їх кількість та максимізувати точність;

4) дефаззіфікація — приведення виходу системи до чіткого вигляду, лінгвістичні змінні, які використовуються як вихід системи часто неможливо прямо застосувати для розв’язання задачі, зазвичай їх потрібно перевести в кількісний вигляд (цей крок часто називають редукцією, найпоширенішим методом є редукція з використанням центру множини). [15]

Як відомо, сегментація зображень — процес виділення значущих сегментів на зображенні з метою покращення подальшого виявлення об’єктів, опису сцени і розуміння контексту.

Процес знаходить області зі спільними ознаками (колір, перехід відтінків, формат текстури). Саме визначення не є чітким: невідомо як визначити схожість

між пікселями чи їх областями. Сегментація зображень є одним із найважливіших завдань комп'ютерного бачення, це один з перших кроків класифікації.

Якщо зображення буде неправильно поділене на області, то і їх подальша класифікація буде невірною. Результати сегментації можуть розглядатися як нечіткі множини. Кожній області присвоюється нечітка множина і визначається рівень з яким кожен піксель належить кожній множині.

Після цього для отримання остаточного результату за традиційними техніками порогових значень, кластеризації, сегментації з учителем та сегментації на базі правил застосовуються техніки теорії нечітких множин.

Розглянуто проблему сегментації дистанційних зображень таких як знімки з висоти пташиного польоту. Вони використовують нечіткі множини двох типів.

Гаусівські моделі першого типу використовуються для моделювання невизначеності, що присутня на зображенні. [16]

В подальшому нечітка модель другого інтервального типу будується шляхом «розмивання» математичного сподівання та дисперсії, в результаті чого отримується верхня та нижня функції належності.

Запропонована модель другого типу допомагає підсилити вираження невизначеності на зображенні та одночасно зменшити невизначеність у системі прийняття рішення. Далі нечітка функція належності з її верхньою та нижньою функціями належності використовується як вхід нейронної мережі, що є системою прийняття рішень. По суті нечіткі моделі використовуються для покращення роботи фінальної моделі — нейромережі.

1.5 Візуалізація даних у складських процесах

Візуалізація - один з найефективніших інструментів презентації даних. Вона дозволяє наочно відстежити основні тенденції, залежності, розподіл і відхилення

досліджуваних ознак, знайти кореляцію показників, оцінити зміни, допомогти прийняти правильні бізнес-рішення.

Важливо не просто показати інформацію, а зробити це максимально зрозуміло і прозоро для користувача, виділивши ключові показники.

Етапи аналізу даних для подальшої візуалізації.

Формулювання мети. Кожне дослідження має відповідати на ряд поставлених питань - не потрібно плодити дослідження для досліджень.

Збір даних. На цьому етапі аналітик або працює з уже зібраними даними, або бере участь в процесі постановки завдання на збір даних (фактично вирішує, яка інформація йому необхідна і в якому вигляді).

У першому випадку особливу увагу варто приділити правильній інтерпретації даних, які записані в базу, і часто змиритися з існуючим форматом даних, дизайном таблиць тощо. У другому випадку аналітик зустрічається з проблемою побудови грамотного сценарію збору даних - він може особливо перестаратися в плануванні A / B - тестів, логуванні подій тощо. Тут важлива комунікація з тими, хто може допомогти в розумінні процесів та оцінки масштабів запланованих записів. [18]

Підготовка данних. «Сміття на вході - сміття на виході» - правило, про яке завжди потрібно пам'ятати. Структурування, усунення помилок, зміна форматів вмісту, розбір аномальних результатів, очищення від викидів, усунення дублікатів, інтеграції даних з різних джерел - одні з найважливіших пунктів в аналізі даних.

Іноді потрібно розширення метрик, наприклад додавання обчислювальної інформації (приріст, ранг, номер). Іноді слід скоротити кількість ознак (змінних) або перейти до допоміжних змінних, які приймають одне з двох значень: true (1) / false (0).

На цьому етапі сирі дані перетворюються в корисну вхідну інформацію для моделювання та аналізу.

Дослідження даних. Для правильної інтерпретації багатовимірних даних необхідно подивитися на них в розрізі як конкретної ознаки, так і групи ознак. Також мають бути подані ключові показники в динаміці з планами і фактичними результатами. Саме на цьому етапі підбирається формат майбутньої візуалізації.

Візуалізація і побудова висновків. Кожне дослідження має закінчуватися результатами і висновками. Навіть якщо вони негативні, їх варто проговорити і обговорити. При цьому правильна постановка задачі, методика проведення збору даних, правильна інтерпретація результатів, виявлені помилки і багато іншого повинні послужити базою для подальших досліджень. [19]

Час, що витрачається аналітиком на кожну фазу, залежить від багатьох змінних: починаючи від рівня професійної підготовки і рівня знання даних, закінчуючи переліком використовуваних інструментів і технічних характеристик ПК.

Необхідно також розуміти, що процес аналізу даних має ітераційний характер і може бути представлений циклом.

1.6 Постановка завдання дослідження

Завданням даного дослідження є покращення процесу введення коробок на автоматизованому складі, а саме автоматизація зчитування візуальних даних коробки, що заїжджає та візуалізація прийнятих рішень і результату роботи для оператора складу.

Необхідно проаналізувати та обрати технологію для парсингу наліпок на коробках, визначити систему підтримки прийняття рішень, яка може бути запроваджена на складі. Після того потрібно визначитись із стеком технологій, який буде використовуватись для створення програмного забезпечення для візуалізації процесів на інтерфейсі оператора.

Потрібно побудувати модель аналізу візуальної інформації за допомогою технологій комп’ютерного бачення та розробити модуль, який може бути використаний, як окремий сервіс, з яким можлива комунікація для існуючих PLC компонентів автоматизованого складу. [19]

Також повинна бути створена система сповіщень та звітності для візуалізії процесів. Така система отримуватиме далі з програмних компонентів та відображатиме їх на консолях операторів складу.

Створені компоненти повинні відкриті для розширення для подальшого використання на підприємстві.

Висновки до розділу 1

Було проаналізовано теоретико-методологічні основи використання інформаційної аналітики у складських процесах. Було описано складову Supply Chain, а саме процес введення коробок на склад, та визнечено роль автоматизації прийняття рішень на швидкість на якість цього процесу.

Було розглянуто, які завдання вирішує візуалізація інформації складських процесів, впроваджена для операторів складу.

А також було поставлено завдання для покращення підвищення ефективності процесу введення коробок на склад шляхом впровадження компонентів аналізу даних в архітектуру програмної складової складу.

РОЗДІЛ 2
ВИКОРИСТАННЯ ТЕОРІЇ НЕЧІТКОЇ ЛОГІКИ ДЛЯ КЛАСИФІКАЦІЇ НАЛІПОК НА КОРОБКАХ У СКЛАДСЬКИХ ПРОЦЕСАХ

2.1 Опис проблеми

Коробки, що завантажуються на склад відбувається валідація кожної з них. Процес зважування та вимірювання не потребує додаткового аналізу, дані, отримані за допомогою сканерів та датчиків, прозоро відправляються у систему, яка, у свою чергу, приймає рішення щодо того, чи приймати дану коробку.

Однак, кожна коробка може мати додаткові наліпки, які доповнюють, або змінюють існуючу інформацію про товар всередині (SKU). Для прикладу, це може бути макрування про те, як конкретну коробку потрібно транспортувати, як її потрібно зберігати, або ж маркування про потенційно небезпечні речовини всередині, алергени тощо.

Приклади таких наліпок наведено на рисунках 2.1 – 2.2.

Рисунок 2.1 – Наліпки транспортування та зберігання

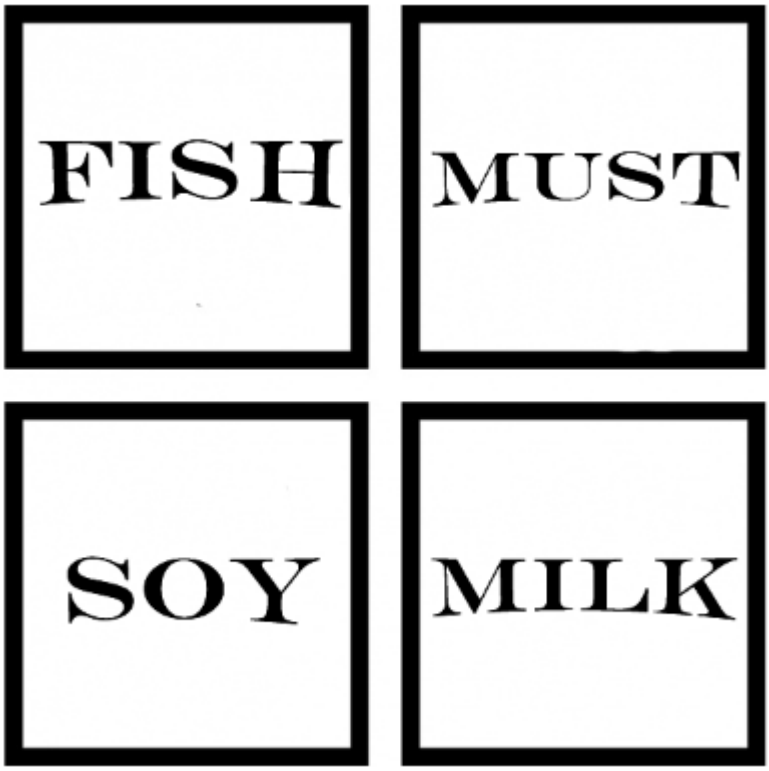

Рисунок 2.2 – Наліпки алергенів

Сканування таких наліпок відбувається сканерами із власним PLC, який у свою чергу відправляє отримані результати у вигляті колекції пікселів у систему. Завдання системи – класифікувати та точно визначити, яке саме маркування було знайдено. Це вплине на прийняття рішення щодо подальших дії для цієї коробки: спосіб її транспортування, місце її зберігання тощо.

Це завдання можна було б виконати за допомогою звичайного шаблонування. Наліпку було б ідентифіковано після того, як її накладання на один із передвизначених шаблонів показало б точність мінімум 90%.

На практиці це не можливо через те, що існує безліч постачальників товарів, у яких маркування може відрізнятись. Хоч загальні принципи та символи є уніфікованими, проте певний зразок однієї наліпки першого постачальника може бути схожим на зразок зовсім іншої за змістом наліпки іншого постачальника.

В такому випадку система може прийняти не вірне рішення. Для вирішення такого завдання можна залучити оператора складу для підтвердження рішень системи, або ж створити сервіс для класифікування та розподільного шаблонування зображень з метою підвищення рівня точності ідентифікації наліпок.

2.2 Класифікація візуальної інформації

Аналіз візуальної інформації методом нечіткої логіки на вхід отримує множину із відомих шаблонів даних. Результатом аналізу є таблиця належності до кожної вхідної множити з такими параметрами:

- Вага належності до множини за методом FMP (fuzzy modus ponens); проміжок значень Y_1 := [0; 5000]
- Вага належності до множити за методом FMT (fuzzy modus tollens); проміжок значень Y_2 :=[0; 500]
- Параметр нечіткості; множина значень Y_3 := [100; 200]

Спробуємо визначити приналежність наліпки із алергеном «молоко» до групи наліпок. Групою вважається множина одного і того ж типу наліпок від усіх можливих постачальників. Їх кількість є скінченною.

Відомі групи наліпок для даної задачі класифікації:

- «вертикаль»
- «скло»
- «пластик класу 1»
- «пластик класу 2»
- «молоко»
- «риба»
- «соя»
- «гірчиця»

Дані результату аналізу методом нечіткої логіки наведені у таблиці 2.1.

Таблиця 2.1 – Аналіз методом нечіткої логіки

Група	Кількість спостережень	Вага FMT	Вага FMP	Параметр нечіткості
Молоко	75	450	3000	150
Соя	90	350	4000	180
Гірчиця	86	300	3000	140
Риба	89	300	4500	145
Вертикаль	17	400	2500	165
Скло	20	300	4000	155
Пластик 1	35	350	3000	170
Пластик 2	39	400	2500	165

Аналіз методом головних компонент дає можливість виділити головні для аналізу компоненти серед усіх вхідних компонент. Скористаємось сервісом STATGRAPHICS та проаналізуємо вихідне зведення методу головних компонент. Результат представлений у таблиці 2.2.

Таблиця 2.2 – Результат зведення головних компонент

Номер компоненту	Власний вектор	Відсоток дисперсії	Накопичений відсоток дисперсії
1	3.4434	35.456	35.456
2	3.2032	30.875	66.331
3	2.765	26.139	92.47
4	0.6463	7.53	100

З отриманої таблиці зробимо висновок, що аналізу піддаються такі змінні: вага належності FMP, вага належності FMT та параметр нечіткості, в той час, як кількість спостережень в групі має мінімальний вплив на дисперсію вихідних даних.

Наведені результати говорять про те, що перші дві головні компоненти описують 66.331 % дисперсії вихідних даних. Третя головна компонента додає ще 26.139 % дисперсії, так що в сумі виходить 92.459% дисперсії.

Отримаємо ваги ознак у головних компонентах (таблиця 2.3)

Таблиця 2.3 – Ваги ознак головних компонент

	Компонент 1	Компонент 2	Компонент 3
Power	**0.30452**	-0.12953	**0.49153**
Lifetime	-0.1905	**0.49622**	-0.12051
Chargetime	**0.35030**	**-0.39422**	0.23044
Price	**-0.36939**	0.16033	**0.3935**

З отриманих значень випливає, що в першому головному компоненті спостерігається найбільша прямопропорційна залежність ваги FMP та ваги FMT. У другому головному компоненті спостерігається прямопропорційна залежність від ваги FMP і параметру нечіткості. У третьому головному компоненті набільша прямопропорційна залежність ваги FMP і параметру нечіткості.

Розглянемо діаграму розсіювання на площині виділених двох головних компонент (рис. 2.3)

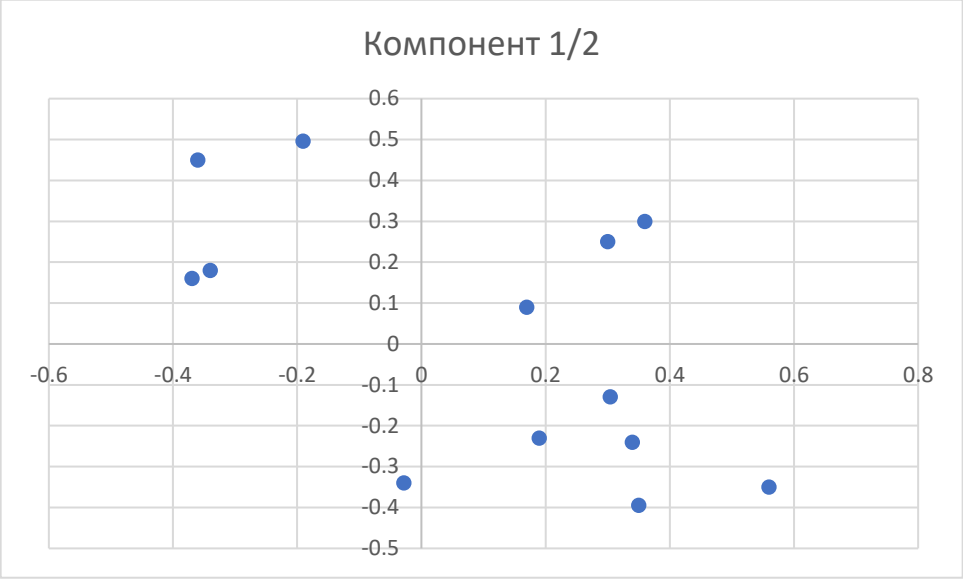

Рисунок 2.3 – проєкція двох головних компонент

На рисунку чітко видно, що вся досліджувана сукупність спостережень розділилась на три класи.

Характеристика класів щодо компонент представлена в таблиці 2.4.

Таблиця 2.4 – Характеристика класів компонент

Клас	Значення компоненти		
	Компонента 1	Компонента 2	Компонента 3
1	Високе	Високе	Середнє
2	Високе	Середнє	Середнє
3	Середнє	Серднє	Низьке

Характеристика класів щодо ознак представлена в таблиці 2.5.

Таблиця 2.5 – Характеристика класів ознак

Значення ознаки	Клас		
	1	2	3
Параметр нечіткості	Високе	Високе	Високе
Вага FMP	Високе	Середнє	Середнє
Вага FMT	Високе	Низьке	Високе

Кластерний аналіз із застосуванням дендрограм призначений для розбиття множини об’єктів на задане або невідоме число класів на підставі деякого математичного критерію якості класифікації.

Дендрограма – деревовидна діаграма, яка складається з n рівнів, кожен із яких відображає один з етапів процесу послідовного укрупнення кластерів.

Дендрограму також можна назвати деревовидною схемою або деревом поєднання окремих, схожих за властивостями, кластерів. Дендрограма описує приналежність окремих точок до кластерів, а також ступінь точності цієї приналежності, графічно представляє об'єднання чи розділення кластерів зберігаючи їх послідовність.

Дендрограми бувають горизонтальними або ж вертикальними. При використанні ієрархічних методів можливо легко визначити виключення в наборі даних й, як наслідок, підвищити не лише точність аналізу, а і якість самих цих даних для можливого іншого дослідження.

Ця процедура лежить в основі двокрокового алгоритму кластеризації. Такий набір даних готовий до використання для проведення неієрархічної кластеризації. Варто зазначити, що при використанні ієрархічних методів, на відміну від неієрархічних, визначення кількості кластерів не є обов’язковим в результаті побудови дерева ієрархічної структури вкладених кластерів.

Скористаємось сервісом STATGRAPHICS, щоб отримати дендрограму методом Варда для трьох кластерів (класів) (рис 2.4).

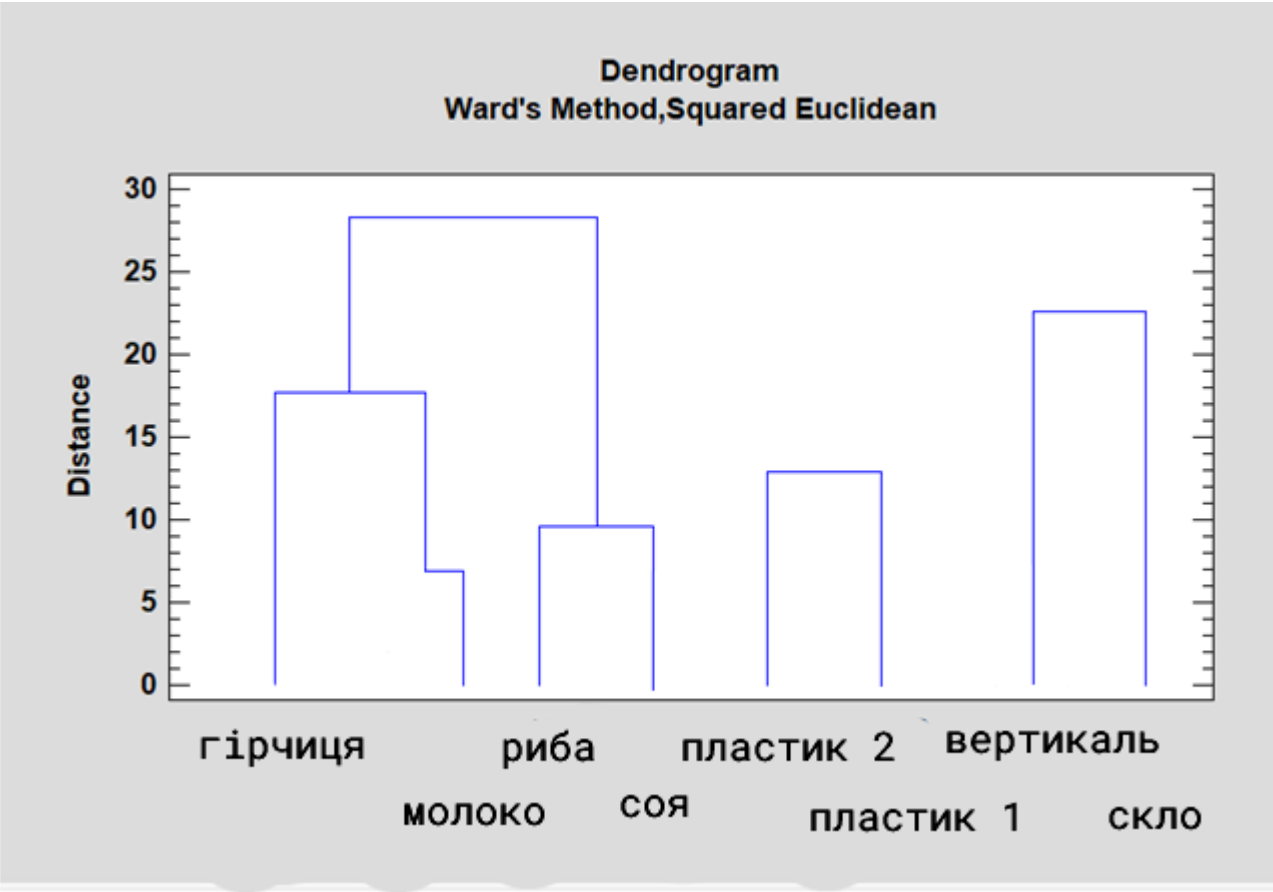

Рисунок 2.4 – Дердрограма для трьох кластерів

Після перетворень отримали зведення кластерного аналізу (таблиця 2.6).

Таблиця 2.6 – Зведення кластерного аналізу

Кластер	Кількість спостерігань	Відсоток спостерігань
1	4	50
2	2	25
3	2	25

Отриману таблицю центроїдів класів в результаті кластерного аналізу представлено у таблиці 2.7.

Таблиця 2.7 – Центроїди класів

Кластер	FMP	FMT	Параметр нечіткості
1	3057.14	571.429	170
2	3000.0	450.0	170
3	4333.33	516.667	165

Як видно з таблиці 2.7, в зведенні кластерного аналізу перш за все враховуються імена змінних, що беруть участь в аналізі, кількість повних зразків, використаний метод кластерного аналізу і прийнята метрика.

Потім, в зведенні описується число кластерів, кількість об'єктів в кожному кластері і відповідний відсоток. Крім того, в нижній частині зведення наводиться додаткова інформація за координатами центроїдів. За цим координатам можна судити про те, які змінні грають найбільш важливу роль в кожному кластері.

Для прикладу, кластер 1 та кластер 3 сильно відрізняються за показниками змінної FMP, кластер 2 та кластер 3 сильно відрізняються за показниками змінної FMT, але усі кластери майже не відрізняються за показниками змінної параметру чіткості.

Після того, як отримали результати кластерного аналізу, можна скласти таблицю приналежності об’єктів до класів.

Оскільки дані на показаному прикладі є примітивними, можна переконатись у тому, що схожі за смисловим навантаженням об’єкти були об’єднані в один клас. Умовно їх можна поділити на клас алергенів, клас типів пластику та клас умов транспортування та зберігання (таблиця 2.8).

Таблиця 2.8 – Приналежність об’єктів до класів

Кластер	Модель
1	Молоко
1	Риба
1	Соя
1	Гірчиця
2	Пластик 1
2	Пластик 2
3	Вертикаль
3	Скло

Розглянемо діаграму розсіювання (рисунок 2.5).

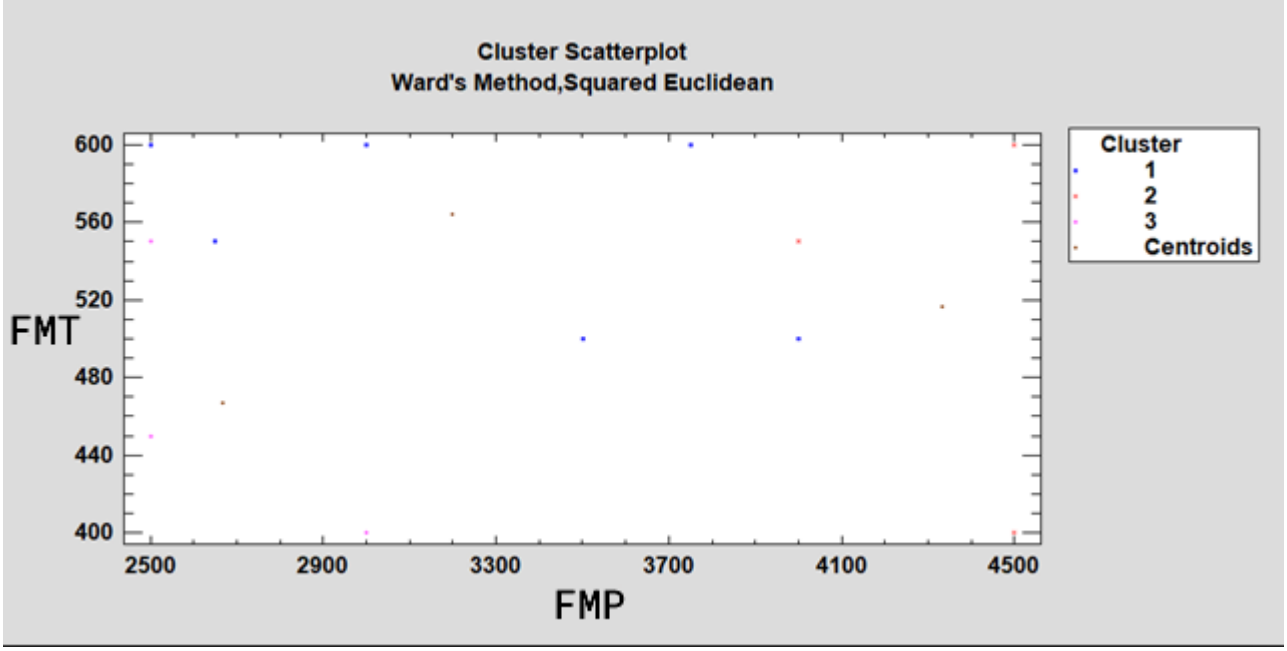

Рисунок 2.5 – Діаграма розсіювання двох змінних

Діаграма розсіювання показує, як групуються досліджуванні спостереження трьох змінних FMP, FMT, параметру нечіткості. Тепер розглянемо трьохмірну діаграму розсіювання (рисунок 2.6).

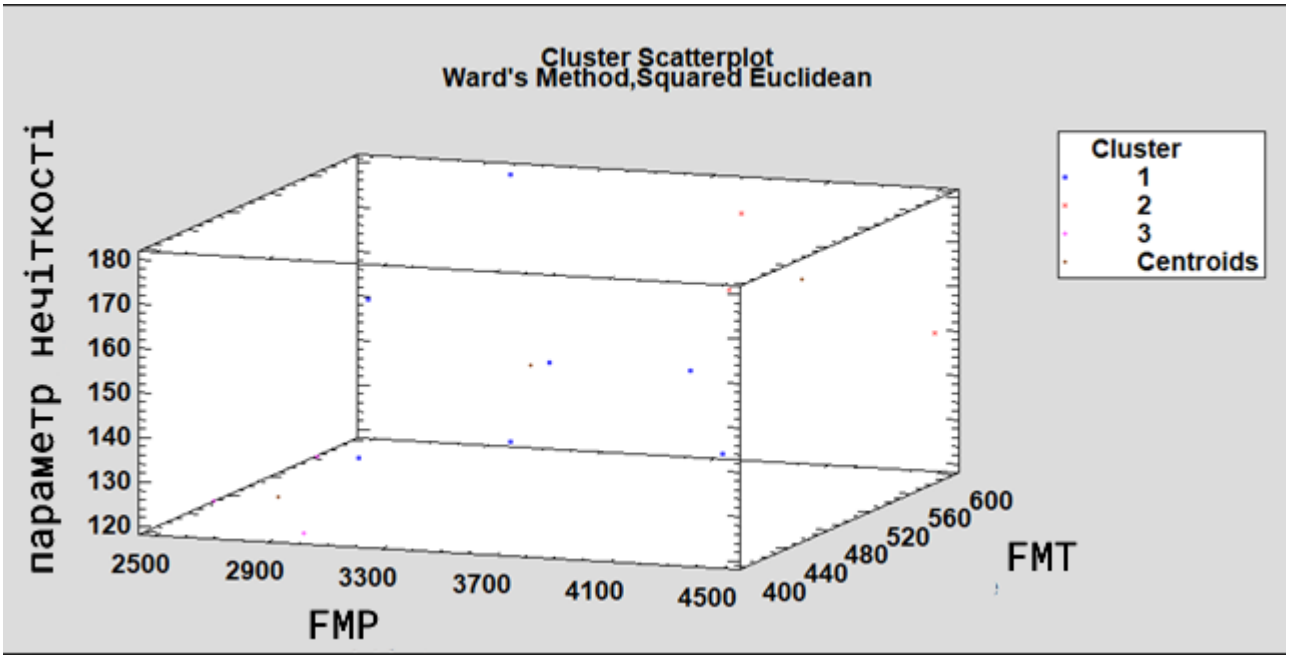

Рисунок 2.6 – Діаграма розсіювання трьох змінних

Діаграма розсіювання трьох змінних показує, як групуються досліджування спостереження в просторі змінних FMP, FMT, параметру нечіткості.

Таким чином, кластерний аналіз допоміг розділити множину із 8 спостережень та результатами їх аналізу за допомогою тоду нечіткої логіки на 3 класи. Ці дани можна дормалізувати та подати у вигляді бази знань.

2.3 Формування мережі виведення та бази знань

Для представлення даних використовується продукційна модель. У загальному вигляді продукційне правило представляється такою нотацією:

$$PR_i = <N, Q, P, C, A => B, S>,$$

де N – ім'я 1-го продукционного правила,

Q – сфера застосування, яка дозволяє визначити деяку область з безлічі продукційних правил, яка буде аналізуватися,

P – передумова, яке встановлює на безлічі правил виділеної сфери деякий порядок пріоритету їх використання,

C – умова застосування правила,

A => B – ядро продукционного правила, яке описує перетворення, со¬ставляющее сутність правила:

A – передумова (антіцендент),

B – висновок (консеквент),

S – дія, яку потрібно зробити після ядра.

Передумова і висновок можуть бути представлені декількома фактами: <передумова> = <факт 1 «і» факт 2> і ... і <факт n>.

База знаннь містить відомі факти, які виражені у вигляді об'єктів, атрибутів і умов, тобто факти представляють у вигляді об'єктів, описуючи їх атрибути і надаючи їм еквіваленти або значення.

У деяких випадках отримане дерево рішень може виявитися занадто складним для сприйняття. Наприклад, при побудові завдань високої розмірності для неоднорідних даних дерево нерідко виходить досить велике. Тому, з метою спрощення та лаконазації логічного висновку рекомендується використовувати логічну зв'язку «І».

Якщо за змістом існує логічна зв'язка «АБО», то формується друга така сама процедура, що містить тільки зв'язки «І».

Таблиця рішень являє собою ім'я змінної, її можливі значення і варіанти вирішення. Таблиці рішень для визначення сутностей Compl, Comp2, Comp3 і Клас наведені в таблицях 2.9, 2.10 та 2.11.

Таблиця 2.9 – Таблиця рішень для визначення сутності

Comp1 = велика	Comp1 = середня	Comp1 = мала	FMP	FMT
+			велика	велика
	+		велика	середня
+			велика	мала
		+	середня	велика
+			середня	середня
	+		середня	мала
	+		мала	велика
		+	мала	середня
		+	мала	мала

Таблиця 2.10 – Таблиця рішень для визначення сутності

Comp2 = велика	Comp2 = середня	Comp2 = мала	FMP	FMT
+			велика	велика
+			велика	середня
		+	велика	мала
	+		середня	велика
+			середня	середня
	+		середня	мала
		+	мала	велика
	+		мала	середня
		+	мала	мала

Таблиця 2.11 – Таблиця рішень для визначення сутності

Comp3 = велика	Comp3 = середня	Comp3 = мала	FMP	FMT
+			велика	велика
+			велика	середня
	+		велика	мала
		+	середня	велика
	+		середня	середня
+			середня	мала
		+	мала	велика
	+		мала	середня
		+	мала	мала

За отриманими даними можна скласти мережу висновків, або набір із правил поведінки для системи підтримки прийняття рішень, який пізніше міг би служити датасетом для нейронної мережі, яку також можна було б використовувати з метою ідентифікації наліпок на коборках.

Перевагою такого підходу є широкий спектр застосування технології, можливість її використання навіть на нових невідомих даних. Для прикладу, якщо коробка надійшла з новою невідомою наліпкою, то система зможе сама ідентифікувати її до однієї з відомих груп.

Недоліком такого підходу є те, що він складніший в обслуговуванні і повинен бути суттєво видозмінений у разу повного переформатування шаблонів наліпок (наприклад, перехід на штрих-коди).

В умовах того, що всі можливі групи наліпок мають скінчену відому кількість елементів (постачальники коробок завжди надають інформацію щодо свого

власного маркування), немає необхідності готувати систему підтримки прийняття рішень до будь-яких вхідних даних.

Отже будуємо мережу виведення за отриманими результатами кластеризації (рисунок 2.7).

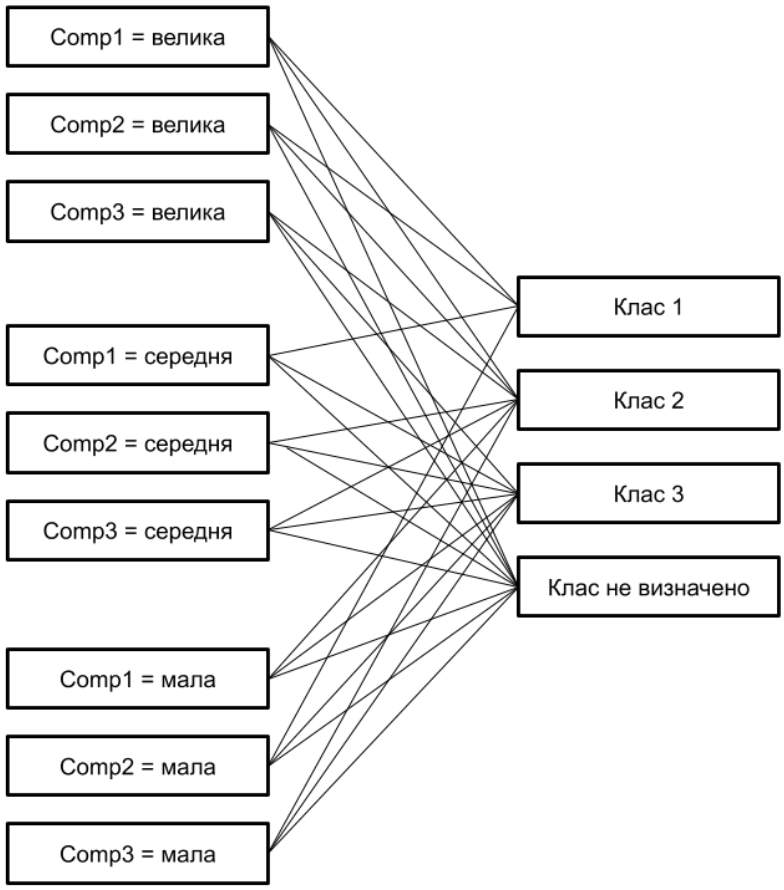

Рисунок 2.7 – Мережа виведення для класів

У ході кластеризації та аналізу таблиць рішень для визначення сутностей можна побудувати наступні правила поведінки системи прийняття рішень:

1. Якщо Comp1 = велика і Comp2 = велика і Comp3 = велика, то Клас = 1
2. Якщо Comp1 = велика і Comp2 = велика і Comp3 = середня, то Клас = 2
3. Якщо Comp1 = велика і Comp2 = велика і Comp3 = мала, то Клас = 1

4. Якщо Comp1 = велика і Comp2 = середня і Comp3 = середня, то Клас = 3
5. Якщо Comp1 = велика і Comp2 = середня і Comp3 = велика, то Клас = 3
6. Якщо Comp1 = велика і Comp2 = середня і Comp3 = мала, то Клас = 2
7. Якщо Comp1 = велика і Comp2 = мала і Comp3 = велика, то Клас = 1
8. Якщо Comp1 = велика і Comp2 = мала і Comp3 = середня, то Клас = не визначено
9. Якщо Comp1 = велика і Comp2 = мала і Comp3 = мала, то Клас = 2
10. Якщо Comp1 = середня і Comp2 = велика і Comp3 = велика, то Клас = 3
11. Якщо Comp1 = середня і Comp2 = велика і Comp3 = середня, то Клас = не визначено
12. Якщо Comp1 = середня і Comp2 = велика і Comp3 = мала, то Клас = 1
13. Якщо Comp1 = середня і Comp2 = середня і Comp3 = велика, то Клас = 2
14. Якщо Comp1 = середня і Comp2 = середня і Comp3 = седеня, то Клас = 2
15. Якщо Comp1 = середня і Comp2 = седерня і Comp3 = мала, то Клас = 2
16. Якщо Comp1 = середня і Comp2 = мала і Comp3 = велика, то Клас = не визначено
17. Якщо Comp1 = середня і Comp2 = мала і Comp3 = середня, то Клас = 2
18. Якщо Comp1 = середня і Comp2 = мала і Comp3 = мала, то Клас = не визначено
19. Якщо Comp1 = мала і Comp2 = велика і Comp3 = велика, то Клас = 1
20. Якщо Comp1 = мала і Comp2 = велика і Comp3 = середня, то Клас = 2
21. Якщо Comp1 = мала і Comp2 = велика і Comp3 = мала, то Клас = не визначено
22. Якщо Comp1 = мала і Comp2 = середня і Comp3 = середня, то Клас = 3
23. Якщо Comp1 = мала і Comp2 = середня і Comp3 = велика, то Клас = не визначено

24. Якщо Comp1 = мала і Comp2 = сердня і Comp3 = мала, то Клас = не визначено
25. Якщо Comp1 = мала і Comp2 = мала і Comp3 = велика, то Клас = 1
26. Якщо Comp1 = мала і Comp2 = мала і Comp3 = сердня, то Клас = 3
27. Якщо Comp1 = мала і Comp2 = мала і Comp3 = мала, то Клас = не визначено
28. Якщо FMP = велика і FMT = велика, то Comp1 = велика
29. Якщо FMP = велика і FMT = середня, то Comp2 = велика
30. Якщо FMP = середня і FMT = середня, то Comp3 = середня
31. Якщо FMP = середня і FMT = мала, то Comp3 = мала
32. Якщо FMP = мала і FMT = мала, то Comp2 = мала
33. Якщо параметр нечіткості = велика і FMT = велика, то Comp1 = велика
34. Якщо параметр нечіткості = середня і FMT = середня, то Comp1 = сердня
35. Якщо параметр нечіткості = середня і FMT = мала, то Comp2 = середня
36. Якщо параметр нечіткості = велика і FMT = мала, то Comp2 = середня
37. Якщо параметр нечіткості = мала і FMT = мала, то Comp3 = мала
38. Якщо FMP = велика і параметр нечіткості = велика, то Comp2 = велика
39. Якщо FMP = велика і параметр нечіткості = середня, то Comp1 = сердня
40. Якщо FMP = середня і параметр нечіткості = середня, то Comp3 = велика
41. Якщо FMP = середня і параметр нечіткості = мала, то Comp1 = середня
42. Якщо FMP = мала і параметр нечіткості = мала, то Comp2 = мала

Висновки до розділу 2

В ході вибору методики аналізу та кластеризації візуальної інформації було обрано теорію нечіткої логіки для отримання таблиці приналежностей об ’єктів до груп. Для подальшої кластеризації був обраний метод Варда та були застосовані дендрограми для відображення результатів аналізу.

Було складено мережу виведення та базу знань для ідентифікації стуностей до класів за їх ознаками.

Особливості підприємства, для потреб якого проводився аналіз вказують на те, що у зв ’язку із скінченністю вхідних даних немає необхідності використовувати отриману базу знань для нейронної мережі, а завдання ідентифікації наліпок може бути вирішене алгоритмічно зазначеними вище способами.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ МОДЕЛІ АНАЛІЗУ ТА ВІЗУАЛІЗАЦІЇ ІНФОРМАЦІЇ У СКЛАДСЬКИХ ПРОЦЕСАХ

Для програмної реалізації та установлення вище описаної системи підтримки прийняття рішень було обрано наступний стек технологій: MS SQL Server як система управління базами даних, .Net Framework, ASP.Net Web API для сервісної частини, WPF, Angular для клієнтської частини застосунків, які цього вимагають, WCF, Redis – для обміну даними, меседжингу між компонентами.

3.1 Система управління базами даних MS SQL SERVER

СУБД MS SQL-Server бере початок із 1989 року і з тих пір зазнав важливих змін. Це стосується масштабованості продукту, його цілісності, зручності адміністрування, продуктивності і функціональних можливостей.

Microsoft SQL Server - це реляційна система управління базами даних (СУБД). У реляційних базах даних дані зберігаються в таблицях. Логічно пов ’язані дані іноді групуються в набори таблиць, крім того, бувають встановлені також і взаємовідносини між таблицями. Це і означає назва реляційні – (англ relational) – споріднений, взаємозалежний, пов'язаний.

Користувачі доступаються до даних на сервері через додатки, а адміністратори, що виконують конфігурування, адміністрування та підтримку бази даних, мають засоби для безпосереднього доступу до серверів. SQL Server є масштабованою базою даних, це означає, що вона може зберігати значні обсяги даних і підтримувати роботу багатьох користувачів, що здійснюють одночасний доступ до бази даних.

Microsoft SQL Server 13.0 - одна з найбільш потужних СУБД з клієнт-серверним типом архітектури. Ця СУБД дозволяє задовольняти такі вимоги, що

пред'являються до систем розподіленої обробки даних, як рівнобіжна обробка, тиражування даних, підтримка масштабних баз даних на порівняно недорогих апаратних платформах при збереженні доступності управління і використання.

MS SQL Server не призначений безпосередньо для розробки користувальних додатків, а виконує функції керування базою даних. Сервер має засоби віддаленого адміністрування і керування операціями, організовані на базі об'єктно-орієнтованої розподіленої середовища управління.

Microsoft SQL Server 13.0 призначений винятково для підтримки систем, що працюють за типом клієнт-сервер. Підтримується широкий вибір засобів розробки і простота в інтеграції з іншими платформати та операційними системами.

SQL Server може підтримувати інформацію в базах даних різних форматів, включаючи IBM DB2, Oracle, Microsoft Access, Sybase та інші (при наявності ODBC драйвера, що відповідає визначеним вимогам).

Microsoft SQL Server 13.0 включає допоміжний продукт - Асистент адміністратора. Цей засіб дозволяє визначати основні процедури підтримки бази даних і вказувати графік виконання для них. Операції супроводу баз даних: перевірка розподілу сторінок та цілісності і актуальності індексів у таблицях (системні у тому ж числі), відновлення втраченої інформації, необхідної оптимізатору, реорганізацію сторінок таблиць та індексів, створення резервних копій таблиць і журналів з переліком транзакцій. Усі ці операції можуть бути налаштовані для автоматичного виконання за графіком, що визначить адміністратор.

Вимоги до програмного та апаратного забезпечення. Мінімальні вимоги до процесора: 486DX-33 (Intel Pentium), PowerPC, MIPS, R4xxx або Alpha AXP Операційна система із найбільш ранньою актуальною підтримкою: Windows NT 3.5 Мінімальний обсяг оперативної пам'яті, Гб: 16. Очікуваний обсяг на жорсткому диску, Гб: 256 (512).

Підтримка різних платформ. Одним з головних подій, що визначили подальшу долю Microsoft SQL Server, стало рішення Microsoft зосередити зусилля на підтримці не тільки платформи Windows NT, а й інших популярних Linux дестрибутивів включаючи Mac OS. Внаслідок його прийняття популярність SQL Server перш за все базується на популярності платформи, яку він підтримує. Таким чином, розвиток СУБД більше не залежить від розвитку операційної системи Windows, як це було раніше.

Настільні і однокористувацький версії. Ширина використання розподілених обчислень робить більш важливою властивість зберігати дані де завгодно, зокрема на робочій станції або переносному комп'ютері. Попри популярне твердження аналітиків про те, що в епоху хмарних застосунків та технологій настільні СУБД втрачають свою актуальність, вони як і завжди широко застосовуються у різних областях бізнесу. SQL Server можна застосовувати на будь-яких комп'ютерах під управлінням Windows чи Linux. Є також версія SQL Server для Windows CE, створена для розміщення на мобільних пристроях.

Однією з переваг SQL Server є зручність його використання, адміністрування, ескалації та розміщення. SQL Server Enterprise Manager, що входить до складу всіх редакцій Microsoft SQL Server (за винятком MSDE), виступає в ролі повнофункціонального і простого засобу для адміністрування всієї СУБД.

Що стутується продуктивності, то за даними Transaction Processing Performance Council (TPC), рекордсметом з виробничості на даний час є саме SQL Server.

Таким чином, головними перевагами SQL-Server є:

- високий рівень захисту даних;
- багатий функціонал роботи з даними;
- висока продуктивність;

– Оптимізація зберігання великих об ’мів даних;
– зберігання секретний даних, впровадження належного рівня безпеки даних;

Почавшись із невеликого, але амбітного проекту, цей продукт перетворювався на те, з чим користувачі все більше мають справу сьогодні. Основні функції в останніх версіях ще раз підтверджують той факт, що Microsoft підтримує розвиток своїх продуктів, намагаючись задовольнити зростаючі вимог споживачів.

3.2 Технологія ASP.NET WEB API

Active Server Page - активна серверна сторінка – технологія попередньої обробки, що використовується для створення динамічних веб-сторінок, які обробляються та генеруються на стороні сервера.

Динамічні сторінки - це сторінки, контент яких генерується та відмальовується залежно від дій користувача. Протилежні їм статичні сторінки завжди однакові, відображають сталу інформацію не залежно від того, як із нею взаємодіє користувач. Спочатку все сторінки були статичними, проте розвиток Інтернету викликав потребу в наданні мінливої інформації. Простими прикладами є прогнози погоди, які оновлюються, новини, чи курси валют. Складнішими прикладами є інтернет-магазини, онлайн сервіси з обробки чи аналізу даних, стрімінгові платформи. [43]

Із зростанням потреби в динамічно змінюваних web-сторінках стали розвиватись і технології їх реалізації, однією з яких і стала технологія ASP.

ASP - технологія Microsoft, що дозволяє легко розробляти програми для web. ASP одинаково ефективно працює на платформі операційних систем лінії Windows NT та на веб-сервері IIS.

ASP не є мовою програмування - це лише технологія попередньої обробки, що дозволяє підключати програмні модулі під час процесу

формування Web-сторінки. Відносна популярність ASP заснована на простоті використовуваних мов сценаріїв (VBScript або JScript) та можливості використання зовнішніх COM-компонент.

Технологія ASP передбачає використання серверних та клієнтських сценаріїв і роботу з об'єктами COM для розробки динамічних web-серверів. Засобами технології ASP можна легко створювати динамічні web-сторінки, в режимі реального часу виконувати обробку даних, отриманих від клієнта, працювати з базою даних.

Особливостями ASP, які приваблюють розробника, можна вважати:

- зручний спосіб об'єднання серверних сценаріїв с HTML;
- скриптова підхід (інтерпретована мова) - тобто файл з вихідним кодом ASP одночасно є його виконуваним файлом, що спрощує розробку та підтримку;
- явище "Session" – спільний контекст запитів для кожного користувача, рішення вічної проблеми stateless-протоколу HTTP;
- надання засобів для організації розподіленої архітектури, що побудована на основі інфраструктури COM, DCOM, COM +.
- зручний передвизначений набір об'єктів, які інкапсулюють роботу із HTTP запитом: Request, Response, Server, Application, Session, ObjectContext.

Логіка роботи сторінки прихована від користувача. Вінне може будь-яким чином отримати контент ASP сторінки чи її частини, тому що сервер повертає йому не саму сторінку, а її інтерпретацію.

ASP.NET Core -це новий фреймворк з відкритим вихідним кодом для створення архітектури та програмної реалізації крос-платформенних хмарних застосунків, наприклад, веб застосунків, застосунків Інтернет Речей та мобільних застосунків. ASP.NET Core програми можуть запускатись на .NET Core або на

повній версії .NET Framework. Підхід був розроблений, щоб забезпечити зручне середовище розробки для різноплатформених застосунків, що можуть бути розгорнуті в хмарі або таких, які прцюють на спеціальних виділених серверах.

Рішення є модульним, поєднання яких відбувається з мінімальними витратами, за рахунок зберігається гнучкість побудови потрібних рішень. ASP.NET Core програми можна запустити крос-платформенно: на Windows, Mac і Linux. ASP.NET Core має відкритий вихідним код на GitHub. [44]

Перша поява ASP.Net відбулася майже 18 років тому в якості частини .NET Framework. З цього часу його використовували для створення і розміщення великих веб-застосунків (Рисунок 3.1).

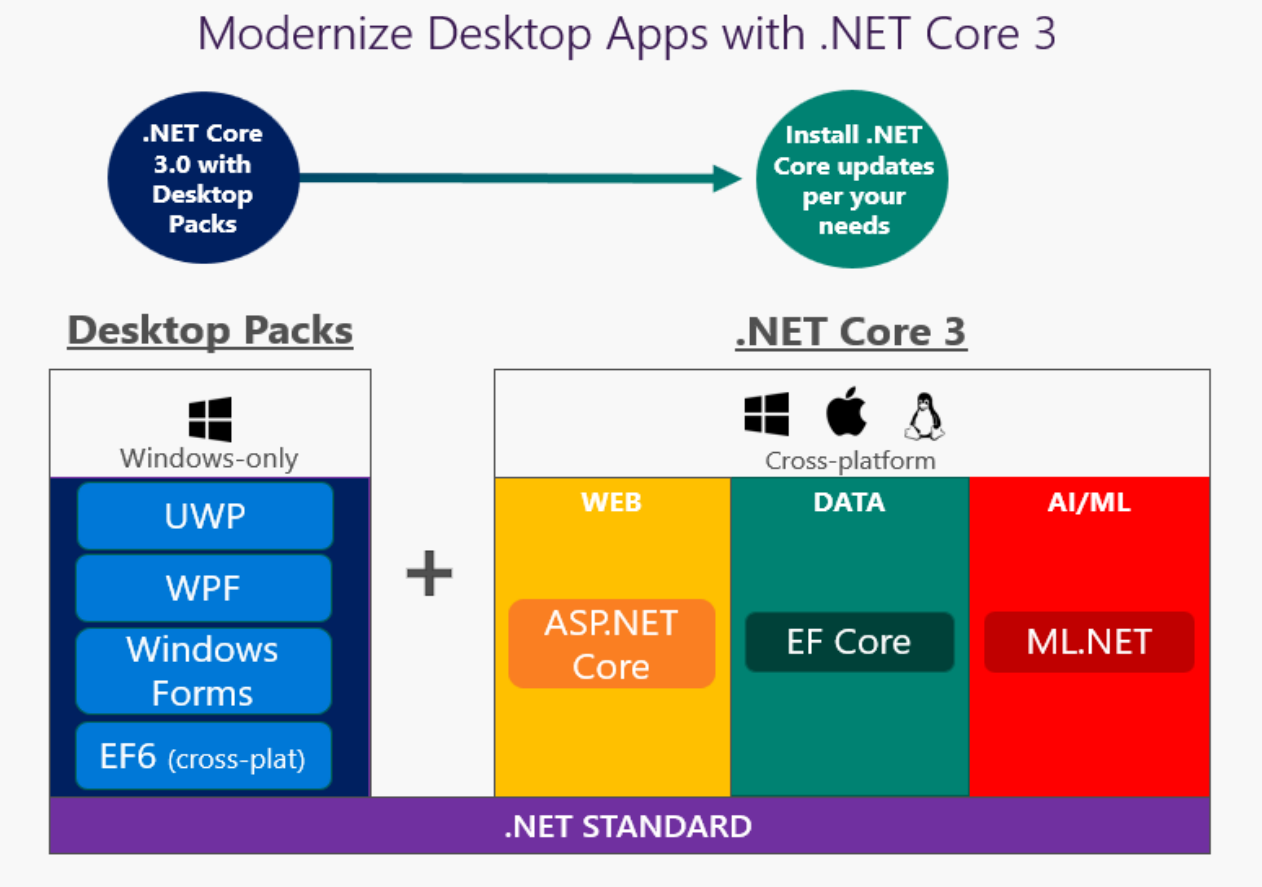

Рисунок 3.1 - Схема компонентів .Net Core 3

Окрім того, технологія включає в себе деякі компоненти та технології, які раніше використовувались як розширення та доповнення. Прикладом таких технології є ін'єкція залежностей – підхід, який дозволяє компонентам взаємодіяти між того без тісного зв'язування, а лише на рівні інтерфейсів.

ASP.NET Core має цілий перелік змін в архітектурі, які зробили фреймворк більш компактним, надали йому модульну структуру яка збільшує можливості машстабування додатків.

Реворюцією ASP.NET Core (ASP.Net 5.0) у порівнянні з попередньою версією було те, що фреймворк перестав бути заснованим на System.Web.dll. Тепер він реалізований на комплексі гранульованих і добре розподілених NuGet пакетів. Це дозволяє спростити застосунок, включаючи у нього лише необхідний базовий функціонал. Менша кількість залежностей застосунку збільшує рівень контролю системи безпеки, зменшує потреби в обслуговуванні, підвищує виробничість та збільшує відсоток значущих компонент застосунку. [45]

Серед ключових характеристик ASP.Net Core варто виділити:

- самостійна та незалежна платформа для створення веб-сервісів;
- інтеграція новий фреймворків для розробки клієнтської частини веб-застосунків;
- система налаштування та імітації середовища, в більшості орієнтована на підтримку хмарних технологій;
- вбудований Dependency Injection;
- легкий і модульний медіатор HTTP запитів;
- можливість запуску на IIS або на окремому сервері чи процесі;
- підтримує інтергацію із популярними системати контролю версій;
- повністю постачається та підтримується у вигляді колекції NuGet пакетів;

– можливість розробки та відладки крос-платформних застосунків ASP.NET на Windows, Mac чи Linux ;
– присутність відкритого вихідного коду та підтримка спільноти розробників.

ASP.Net Core дає можливість створювати HTTP-сервіси, що мають широке коло клієнтів, включаючи браузері та мобільні пристрої.

Також надається вбудована підтримка декількох форматів даних та домовленостей щодо вмісту. Ще варто додати, що сервіси ASP.Net Core, як правило, побудовані на паттерні MVC (Model-View-Controller). Це визначає та контролює хід розробки і тестування, спрощує написання інтеграційних тестів для окремих компонент сервісу. (Рисунок 3.2)

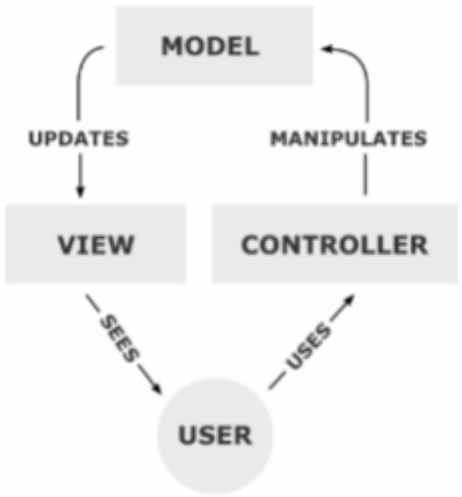

Рисунок 3.2 - Патерн MVC

Сам патерн MVC не є новою ідеєю в архітектурі застосунків, він з'явився ще в 1970-х роках. Компанія Xerox обрала такий підхід як спосіб організації візуальних та програмних компонентів в графічному застосунку на мові Smalltalk.

Концепція паттерна MVC передбачає поділ застосунку на три складових:

Модель (model): дає опис даних, що використовуються в застосунку, а також логіку, яка взаємодіє з цими даними, Це може бути валідація, обчислення, перетворення, візуалізація. Як правило, об'єкти моделей зберігаютьсяв базі даних. У MVC є два типи моделей даних: це доменні об'єкти, які використовуютсья для взаємодії на бізнес-рівні, та моделі представлень, які використовуються для відображення і передачі даних для їх подальшої візуалізації. Модель може містити дані, або ж містити логіку управління такими даними. Одночасно модель не повинна знати нічого про взаємодію з користувачем і не повинна впливати на механізм обробки запиту клієнта. Також, модель не повинна мати логіку щодо візуалізації даних для клієнта. [45]

Представлення (view): виступає візуальною частиною, призначений для клієнтського інтерфейсу. Згенерована view це html-сторінка, через яку здійснюється взаємодія користувача із застосунком. Також представлення може містити деяку логіку, пов'язану лише з візуалізацією даних. Разом з ти відображення не повинно мати логіку обробки чи фільтрації запиту користувача. Дані можуть передаватись між відображенням та контролером, але не можуть передаватись між відображенням та моделлю даних і навпаки.

Контролер (controller): це центральний компонент MVC, який забезпечує зв'язок між інтерфейсом користувача та сервером моделі, він відповідає за подання, обробку, валідацію запитів. Він єдиний компонент, що може обробляти запит користувача. Контролер на вхід отримує введені користувачем дані і направляє роботу на модель. В залежності від результатів обробки відправляє користувачеві певний результат, найчастіше це є згенероване представлення з результатами запиту користувача.

Важливим є той момент, що ASP.NET сам піклується про безпечність потоків, тому для розробки системи незначних масштабів користувачу не потрібно тубруватись про критичні секції контролера.

При такому підході модель є незалежним компонентом - будь-які зміни контролера або відображення ніяк не можуть впливати на модель. Модель повинна бути готового до того, щоб використовуватись у різних застосунках з різним типом взаємодії з користувачем. Контролер і представлення - відносно незалежні компоненти. Так, з представлення можна звертатися до пов’язаного контролеру, а з контролера можна генерувати представлення. При цьому їх нерідко можна змінювати незалежно один від одного. (Рисунок 2.3)

Рисунок 3.3 - Потік даних MVC

Таке розмежування компонентів застосунку уособлює концепцію розділення відповідальності, за якої кожен компонент відповідає за свою строго визначену задачу. Це сильно полегшує структуру роботи окремих компонентів і спрощує можливість її розширення. Завдяки цьому застосунок легший у розробці, підтримці та тестувані своїх компонентів незалежно. Якщо для розробки важливішою є візуальна частина або фронтенд, то є можливість тестувати представлення окремо від контролера. Або навпаки, можна виокремити контроллер і тестувати бекенд окремо. [46]

Однак повністю асоціювати цілу платформу ASP.NET Core з MVC не є правильно. MVC - це лише шаблон, що реалізується в межах платформи. Завжди існує можливість створити проект без шаблону, де не буде жодних передвизначених контролерів, моделей, відображень, де буде присутній лише один клас Startup. І через нього розробниу буде будувати всю логіку обробки запитів. Але зазвичай застосування MVC полегшує процес розробки застосунків.

Часто модель у застосунках MVC будується на базі Entity framework. Таким чином, зникає потреба у підключенні чи ручній розробці інтерфейсу бази даних для веб-застосунку.

Модель прив'язки автоматично передає дані з HTTP-запитів до безпосередніх дій сайту.

Модель валідації автоматично підтримує перевірку даних на стороні клієнта, так і сервера.

ASP.NET Core відкритий для розширень та зручно реалізований для інтеграції з популярними фреймворками для створення інтерфейсів користувача, серед яких Angular, Knockout, React.

3.3 Фреймворк для десктопної розробки інтерфейсу користувача WPF

Технологія WPF (Windows Presentation Foundation) - частина екосистеми платформи .NET, підсистема для побудови графічних інтерфейсів.

Якщо для створення традиційних додатків на основі WinForms за відображення елементів управління і графіки відповідали такі частини ОС Windows, як User32 і GDI +, то додатки WPF засновані на DirectX. У цьому полягає ключова особливість рендерингу графіки в WPF: коли WPF рендерить об’єкти, значна частина роботи з відображення графіки від найпростіших кнопок до складних 3D-моделей лягає на графічний процесор на відеокарті, що також дозволяє скористатися апаратним прискоренням графіки. [47]

Однією з важливих особливостей є використання мови декларативною розмітки інтерфейсу XAML, заснованого на XML: можна створювати насичений графічний інтерфейс, використовуючи або декларативне визначенняч інтерфейсу, чи код на керованих мовах C # і VB.NET. (Рисунок 3.4)

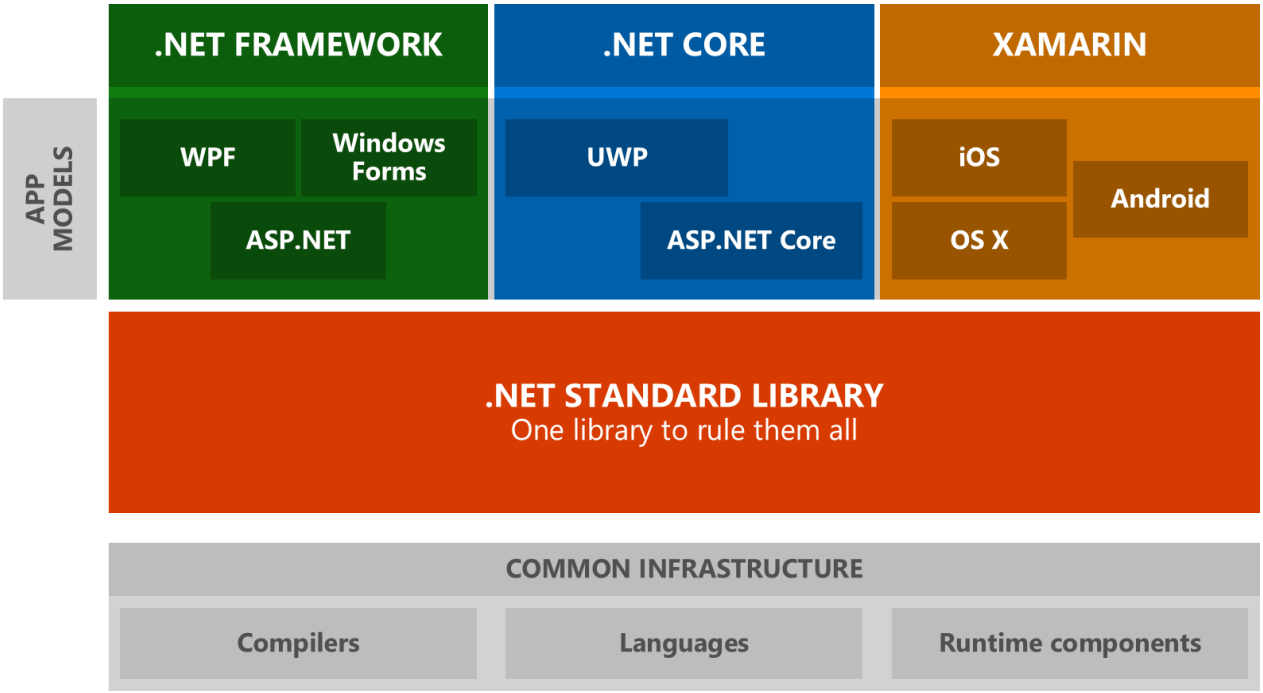

Рисунок 3.4 - Модулі .Net застосунків

Основні переваги WPF:

- використання прийнятих мов .NET-платформи - C # і VB.NET для створення бекенду застосунку;
- можливість визначення графічного інтерфейсу за допомогою спеціальної визначеної мови розмітки XAML, заснованому на xml, програмному створенню графіки та елементів управління, а також шляхом комбінування XAML і C # / VB.NET;

- незалежність від розширення екрану: оскільки в WPF всі елементи вимірюються в незалежних від пристрою одиницях (DPI – dots per inch – точки на дюйм), додатки на WPF легко масштабуються під різні екрани з різним дозволом;
- нові можливості, яких неможливо було б досягти в WinForms, наприклад, створення тривимірних моделей, прив'язка даних у моменті виконання, використання таких спільних елементів, як стилі, шаблони, теми і ін.;
- достатню взаємодію з WinForms, завдяки чому, наприклад, в додатках WPF можна використовувати традиційні елементи управління з WinForms;
- широкі можливості зі створення різних додатків: це і мультимедіа, і тривимірна графіка, і багатий набір стандартних елементів управління, а також можливість створювати власні специфічні елементи керування, створення анімацій, прив'язка даних,шаблони, теми, стилі і багато іншого;
- апаратне прискорення графіки - незалежно від того, чи працюєте ви з 2D або 3D, графікою або текстом, все компоненти програми транслюються в об'єкти, зрозумілі Direct3D, і потім візуалізуються за допомогою процесора на відеокарті, що підвищує продуктивність, робить графіку більш плавною;
- створення додатків під безліч ОС сімейства Windows - від Windows XP до Windows 10. [47]

У той же час WPF має певні обмеження. Незважаючи на підтримку тривимірної візуалізації, для створення додатків з великою кількістю тривимірних зображень, перш за все ігор, краще використовувати інші засоби - DirectX або спеціальні фреймворки, такі як Monogame або Unity.

3.4 Фреймворк Angular

Angular – оновлена версія популярного раніше фреймворка для розробки клієнтської частини односторінкових додатків від Google. Angular досі знаходиться на стадії активної розробки та підтримки, однак вже користується великою популярністю, оскільки запропонована повноцінна інфраструктура додатка дозволяє вирішувати багато проблем, які не вирішують інші фреймворки.

Серед його можливостей:

- Зв’язування
- Відкритий сирцевий код (розповсюджується за MIT ліцензією)
- Шаблони (прототипування)
- Компонента архітектура додатка
- Впровадження залежностей
- Клієнтська маршрутизація

Angular – це один з найновіших й найбільш багатих за можливостями фреймворк, який не лише допомагає в розробці односторінкових додатів а й пропонує повноцінну інфраструктуру з використанням багатьох нових допоміжних технологій, нприклад TypeScript й ECMAScript6. Також Angular включає в себе багато мінорних бібліотек для вирішення локальних проблем, які звели до мінімуму проблеми першої версії, наприклад Zone.js [48]

Також варто враховувати, що в порівнянні з іншими фреймворками для односторінкової розробки обсяг програм на Angular і споживання ними пам'яті в процесі роботи в середньому трохи вище. Але це з надлишком компенсується більш широкими графічними можливостями і підвищеною продуктивністю при відображенні графіки.

Вагомим аргументом є технологія Angular Native, яка дозволяє створювати застосунки для мобільних пристроїв за веб технологією з використанням нативних функції операційної системи.

3.5 OPC уніфікована архітектура

OPC Unified Architecture (англ. Уніфікована архітектура OPC) - специфікація, що визначає передачу даних в промислових мережах і взаємодія пристроїв в них. Розроблено промисловим консорціумом OPC Foundation і значно відрізняється від його попередніх специфікацій. Перша версія Уніфікованої архітектури OPC була випущена після 3 років робіт над специфікацією і ще 1 року прототипирования. [49]

Попередні специфікації OPC Foundation спиралися на механізми COM / DCOM, але незважаючи на те, що прив'язка до COM / DCOM допомагає OPC добре працювати в розподілених системах, є також і негативні сторони:

- часті проблеми з конфігурацією DCOM
- неконфігурабельні тайм-аути
- доступність тільки в операційних системах Microsoft Windows
- непридатність безпеки
- відсутність управління поверх DCOM (COM / DCOM надаються чорним ящиком, розробники не мають доступу до вихідного коду і тому змушені жити з багами або неповними реалізаціями)

Ці та інші причини породили рішення про розробку нового незалежного комунікаційного стека для OPC UA, який замінить COM / DCOM. Очікуваними характеристиками цього стека будуть:

- реалізація на транспортація мови програмування ANSI C
- масштабованість від вбудованих систем до мейнфреймів

– стек може бути скомпільовано як для багатопотокового, так і для однопотокового застосунку, що необхідно для портування стека на вбудовані пристрої
– власна реалізація безпеки, заснована на нових стандартах, що реалізують «справжню» безпеку
– настройка тайм-аутів для кожної служби
– формування великих датаграм.

Цей комунікаційний стек відображає тільки початок різних інновацій. Уніфікована архітектура OPC є сервісно-орієнтованою архітектурою (SOA) і заснована на різних логічних рівнях.

Всі Базові Служби певні OPC є описами абстрактних методів, що не залежать від протоколу і утворюють основу всієї функціональності уніфікованої архітектури OPC. Транспортний рівень цих методів в протокол, що має на увазі серіализацию / десеріалізацію даних і їх передачу по мережі. (Рисунок 3.5)

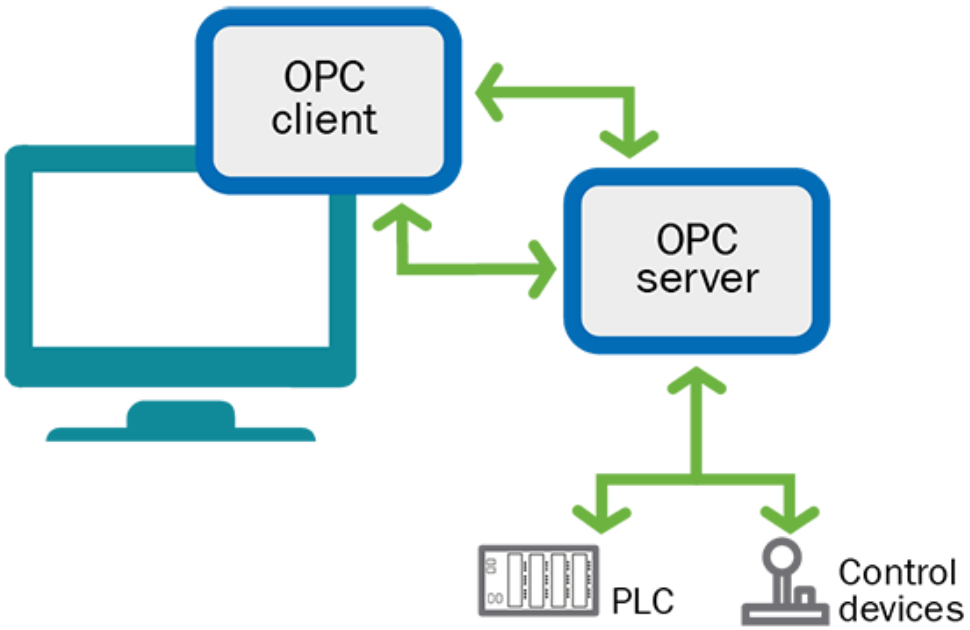

Рисунок 3.5 - OPC UA архітектура.

На даний момент існує два протоколи призначених для цих цілей. Один є двійковим, щодо високопродуктивно оптимізований протокол TCP і другий, на основі веб-служб. Інформаційна модель OPC більше не ґрунтується тільки на ієрархічних папках, елементах і властивості, замість цього використовується, так звана, повна чарункова мережа, заснована на вузлах. Крім того, ці вузли можуть передавати все різноманіття метаінформації.

У найпростішому випадку вузол можна порівняти з об'єктом, відомим з об'єктно-орієнтованого програмування. Він має атрибути доступні для читання (DA, HDA), методи, які можуть бути викликані (Commands), і ініціюють події (triggering events), які можуть бути передані (AE, DA DataChange).

Вузли використовуються для обробки даних нарівні з усіма іншими типами метаданих. Використовуваний простір імен OPC тепер також включає модель типів, використовувану для опису всіх можливих типів даних. Грунтуючись на вищевикладеному, інші організації, наприклад, EDDL, визначають своє власне джерело інформації.

Клієнтське програмне забезпечення може мати можливість перевірки того, який з так званих профілів підтримує сервер. Додатково, може бути отримана інформація про те, чи підтримує сервер, наприклад, профіль EDDL і, отже, клієнт може дізнатися, що тут також є опис певного EDDL пристрою. Доданими новими і важливими можливостями OPC UA є:

- Підтримка резервування.
- Пульс з'єднання в обох напрямках. Це означає, що і сервер, і клієнт розпізнають роз'єднання.
- Буферизація даних і підтвердження передачі даних. Втрата з'єднання більше не призводить до втрати даних. Втрачені датаграми доставляються повторно.

Як описано вище є два протоколи. Прикладний програміст буде розпізнавати це тільки через відмінність в URL передані їм для двійкового протоколу opc.tcp: // server і http://server/ для веб-служб. Інакше кажучи, OPC UA працює повністю прозоро для API.

1. Двійковий протокол:
   - краща продуктивність, мінімальні накладні витрати
   - споживає мінімум ресурсів (не потрібні парсер XML, SOAP і HTTP, що важливо для вбудованих пристроїв)
   - найкраща можлива сумісність (двійковий код визначений явно і допускає меншу ступінь свободи в процесі виконання на відміну від XML.
   - всього один порт TCP (4840) використовується для комунікації і легко може тунелюватись або пропускатись через міжмережевий екран.

2. Веб-служби (SOAP):
   - найкраща підтримка з доступних інструментів. Легко може бути використаний, наприклад, з оточення Java або .Net.
   - можна застосовувати з міжмережевими екранами. Порт 80 (http) і 443 (https) зазвичай здійснюється без додаткових налаштувань.

Так як стек ANSI C підтримує обидва протоколи, то очікується, що більшість кінцевих продуктів зможуть обмінюватися інформацією щодо більш ефективного бінарного протоколу. Але за необіхдності фреймворк дозволяє легко підключити SOAP протокол, що дасть можливість працювати із об’єктною моделлю замість сирих бінарних даних.

В архітектурі застотсунку OPC UA, незалежно від того, чи це клієнська чи серверна частина, можна виділити рівні, зображені на рисунку 3.6.

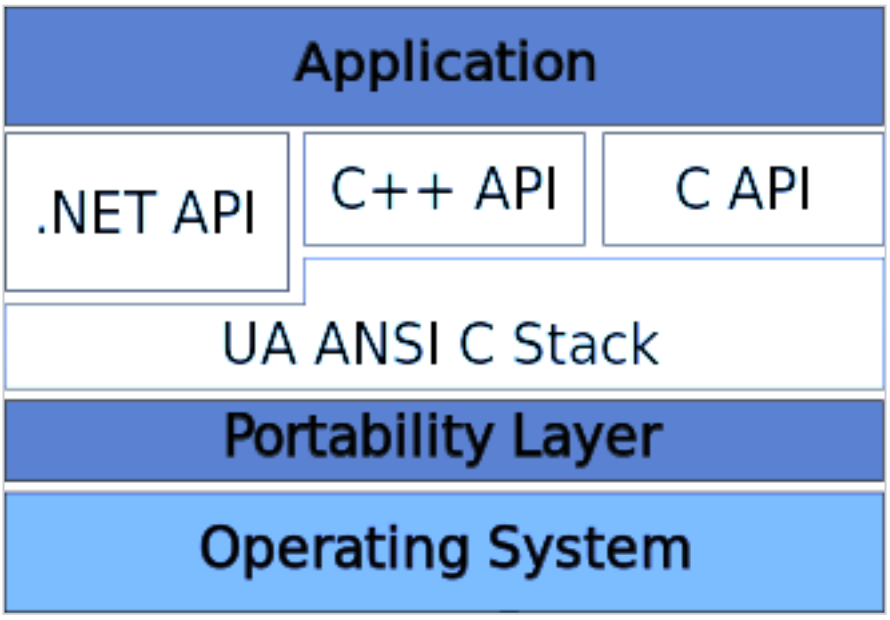

Рисунок 3.6 - Рівні застосунку OPC UA

Зелені частини відповідають COM проксі / заглушкам і надаються OPC Foundation. Новим є рівень портування, який дозволяє легко переносити стек UA ANSI C на інші платформи. Портована версія для Windows і Linux також надається OPC Foundation. Як описувалося раніше, можуть бути розроблені додатки засновані на API, подібно до того як вони програмувались в минулому для COM. На конференції OPC UA DevCon в жовтні 2006 в Мюнхені в живу були представлені перші прототипи. Компанія ascolab GmbH, яка також розробила стек ANSI C для OPC Foundation, продемонструвала різноманітні прототипи взаємодії UA клієнта під Windows / .NET і UA сервера під Linux. До того ж різні UA сервери були показані на Beckhoff PLC і вбудованих тестових платах європейських виробників. Через те, що Beckhoff PLC засновані на Windows XP Embedded і вбудований контроллер заснований на ОСРВ європейських виробників.

Безпека UA складається з аутентифікації і авторизації, шифрування і забезпечення цілісності даних за допомогою сигнатур. Для цього OPC Foundation не стала заново винаходити колесо, а орієнтувалася на специфікації Web Service Security. Для веб-служб використовуються WS Secure Conversation і отже вони

сумісні з .NET і іншими реалізаціями SOAP. Для двійкового протоколу дотримуються алгоритми WS Secure Conversation і також конвертуються в двійковий еквівалент. Це тепер називається UA Secure Conversation. Також присутня змішана версія, де код двійковий, але транспортним рівнем є SOAP. Це компрометує ефективність двійкового кодування і зручною для міжмережевих екранів передачі. Двійкове кодування завжди вимагає UA Secure Conversation. При аутентифікації використовуються виключно сертифікати x509. Вибір схеми сертифікації, використовуваної додатком, покладається на розробників додатків. Наприклад, можна використовувати Public Key Infrastructure (PKI) з Active Directory.

3.6 Docker

Docker є найпопулярнішою платформою, що використовує технологію контейнеризації. Платформа Docker вирішує такі основні проблеми розміщення сервісів:

- оновлення коду на сервері;
- Компіляція та миттєвий запуск коду;
- кросплатформеність оточення.

Docker дозволяє ховати сервіси від інфраструктури, на якій вони запускатимуться, таким чином стає доступною можливість доставляти їх не залежно від платформи. Docker дозволяє керувати інфраструктурою за рахунок тих же принципів, що використовуються для управління додатками. При використанні інструментів Docker для встановлення, тестування і розгортання сервісів, значно зменшується затримка між створенням нового коду в системі контролю версій та запуском серверу, що базується на цьому новому коді.

Docker надає можливість упаковувати і запускати сервіси в ізольованому оточенні, яке і називається контейнером. ДТакаана ізольованість і обмеженість

дозволяє запускати будь-яку кількість контейнерів на одному логічному хості одночасно.

Контейнери налаштовані одинаковим чином, тому не вимагають роботи супервізора. Вони запускаються безпосередньо на ядрі хост машини. Таким чином стає можливим запуск великої кількості контейнерів, якщо порівнювати це із запуском компонентів на віртуальних машинах на одному і тому ж фізичному обладнанні. Звісно, Docker контейнери можна також запускати і на самих віртуальних машинах.

Після розробки, додаток може бути перебудований та розміщений на сервері експлуатації вручну чи за допомогою оркестратора адміністрування контейнерів. Така методика розгортання одинакова незалежно від того, в якому середовищі розгортається застосунок: чи це локальному сервер, чи хмарний провайдер чи гібридне тестове оточення.

Docker надає зручні інструменти для управління життєвим циклом контейнерів, а також платформу для реалізації цих засобів. За допомогою Docker відбувається обгортання застосунків чи їх компонентів в контейнер. В екосистемі Docker, контейнер стає атомарним юнітом для поширення і тестування програми.

Docker оптимізує життєвий цикл розробки, дозволяючи розробникам працювати в стандартизованому оточенні. При використанні Docker, оточення при розробці нічим не відрізняється від оточення в промисловій експлуатації. Контейнери ідеально підходять для безперервної інтеграції або безперервного постачання. Звичайний сценарій розробки програмного забезпечення зводиться до отримання якогось артефакту, який будується після будь-якої зміни вихідного коду програми. [50]

При використанні Docker, цим артефактом може бути контейнер. Розробники або інженери з тестування використовують докер контейнери для розгортання додатків на тестове оточення і запускають автоматичні або ручні

тести. Якщо під час перевірок виявляється баг, даний контейнер може бути розгорнуто в оточенні розробки, виправлено і перенесено назад в тестове оточення, для повторної перевірки. Якщо всі перевірки успішно завершені, досить доставити контейнер з виправленнями в промислове оточення.

Контейнерна платформа Docker забезпечує високу транспортабельність. Контейнер може виконуватися на робочій станції розробника, фізичних або віртуальних машинах в дата центрі, в інфраструктурі хмарного провайдера. Транспортабельність докера, а також його легковажність дозволяють динамічно міняти завантаження, збільшуючи або зменшуючи кількість додатків і сервісів, практично в реальному часу.

Docker легкий і швидкий. Він надає життєздатну, економічно вигідну альтернативу віртуальним машинам на основі гіпервизора. За допомогою Docker можна використовувати більше обчислювальних потужностей, оскільки ці ресурси не витрачаються на обслуговування гіпервизора.

Docker використовує клієнт-серверну архітектуру. Клієнт звертається до фонового процесу – Docker серверу, який вже самостійно займається запуском контейнерів. Клієнт і сервер можуть запускатись в межаї однієї операційної системи, а також є можливість підключення клієнта до віддаленого серверу.

Docker клієнт і сервер взаємодіють через REST API або через локальний мережевий інтерфейс. Фоновий процес Docker (dockerd) приймає запити і керує об'єктами Docker. Фоновий процес може також взаємодіяти з іншими сервісами чи процесам. Фоновий процес керує такими об'єктами:

- контейнери;
- образи;
- мережеве взаємодія;
- томи.

Docker клієнт - це головний спосіб взаємодії з Docker сервером. Для виконання будь-якої команди клієнт надсилає її серверу, який в свою чергу її виконує. Docker клієнт може взаємодіяти з будь-якою кількістю різних Docker серверів.

Висновки до розділу 3

В ході аналізу існуючих технологій для створення програмних систем, що можуть бути застосовані на автоматизованому складі, було обрано такий стек технологій: MS SQL Server як система управління базами даних, .Net Framework, ASP.Net Web API для сервісної частини, WPF, Angular для клієнтської частини застосунків, які цього вимагають, WCF, Redis – для обміну даними, меседжингу між компонентами.

Необхідність використання саме цих технологій пояснюється тим, що застосунки створені та базі кожної з них можуть бути поміщені в Docker контейнер для подальшого розгортання на серверах складу.

Платформонезалежність рішення дозволяє власникам складу встановлювати сервери будь-якої архітектури та будь-якої операційної системи з одинаковим кінцевим результатом.

РОЗДІЛ 4
ЗАСТОСУВАННЯ МОДЕЛІ АНАЛІЗУ ТА ВІЗУАЛІЗАЦІЇ ІНФОРМАЦІЇ У ВИГЛЯДІ ГІБРИДНИХ ЗАСТОСУНКІВ У СКЛАДСЬКИХ ПРОЦЕСАХ

4.1 Особливості гібриднх застосунків

Гібридні додатки являють собою поєднання веб і нативних додатків. Особливо, мається на увазі їх кроссплатформенність і доступ до функціоналу пристрою, на якому встановлений застосунок.

Серед багатьох компаній вибір найчастіше падає на розробку саме гібридного додатку. Це можна пояснити тим, що гібридні програми здатні поєднувати плюси нативних за технологічною актуальністю, яка забезпечується останніми веб-технологіями чи десктоп-технологіями. Однак, на відміну від нативних, вартість створення гібридних на порядок нижче, а швидкість - вище.

Спорідненість гібридних додатків з веб-додатками, в свою чергу, дає плоди у вигляді того, що в них можна легко і оперативно вносити корективи. Тобто розробникам не доводиться, як у випадку з нативними, повторно розгортати додаток заради усунення помилок попередньої версії. [42]

Розробка гібридного додатку виглядає перспективною ще й тому, що вона має на увазі створення його відразу під декілька платформ. Як наслідок, це позбавляє необхідності окремої розробки програми під кожну ОС та під кожен пристрій.

Крім усього іншого, потрібно взяти до уваги те, що якість і можливості гібридних додатків залежать, перш за все, від фреймворка, який використовує розробник. Також варто приділити належну увагу факторам, які роблять гібридні програми більш бажаним варіантом на тлі інших.

Отже, варто розробляти гібридний застосунок, якщо:

- є необхідність заощадити в бюджет;
- потрібно створити відносно нескладний додаток з простою анімацією;

- є завдання оперативної розробки та підтримки програми як мінімум на 2 платформи.

- пристрій, на якому буде розміщений застосунок не може бути доступний під час розробки та тестування;

Переваги гібридних застосунків:

- вартість і швидкість розробки;

- невелика кількість розробників;

- багатоплатформність;

- опція автоматичного оновлення.

Недоліки гібридних застосунків:

- некоректна робота при відсутності інтернет-з'єднання;

- середня швидкість роботи на тлі нативних;

- мінімалізм щодо візуальних елементів. [36]

4.2 Вибір середовища для розробки гібридних застосунків

1) Eclipse - це безкоштовнt середовище розробки від некомерційної організації Eclipse Foundation. Сама програма - це основа, до якої підключаються різні модулі. Наприклад, Java Development Tools (Для створення додатків на Java), C / C ++ Development Tools (для розробки програм на мові C або C ++) і т. д. Завдяки активному розвитку, а також підтримки з боку компанії і сторонніх розробників, на даний момент можна виділити наступні переваги середовища Eclipse:

- Відмінна продуктивність на слабких машинах;
- Велике число доповнень (наприклад, для роботи з сервером, базою даних і т. д.);
- Можливість підключення модулів;
- Можливість групової розробки.

Середовище розробки Eclipse мало велику популярність кілька років. тому і вважалася монополістом на ринку IDE для Android. Однак у зв'язку з виходом Android Studio, компанія Google перестала підтримувати Eclipse як основну середовище для розробки додатків під Android.

2) IntelliJ IDEA. Розробкою даного середовища програмування займається компанія JetBrains. Як і Eclipse, це середовище розробки дає можливість створювати програми на декількох мовах програмування. Головний недолік - це наявність платної версії.

3) Xcode - швидкий редактор, укомплектований повним набором інструментів для розробки під iOS, macOS і ін. Викачується з App Store безкоштовно. Особливості:

IB (Interface Builder) – додаток з набором інструментів для розробки графічних інтерфейсів, інтегроване в Xcode. Процес роботи нагадує створення дизайну «на полотні». Всю верстку можна зробити в IB, а потім зв'язати візуальні елементи з файлом реалізації, в якому описана вся логіка взаємодії з ними. Плюсом Interface Builder є наочна верстка, настроювання стилів, бекграунду, шрифтів і т. Д., А мінусом буде, наприклад, робота з анімацією, її можна виконати тільки кодом;

Симулятор; надає багатий набір пристроїв, на яких можна запустити і протестувати свій додаток; [37]

Відладник; вміє розбирати візуальну частину поелементно для пошуку помилок у верстці, також допоможе відловити баг і розібратися з проблемою витоку пам'яті. Також відладчик може виконувати всі ці операції і UI-тести в «бездротовому» режимі;

Вбудована система контролю версій GitHub; крім стандартних функцій розгалуження, в режимі розділеного екрану дозволяє переглядати зміни в різних «гілках»;

Функція імітації геолокації; незамінна при роботі з картою. Має «зашитий» набір міст, список яких можна доповнити;

Містить власний Playground, так звану «пісочницю», яка дозволяє швидко перевірити новий алгоритм або графічну рутину (наприклад, кілька рядків коду), не створюючи цілий додаток; [38]

Може збирати додаток відразу на визначений пристрій з визначеною OS;

Нестабільний; в процесі роботи може мимоволі закриватися з помилкою. Проект ніяк не постраждає, і останні зміни коду не пропадуть.

Часто не працює автодоповнення (функція доповнення тексту по введеної частини).

4) Visual Studio Code це безкоштовний редактор коду від Microsoft. Він має відкритий вихідний код і легкий для установки. VSCode відмінно підходить для JavaScript-розробників, оскільки має хороший набір функціоналу «з коробки», без необхідності встановлювати додаткові плагіни. Але цей редактор популярний не тільки серед розробників-початківців. Він може стати ідеальним вибором для більш просунутих користувачів, яким потрібно просто швидко приступити до роботи, не витрачаючи зайвий час на налаштування.

Унікальна особливість VSCode - можливість використовувати його в браузері. Таким чином ви можете користуватися редактором на планшеті і при цьому мати таке ж середовище, до якого звикли в десктопній версії. Щоб цей функціонал запрацював, потрібно ще налаштувати code-server в мережі, до якої ви маєте доступ, але налаштовувати систему доведеться тільки один раз, і це того варте.

Git вбудований в редактор, але інтеграція не така надійна, як в деяких інших редакторах. Наприклад, користувачі WebStorm воліють застосовувати push / merge, а не функціонал, що пропонує VSCode.

При необхідності ви можете встановити безліч додаткових плагінів як розширень. Є сніппети коду, доповлення для відладки, аналізу коду, рендеру верстки тощо. [39]

Будь-які складнощі користування VSCode не є проблемою. У цього редактора велике співтовариство користувачів, які публікують вирішення ситуації, у які вони потрапляли, дають відповіді на поставлені питання, спільними зусиллями вирішують проблеми.

5) Atom - ще один безкоштовний редактор коду, який був розроблений GitHub. Це, власне, спеціалізована версія браузера chromium, конвертована в редактор коду. Під капотом він для підтримки плагінів використовує Node.js.

Редактор Atom «з коробки» не такий потужний, але доступна велику кількість плагінів, завдяки яким можна отримати будь-який бажаний функціонал. Для створення комфортного середовища розробки, доведеться зібрати великий особистий набір плагінів. Для ефективної роботи з JavaScript рекомендується спористатись наступними плагінами: atom-typescript, file-icons, atom-beautify,linter.

За допомогою плагіна teletype над одним файлом можуть працювати кілька людей одночасно, причому у кожного буде кілька курсорів на екрані. Цей функціонал можна використовувати для наставництва, парного програмування або просто спільної роботи. Це одна з тих особливостей, які виділяють Atom на загальному тлі.

Інтеграція git в Atom реалізована добре, оскільки обидва продукти розроблені однією компанією. Також може стати в нагоді плагін git-plus, що дозволяє

запускати команди за допомогою скорочень клавіш, без використання терміналу git.

Atom настільки гнучко налаштовується, що є можливість редагувати файл less, щоб підібрати гаму кольорів для застосунку. Це істотний плюс для дизайнерів.

Скрипт coffe, що запускається при завантаженні, дозволяє швидко змінити поведінку редактора. Також ви можете писати власні плагіни на JavaScript - це важливо на випадок, якщо велика команда розробників користується спільною базою сніпетів.

Робота з написання коду в редакторі Atom проходить плавно. Цей досвід можна зробити ще кращим за допомогою плагінів, наприклад, minimap. Але щоб встановити і налаштувати весь функціонал певним чином, може піти деякий час. Втім, у такого підходу є і перевага: в Atom не буде непотрібних вам речей, які сповільнювали би завантаження і роботу редактора. [39]

Що стосується уповільнення – можна відчути деяку повільність редактора при завантаженні великих файлів або при перемиканні між вкладками.

6) Sublime Text - це платний редактор з безкоштовним пробним періодом. У ньому немає надмірної кількості встановлених плагінів. Але комфортне середовище розробки організувати не складно. Деякі пакети, наприклад SideBarEnhancements (дозволяє перейменовувати, переміщати, копіювати і вставляти файли і папки), повинні бути вбудовані в редактор, але ви лише можете їх завантажити і встановити додатково.

Як і у випадку з Atom, на первинне налаштування цього редактора може знадобитися якийсь час. Sublime Text це легкий редактор, завдяки чому він дуже швидкий і може працювати з великими проектами і об'ємними файлами.

Цікава функція Sublime Text «goto anything», яка може застосовуватися для швидкого переходу до файлів, символів, номерів рядків. Подібний функціонал в

тій чи іншій формі є в більшості редакторів, але тут можна комбінувати різні варіанти, складаючи довші запити, наприклад «fileName @ functionName».

Виділення змінної поширюється на всі екземпляри цієї змінної, а її перейменування означає перейменування всіх змінних з цим ім'ям. Потреба в такому функціонал виникає часто, тому його наявність покращує досвід використання редактора.

4.3 Розробка застосунку для візуалізації інформації на складі

Реалізована система складається зі спеціалізованих сервісів різного типу: веб-сервіси слухачі, веб-сервіси з інтерфейсом користувача, шини даних, десктопні застосунки (Рисунок 4.1). Розглянемо діаграму головних компонентів:

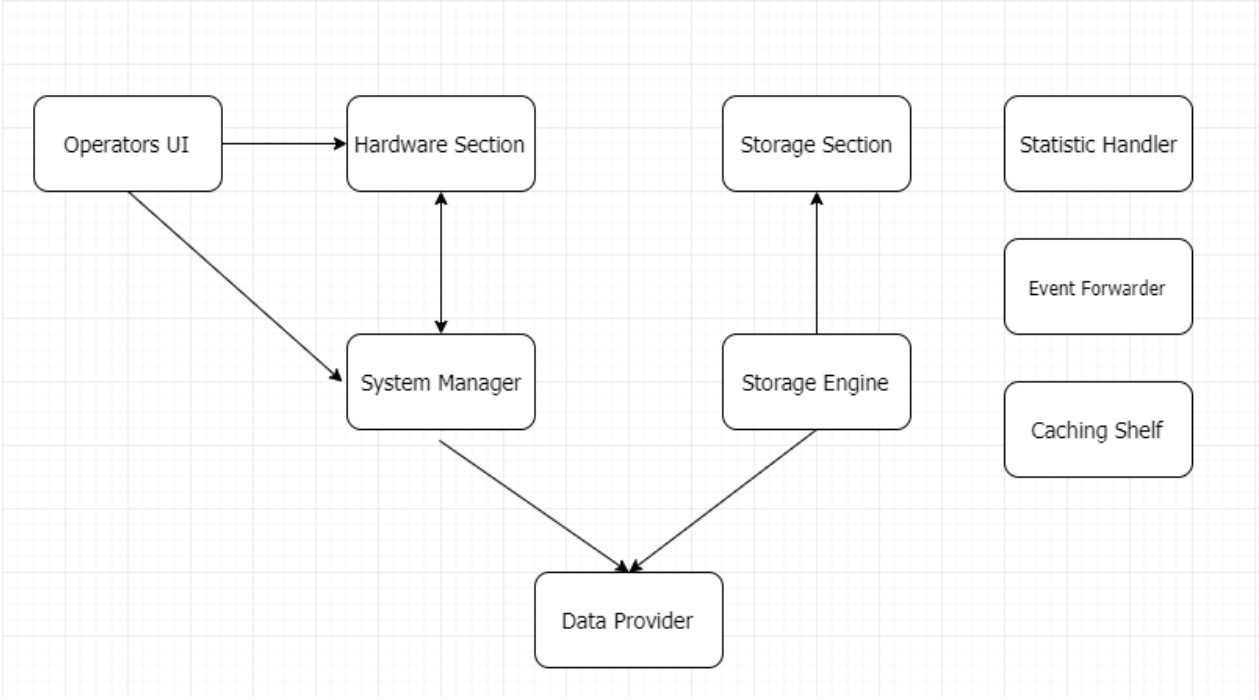

Рисунок 4.1 - Архітектура системи

Data Provider (постачальник данних) - єдиний комронент, який має доступ до бази даних. Містить публічний інтерфейс взаємодії, який доступний для System

Manager та Storage Engine. Безпосередньо зв'язок бази даних та бекенду відбувається через ORM систему Entity Framework Core. Завдяки цій системі, база даних генерувалась при створенні об’єктної моделі програми, а не навпаки (Code First). Такий підхід дозволяв легко мігрувати до нової схеми бази даних, якщо об’єктна модель зазнавала змін під час розробки. Ще однією особливістю підходу є фактична втрата необхідності у Stored Procedures. Пов’язано це з тим, що тепер логічно база даних не є цінністю, а просто відображенням об’єктної моделі. Тому вважається, що краще тримати Stored Procedures у вигляді методів чи функції на об’єктній стороні, а не на стороні даних.

Не зважаючи на всі переваги Entity Framework Core, програма не повина напряму залежати від ORM системи. (Принцип Dependency Inversion.) Саме тому вона обгорнута інтерфейсом, яким безпосередньо користується бекенд. Це реалізація патерну Repository.

Основною перевагою цього патерну є відсутність залежностей між об’єктною моделлю, та моделлю бази даних. Обмін повідомленнями відбувається через високорівневий інтерфейс, що дозволяє не лише підмінювати джерело даних без потреби внесення змін в об’єктну модель, а й підмінювати ORM систему вцілому. Тісно з патерном Repository пов’язаний також патерн Unit Of Work.

System Manager (менеджер системи) - головний компонент логістики. Відповідає за всі внутрішні процеси системи, стан Hardware та Storage секцій, підтримує месседжинг з ними. Постачається у вигляді набору WCF та HTTP сервісів. Реалізований на платформі ASP.NET Core.

Комбінуючи функціонал існуючих технологій .NET з розробки розподілених застосунків (ASP.NET XML Web Services — ASMX, WSE 3.0, .NET Remoting, .NET Enterprise Services і System.Messaging), WCF надає єдину платформу розробки, яка підіймає рівень продуктивності і зменшує витрати на розробку та підтримку веб-сервісів. Принципи інтероперабельності, що лежать в основі цієї

технології, дозволяють легко контролювати взаємодію з іншими платформами, для яких використовуються технології взаємодії платформ, наприклад WSIT які розробляються на основі відкритих джерел коду.

У 2015 році Microsoft оголосила про публікацію вихідного коду бібліотек Windows Communication Foundation на GitHub. Код відкритий під ліцензією MIT. Варто зазначити, що відкритим став не повний набір WCF бібліотек для десктопу, а лише його частина, спрямована на взаємодію між сервісами для мобільних і серверних систем.

Наприклад, відкрились бібліотеки System.ServiceModel.Primitives.Provides, System.ServiceModel.Http, System.ServiceModel.NetTcp та інші. Для порівняння, всього в наборі WCF нараховується більше 40 компонентів

Hardware Section (залізна секція) - набір роботів-модулів, які наділені певними властивостями та механізмами, впорядкована комбінація яких створює автоматизовані конвеєрні постачальники товарів на Storage Section та у зворотньому напрямку. Більшість модулів комунікує із System Manager, деякі не мають комунікації взагалі. Приклади модулів:

- Сканер - зчитує штрих-код з коробок;
- Вимірювач - вимірює розміри коробки і передає дані System Manager’у;
- Зважувач - зважує коробку і передає дані System Manager’у;
- Конвеєр - перевозить коробки між іншими модулями;
- Сингулятор - розбирає палету на окрему коробки;
- Палетайзер - будує палету з коробок;
- Обгортувач - готує палету до транспортації;
- Ліфтова станція - місце, де коробки підіймаються ліфтами;
- Ліфт - модуль для перевезення коробок між поверхами Storage.

Окреме питання стосується емуляції Hardware Section. Вона необхідна з метою тестування та реалізації нових компонентів чи функцій системи. Таким чином, кожен модуль та кожна впорядкована система модулів має свою проксі для емуляції меседжингу із System Manager. Це реалізується шаблоном проектування “Заступник” або ж “Проскі”.

Для того, щоб отримувати доступ до громіздкого об’єкта, з метою зменшення затрачених ресурсів можна створити сурогат цього об'єкта. «Заступник» інкапсулює посилання на реальний об'єкт, яке дозволяє заступникові керувати запитами до цього об' єкту(для того, щоб об'єкт класу «Заступник» мав можливість звертатися до об'єкта класу «Суб'єкт», їх інтерфейси повинні бути однакові). Оскільки інтерфейс «Суб'єкта» ідентичний інтерфейсу «Заступника», так, що «Заступника» можна на підставити замість «Суб'єкта», додавши необхідний рівень абстракції, контроль доступу до «Суб'єкта», може відповідати за створення або модифікування «Суб'єкта». «Суб'єкт» визначає для «Заступника» свій інтерфейс, таким чином, що «Заступник» може бути використаний будт-де, де очікується сам «Суб'єкт».

«Заступник» може бути відповідальний за:

- шифрування запиту чи його аргументів і передачу обробленого запиту реальному «Суб'єктові»;
- кешування додаткової інформації про реального «Суб'єкта», щоб самостійно визначити момент його створення;
- перевірка прав, необхідих для виконання отриманого запиту.

Operators UI (інтерфейси користувача для Операторів) - набір десктопних застосунків, які виконуються на терміналах в окремих місцях складу. Основне завдання - комунікація із System Manager або Hardware Section. Для прикладу, завдання завести палету до Storage може ініціюватись або на етапі сканування

палети (Сканер модуль) або вручну оператором за допомогою Unpalletize UI. Усі інтерфейси реалізовані за допомогою технології WPF та патерну MVVM.

Model-View-ViewModel — це патерн проектування, який використовується під час проектування архітектури десктопних застосунків. Публічно він вперше був продемонстрований у 2005 році Джоном Госсманом. Це було позиціоновано, як модифікація шаблону Presentation Model. MVVM готовий до впровадження на таких платформи розробки, як Windows Presentation Foundation, Silverlight тощо.

MVVM розмежовує розробку графічного інтерфейсу від розробки бізнес логіки. Бекенд позначається як модель, повністю автономна від свого представлення. Модель представлення є сполучною частиною, яка відповідає за подання даних для їх подальшої візуалізації і надання користувачу. Таким чином, модель представлення за своїми обов’язками більше схожа на модель, ніж на представлення і відповідає в більшості за логіку відображення даних. Модель представлення також часто реалізовує патерн «Посередник», організовуючи доступ до моделі, забузпечуючи виконання певних правил використання, які необхідні для представлення.

MVVM часто використовують замість класичного підходу MVC та йому подібних тоді, коли на платформі, де проводиться розробка, необхідним є «зв'язування даних».

В MVC та MVP зміни на інтерфейсі користувача не повинні впливати безпосередньо на модель, а обов’язково повинні пройти шлях через Контролер/Presenter. У таких технологіях, як WPF та Silverlight, використовується концепція «зв'язування даних», яка дозволяє пов’язувати дані моделі із візуальними елементами для підтримки оновлень в обидві сторони. [30]

Архітектура MVVM вирішує проблему зв’язування даних чітким поділом відповідальностей:

– створення інтерфейсу користувача здійснюється дизайнером інтерфейсів на базі технології, яка є природною для такої роботи (XML);
– поведінка та зв’язок між відображенням інтерфейсу користувача і бізнес даними реалізується розробником як ViewModel компонент;
– функціональні обов’язки між інтерфейсом користувача та ViewModel впроваджуються через біндинги, які, по своїй суті, є правилами пов’язування візуальних контролів із функціями контролера. Біндинги можуть бути створені програмно в коді або визначені дизайнером інтерфейсів декларативним шляхом.

Архітектура MVVM використовується в тому чи іншому вигляді багатьма сучасними технологіями, наприклад Microsoft WPF і Silverlight, Oracle JavaFX, Adobe Flex, AJAX. (Рисунок 4.2)

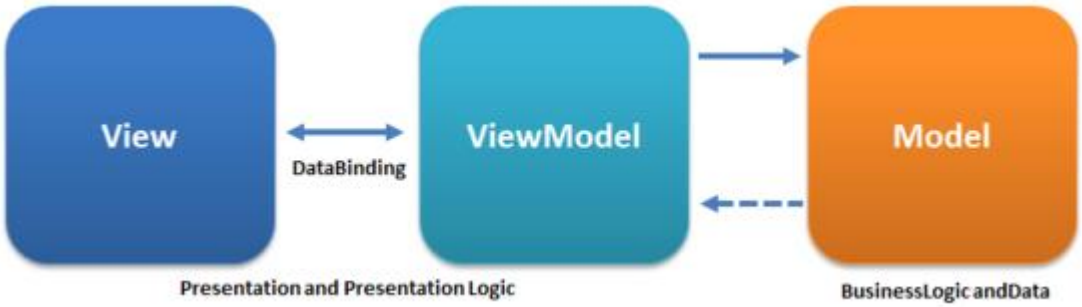

Рисунок 4.2 - Шаблон MVVM

Storage Section (секція зберігання) - веб-сервіс, який складається з багатьох сервісів, що керують наступними фізичними компонентами: роботами, які перевозять коробки на поверсі, станціями для зарядки та координування роботів, даними про кожну комірку на полицях всіх поверхів секції, ліфтами. Storage Section тісно взаємодіє із Data Provider та System Manager для вирішення

головного свого завдання - взаємодії логістики складу із його фізичними компонентами.

Statistic Handler (оброблювач статистики) - веб-сервіс без користувацького інтерфейсу, який пасивно слухає меседжинг між компонентами системи та вираховує деякі статистичні метрики. Раз в певний період часу генерується звіт, який подається у вигляді графіків та діаграм відповідному підрозділу менеджменту. Пасивна модель прослуховування події від різних компонентів системи реалізована за допомогою реактивного програмування. (Рисунок 4.3)

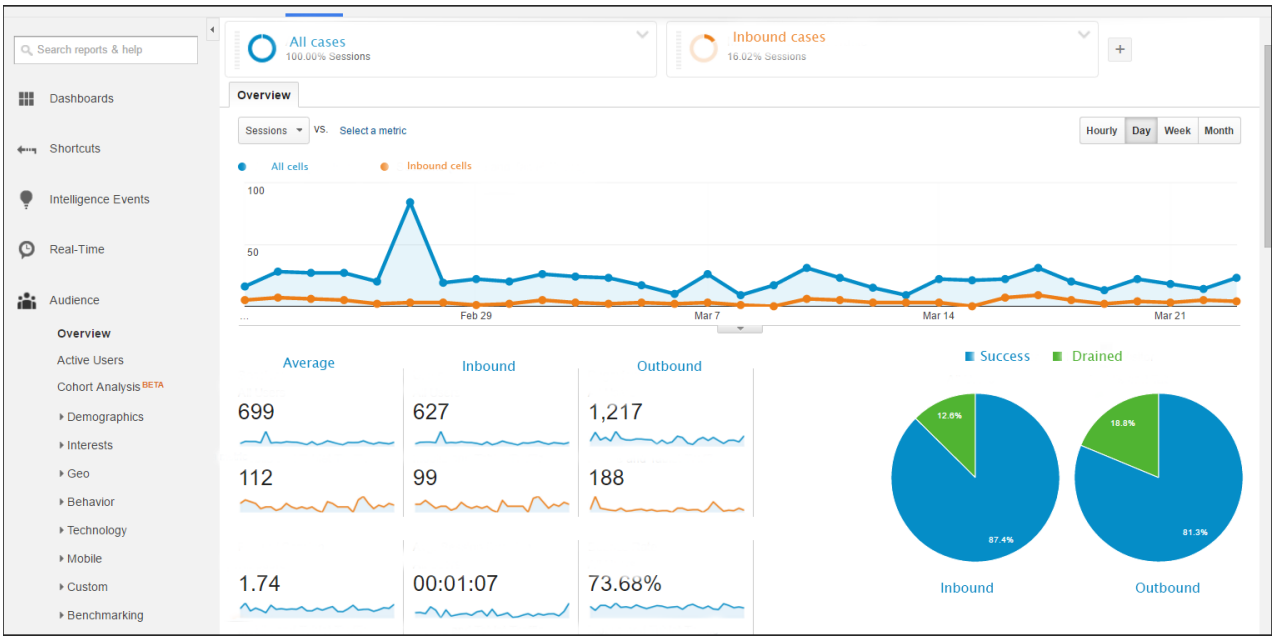

Рисунок 4.3 – Статистичний аналіз

Реактивне програмування — це парадигма програмування, побудована моделювання даних та операцій у формі потоків. Це означає, що в момент програмування має бути можливість легко визначаити динамічні або ж статичні потоки даних, а модель виконання буде автоматично представляти будь-які зміни у форматі потоку даних.

Наприклад, в традиційному програмуванні вираз, $a:=b+c$ означає, що a набуває результату виконання $b+c$ тільки під час обчислення виразу, а якщо у майбутньому значення b і c зміняться, то це не матиме ніякого впливу на вже обчислене та закешоване значення a.

Натомість, в реактивному програмуванні значення a буде автоматично оновлено за новими значеннями, що є протилежністю функціонального програмування.

Прикладом реактивного програмування є електрона таблиця. Комірки цієї таблиці найчастіше мають значення, які представляються у вигляді правила чи формули. Ця формула обчислюються відповідно до значень інших пов'язаних з нею комірок. Якщо значення іншої пов'язаної комірки зміниться, то значення формули оновлюється автоматично.

Ще один приклад — це мова опису апаратури типу Verilog. Реактивне програмування дозволяє моделювати зміни прямо у момент їх оновлення та розповсюдження по електричному ланцюгу.

На початку свого розвитку реактивне програмування подаваолсь як засіб створення користувацьких інтерфейсів, потоки найчастіше використовувались для впровадження анімацій у системах реального часу. І вже згодом це стало загальною парадигмою програмування і почало використовуватись у ріхних технологіях розробки. Наприклад, у патерні MVC, реактивне програмування дозволяє автоматично відображати зміни моделей у їх представленні та навпаки.

Event Forwarder (передавач подій) – компонент, що служить для слабкої зв'язаності системи. Він делегує події, які піднімає будь-який компонент системи на той сервіс чи застосунок, який повинен їх обробити. При цьому на Event Forwarder не накладається ніяка додаткова логіка, навіть валідаційна. Реалізація такого посередника полегшує меседжинг між компонентами, що створені за допомогою різних технології та спілкуються за допомогою різних протоколів.

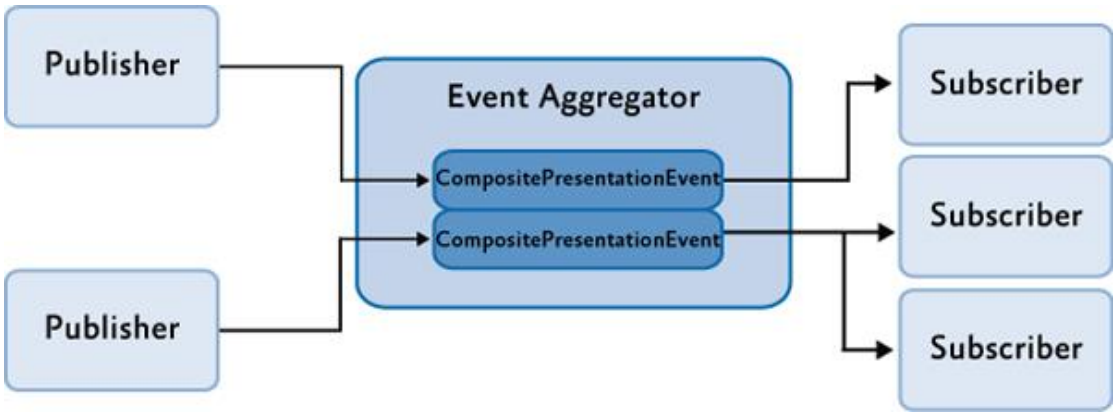

Рисунок 4.4 - Шаблон «Посередник»

Патерн «Посередник» замінює взаємодію "всі з усіма" взаємодією "один з усіма". Розбиття системи на безліч об'єктів в загальному випадку підвищує ступінь повторного використання, проте безліч взаємозв'язків між цими об'єктами, як правило, призводять до зворотного ефекту. Щоб цього не допустити, взаємодію між об'єктами інкапсулюють в об'єкт-посередник. Діючи як центр зв'язку, цей об'єкт-посередник контролює і координує взаємодію групи об'єктів. При цьому об'єкт-посередник робить взаємодіючі об'єкти слабо пов'язаними, так як їм більше не потрібно зберігати посилання один на одного - все взаємодія йде через цього посередника. (Рисунок 4.4)

Caching Shelf (полиця кешу) – Redis кластер для кешування певних даних обміну. Головна задача – не навантажувати Data Provider функціями, що послідовно анулюють одна одну. На приклад, якщо дані потрібно помістити в таблицю, а потім після одного чи двох читань одразу видалити, зручніше та ефективніше буде просто закешувати ці дані.

Redis (від англ. Remote dictionary server) - резидентна система управління базами даних класу NoSQL з відкритим вихідним кодом, що працює зі структурами даних типу «ключ - значення». Використовується як для баз даних, так і для реалізації кешів, брокерів повідомлень. Кешування даних відбувається в

оперативній пам’яті, зберігання даних на диск вважається окремою додатковою функцією. Саму цьому Redis використовується як тимчасове а не як основне сховище даних системи.

Орієнтована на досягнення максимальної продуктивності на атомарних операціях (заявляється про приблизно 100 тис. SET- і GET-запитів на Linux-сервері початкового рівня). Написана на C, інтерфейси доступу створені для більшості основних мов програмування.

4.4 Огляд користувацького інтерфейсу

Оскільки Operators UI – єдиний продукт кінцевого користувача є користувацьким інтерфейсом, розглянемо його основні можливості. Застосунки реалізовані за допомогою технології WPF та виконуються на вбудованих терміналах на спеціальних станціях на складі, призначених для операторів.

Окрім того, операційна система Windows надає достатніх ресурсів для розробки, щоб тестування таких застосунків було можливе без використання спеціальних терміналів. В такому випидку ці інтерфейси запускаються як звичайні десктопні застосунки Windows.

Саме рішення створення гібридного застосунку дозволяє розробнику мати повний функціонал інтегрованого середовища розробки без необхідності користуватись фізичним пристроєм, на якому пізніше буде встановлене це заберпечення. Відладка та тестування може бути виконаним на будь-якому десктопі з симуляцією поведінки реального терміналу оператора на складі.

Також цей застосунок підлягає тестуванню через фреймворки імітації дій користувача на UI елементах. Стає доступною емуляція натискання кнопок та вводу даних оператором складу.

Таким чином вони забезпечують комунікацію між операторами та програмною системою складу. Винятком є палетизація палети із окремих коробок

різного типу. В такому випадку повідомлення щодо кожної коробки відправляється безпосередньо до меседжингових компонентів Hardware Section.

Базова реазілація включає в себе 4 сторінки. Order In сторінка містить таблицю із статусами та прогресом виконання всіх запитів. Інші 3 сторінки відображають 3 відповідних запити в систему

– Item Master Recon
– Add Order
– Replenishment Confirmation

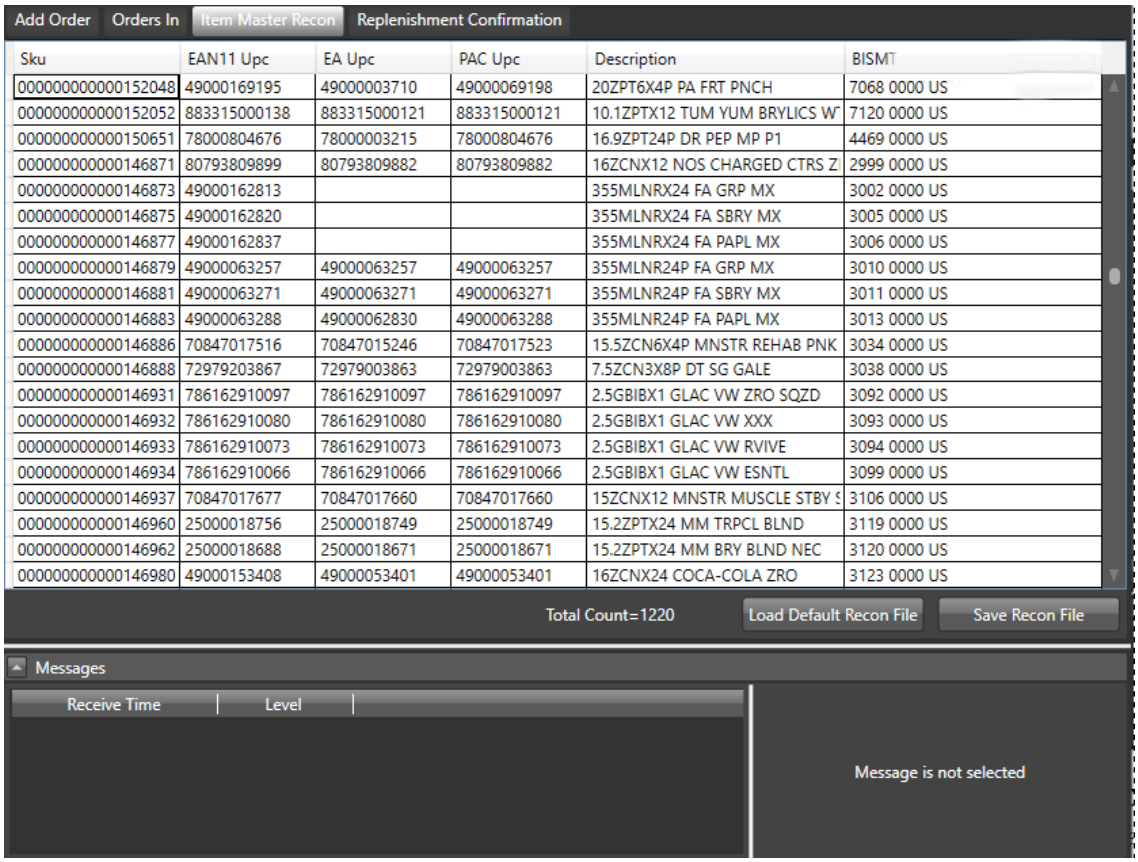

Рисунок 4.5 - Item master recon

Item Master Recon – запит на додання нового типу товарів у базу складу (SKU). Під час додавання товару оператор повинен визначити нове унікальне значення SKU, додати його опис, вказати допустимий діапазон розмірів та ваги

товару разом із упаковкою та без неї, тип пакування, термін придатності продукту, належність його до алергенів тощо. Після першого успішного введення коробки з товаром нового типу цей тип знає «вивченим» і система додає його в реєстр типів товарів. (Рисунок 4.5)

Також існує можливіть поширювати знання про товари, використовуючи відкриті дані з деяких відомих баз знань. Таким чином, можна «вивчити» відомі ринку типи товарів ще до того, як вони фізично потраплять на склад.

З часом товари втрачають свою актуальність, тому ті з них, які надходять на склад рідше зі ішні, не будуть більше вважатись «вивченими».

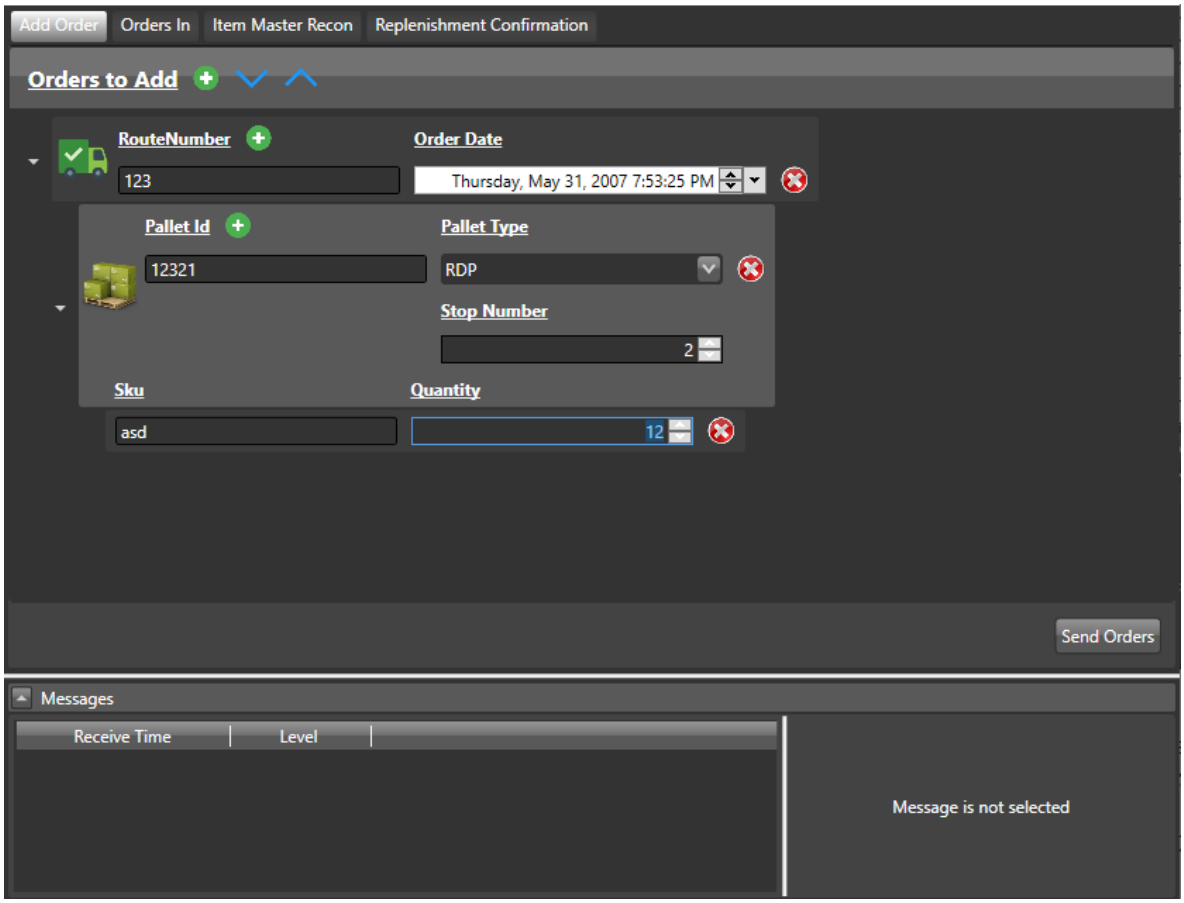

Рисунок 4.6 - Add Order

Add Order – запит на додавання в базу складу нової палети для її вводу\виводу. Палеті присвоюється ідентифікатор (LPN), вказуються типи товарів, які належатимуть палеті та їх кількість, загальна вага палети, ставиться дата.

Палета може містити як товари одного типу, так і товари різних типів. При цьому необхідно, щоб кожен шар палети складався із товарів одного і того ж продукту. В такому випадку, кожна коробка повинна мати власний ідентифікатор, якому належатиме відповідний тип продукту (SKU). Якщо ж тип продукту не є «вивченим» (див. попередній запит), тоді система не розпізнає палету, і ввід чи вівід такої коробки стане не можливим. (Рисунок 4.6)

Окрім цього, оператор може взяти одну картонку з палети, зробити її «вивченою», і тоді введення всієї полети з коборками такого продутку стане можливим.

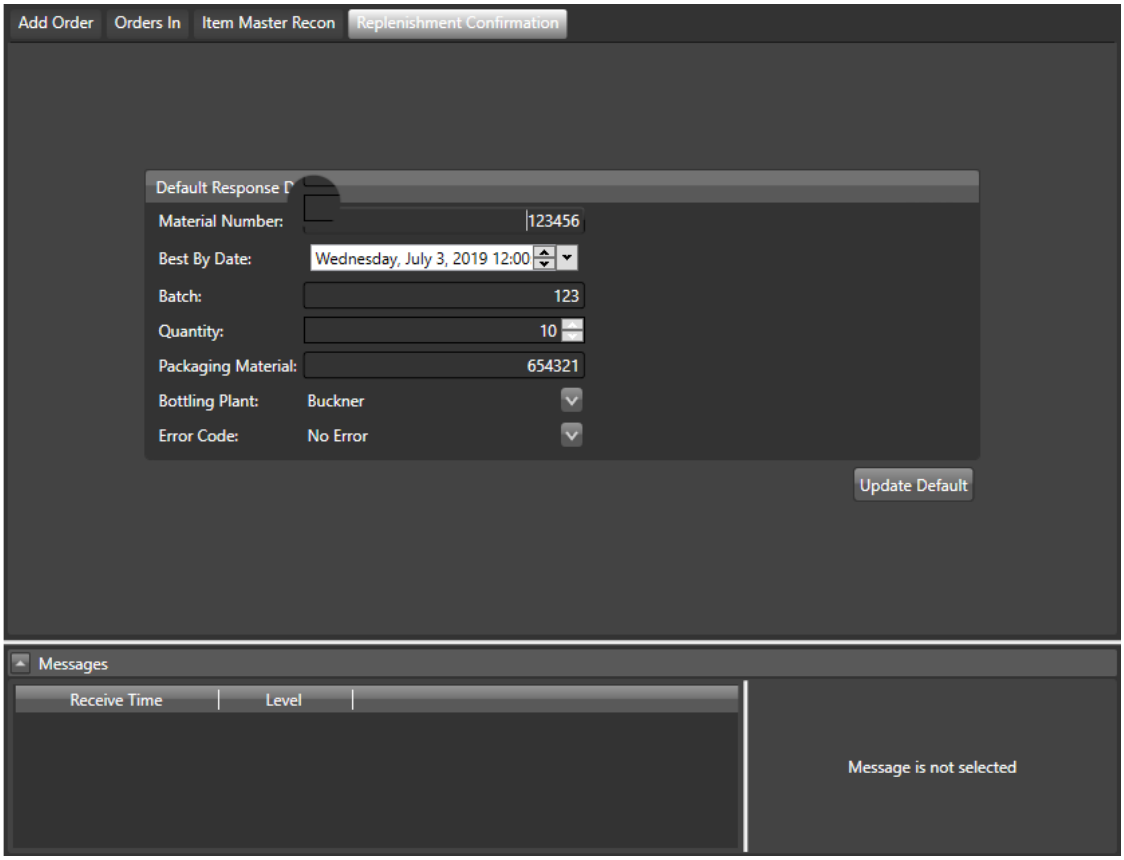

Рисунок 4.7 - Replenishment Confirmation

Replenishment Confirmation – запит на вивезення товару зі складу (Рисунок 4.7). При цьому необхідно зазначити наступні поля:

- Тип продукту (SKU)
- Мінімальну дату початку вивезення
- Кількість коробок з продуктом
- Номер палети

Як тільки настане допустимий час для вивезення коробок, System Manager одразу відправить набір небхідних команд, які ініціюють цей процес.

Висновки до розділу 4

Було проаналізовано існуючі підходи на методики розробки гібридних застосунків. При цьому були враховані особливості автоматизованого складу, які визначають спосіб встановлення застосунків на ньому.

Створені компоненти для візуалізації процесів на складі мають такий пулбічний інтерфейс, який може бути впровадженим у архітектуру програмної частини складу.

Для реалізації застосунку використовувався наступний стек технологій:
.Net Core, WPF, Angular, Rabbit MQ, Redis.

ВИСНОВКИ

В ході виконання даної магістерської роботи було підвищено ефективність пропускної здатності автоматизованого складу. Процес візуального аналізу коробок, що входять на склад було покращено на основі досліджуваних методів комп ’ютерного бачення. Також було розроблено систему візуалізації інформації для операторів автоматизованого складу.

Впровадження результатів цього дослідження збільшує швидкість обробки коробок автоматизованим складом, а також потенційно кількість ресурсів для його обслуговування.

Було створено сервіз аналізу та класифікації візуальних зображень – наліпок на коробках, що вводяться в склад. Для цього були проаналізовані існуючі технології та методології аналізу зображень. В результаті аналіз відбувався на основі теоріх нечіткої логіки. Пов ’язано це з тим, що особливість даного автоматизованого складу вказує на те, що кількість вихідних даних, приналежність до яких мав показати аналіз є скінченою

Було визначено, що кластеризація даних методом Варда підходить для того, аби результати аналізу зображень поділити на класи, які відповідають групам наліпок на коробках. Таке рішення допомагає з помітно вищою точністю визначити конкретну наліпку на конкретній коробці, навіть, якщо вона була пошкоджена чи прикріплена з похибкою в орієнтуванні.

Було запропоновано розришення до існуючої архітектури програмного забезпечення складу з метою додавання застосунку для візуалізації інформації для оператора складу. Слід зазначити, що особливістю автоматизованого складу є те, що будь-які застосунки із користувацьким інтерфейсом будуть відображатись на вбудованих терміналах, що робить розробку таких застосунків особливою. Для підвищення ефективності тестування, відладки та встановлення таких застосунків

було проаналізовано існуючий стек технологій та середовищ розробки для створення гібридних застосунків.

Було реалізовано систему візуалізації інформації для операторів складу. Застосунок був побудований на платформах .Net Core та WPF з використанням додаткових технологій для меседжингу, деплойменту тощо. Кросплатформеність та зручне середовище розробки такого рішення знищує різницю між розробкою застосунку для десктопу та розробку застосунку для терміналу, вбудованого у певні станції автоматизованого складу. Система відображає процеси, які відбуваються на складі, а також дає можливість створювати, зупиняти та змінювати такі процеси.

Результатом дослідження є створені компоненти аналізу візуальних даних на та візуалізації інформації на складі, які готові до інтергування із існуючою програмною системою складу.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Taborovskyi A., Kolesnikova K., Khlevnyi A. The impact of automated and robotic warehouses on the scope of supply chain process. //Information Technology and Interactions (Satellite): Conference Proceedings, December 04, 2020, Kyiv, Ukraine / Taras Shevchenko National University of Kyiv and [etc]; Vitaliy Snytyuk (Editor). Kyiv: Stylos, 2020. P. 172 – 174
2. Белінський П. І. Менеджмент виробництва та операцій: підручник. Київ, 2005. 380 с.
3. Л. А. Киржнер. Менеджмент организаций: Учебное пособие. КНТ: 2006. 240 с.
4. Бауерсокс Дональд Дж. Логістика. Інтегрований ланцюг: Москва, Олимп-Бізнес, 2017. - 500 с.
5. Бауерсокс Доналд Дж. Тара і упаковка: Випуск 3: Москва, Торговий Дом Металів, 2009. 112 с.
6. В.В. Дибска. Управління складами в ланцюгах постачання: моногр. Альфа-прес, 2014. 486 с.
7. В.М. Курганов. Логістика. Транспорт і склад в ланцюгу поставок товаров: Книжковий світ, 2004. 432 с.
8. Владислав Волгін. Погрузка и розгрузка. Довідник груз-менеджера: Москва, АВТОР, 2006. 233 с.
9. Джуліан Дент. Як організувати ланцюг поставок: Юнайтед Прес, 2008. 208 с.
10. Джуліан Дент. Все про дистрибуцію: Аквамаринова Книга, 2011. 360 с.
11. Н.С. Киреєва. Складське господарство: Academia, 2009. 192 с.
12. Р.Н. Паршина. Логістика контейнерних перевозок: Київ, 2008. 420 с.
13. Introduction To Machine Learning by Nils J Nilsson – 1997, 209 p.
14. Inductive Logic Programming: Theory and Methods by Stephen Muggleton, Luc de Raedt - ScienceDirect 1994, 51 p.

15. Information Theory, Inference, and Learning Algorithms by David J. C. MacKay - Cambridge University Press 2003, 640 p.
16. Gaussian Processes for Machine Learning by Carl E. Rasmussen, Christopher K. I. Williams - The MIT Press 2005, 266 p.
17. Задача класифікації: веб-сайт. URL: uk.wikipedia.org/wiki/Задача_класифікації. (дата звернення 12.03.2021).
18. Practical Data Analysis, 2nd Edition by Hector Cuesta, Sampath Kumar - Packt Publishing 2016, 338 p.
19. Python Data Analysis Cookbook by Ivan Idris – Packt Publishing 2016, 462 p.
20. Mastering Python Data Analysis by Magnus Vilhelm Persson, Luiz Felipe Martins - Packt Publishing 2016, 284 p.
21. Practical Machine Learning by Sunila Gollapudi – Packt Publishing 2016, 468 p.
22. Mastering Python Data Analysis by Magnus Vilhelm Persson, Luiz Felipe Martins - Packt Publishing 2016, 284 p.
23. Shi, J., Lei, Y., Zhou, Y. A narrow band interval type-2 fuzzy approach for image segmentation: Journal of Systems Architecture, v. 64, 2018. 86—99p.
24. Murugeswari, P., Manimegalai, D. Noise Reduction in Color image using Interval Type-2 Fuzzy Filter (IT2FF): International Journal of Engineering Science and Technology, 2011. 1334—1338p.
25. Beimer, W.; Maennig, W. Noise effects and real estate prices: A simultaneous analysis of different noise sources. Transp. Res. Part D 2017, 54, 282–86.
26. Holland, J. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence; University of Michigan Press: Ann Arbor, MI, USA, 1975; pp. 439–444.
27. Inductive Logic Programming: Theory and Methods by Stephen Muggleton, Luc de Raedt - ScienceDirect 1994, 51 p.

28. Bayesian Reasoning and Machine Learning by David Barber – Cambridge University Press 2011, 644 p.
29. S. Salvador and P. Chan, "FastDTW: Toward Accurate Dynamic Time Warping in Linear Time Space", in Intelligent Data Analysis, vol. 11, no. 5: IOS Press, 2007, pp. 561-580.
30. A. Schmidt and K. Van Laerhoven, “How to build smart appliances?” IEEE personal Communications, vol. 8, no. 4, pp. 66–71, 2001.
31. Big Data Visualization by James D. Miller - Packt Publishing 2017, 304 p.
32. The Elements of Statistical Learning: Data Mining, Inference, and Prediction by T. Hastie, R. Tibshirani, J. Friedman - Springer 2009, 764 p.
33. Yuksel, M., Basturk, A. Application of Type-2 Fuzzy Logic Filtering to Reduce Noise in Color Images: IEEE Computer Intelligence Magazine, 2012. 25—35p.
34. Own, C. M., Tsai, H. H., Yu P.T., Lee, Y.J. Adaptive type-2 fuzzy median filter design for removal of impuls noise: Imaging Scientific Journal, 2006. 3—18p.
35. Melin, P. Mendoza O., Castillo, O. An improved method for edge detection based on interval type-2 fuzzy logic: Expert Systems with Applications, 2010. 8527—8535 p.
36. Don Gosselin Microsoft Visual C++.NET: Москва: СПб : Санкт-Петербург, 1985. 385 с.
37. Duncan Mackenzie Microsoft Visual Basic .NET 2003 Kick Start: Москва Вогні, 1979. 488 с.
38. Elijah Lovejoy Elijah Lovejoy's ASP Training Course (With CD-ROM: Москва, Вища школа, 1984. 155 с.
39. Fritz Onion Essential ASP.NET With Examples in C#: Москва СПб, 2005. 390 с.
40. Kalani Kirk Hausman Developing and Implementing Windows-based Applications with Visual Basic .NET and Visual Studio .NET (+ CD-ROM): Que Certification, 2003. 178 с.

41. Kyle Lutes An Information Systems Approach to Object-Oriented Programming Using Microsoft Visual C#: Москва, Наука, 2005. 469 с.
42. Michael Amundsen ASP.NET for Developers / Michael Amundsen, Paul Litwin. – Москва, СИНТЕГ, 1989. 449 с.
43. Nick Symmonds Internationalization and Localization Using Microsoft .NET: Москва: Наука, 2000. - 113 с.
44. Venkat Subramaniam .NET Gotchas: Москва: СПб, 2005. - 396 с.
45. Маккі Алекс Введение в .NET 4.0 и Visual Studio 2010 для : Діалектика, Вільямс, 2010. - 488 с.
46. Zeldman Jeffery. Taking Your Talent to the Web: Making the Transition from Graphic Design to Web Design: New Riders, 2001. 323 с.
47. Офіційна документація Microsoft з ASP.Net Core: веб-сайт. URL: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.1 (дата звернення: 21 січня 2021 року)
48. Офіційна документація технології WPF (Windows Presentation Foundation): веб-сайт. URL: Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/framework/wpf/ (дата звернення: 2 лютого 2021 року)
49. Офіційний сайт уніфікованої архтектури OPC: веб-сайт. URL: https://opcfoundation.org/ (дата звернення: 17 лютого 2019 року)
50. Офіційний сайт Docker: веб-сайт. URL: https://www.docker.com/ (дата звернення: 21 березня 2021 року)
51. Огляд шаблону MVVM: веб-сайт. URL: https://www.tutorialspoint.com/mvvm/ (дата звернення: 26 березня 2021 року)

ДОДАТКИ

ДОДАТОК А

Реалізація патерну Repository для роботи із таблицею SKUs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using coldel.Persistance.Models;
using coldel.Persistance.Models.DTOS;
using coldel.Resources;
using Microsoft.EntityFrameworkCore;

namespace coldel.Persistance.Core
{
    public class ProductRepository : IProductRepository
    {
        private readonly ProductlDbContext _context;
        public ProductRepository(ProductlDbContext context)
        {
            _context = context ?? throw new
ArgumentNullException(nameof(context));
        }

        public void AddRegistation(Registration registration)
        {
            _context.Registrations.Add(registration);
            _context.SaveChanges();
        }

        public void DeleteRegistration(Guid id)
        {
            var registration = _context.Registrations.Find(id);
            _context.Registrations.Remove(registration);

            _context.SaveChanges();
        }

        public Client GetOrAddClient(string clientName, string phone)
        {
            Client client;
            client = _context.Clients.FirstOrDefault(cl => cl.Name == clientName
&& cl.Phone == phone);

            if (client != null)
```

```
            {
                return client;
            }

            client = new Client()
            {
                Id = new Guid(),
                Name = clientName,
                Phone = phone
            };

            _context.Add(client);
            _context.SaveChanges();

            return client;
        }

        public IEnumerable<RegistrationDTO> GetRegistrations()
        {
            return _context.Registrations.Include(reg => reg.Client)
                                         .Include(reg => reg.Phobe)
                                         .Select(reg => new RegistrationDTO(reg))
                                         .ToList();
        }
        public IEnumerable<RoomDTO> GetSKUs()
        {
            return _context.SKUs.Select(r => new SKUDTO(r)).ToList();
        }

        public void SaveChanges()
        {
             _context.SaveChanges();
        }

        public Registration GetRegistration(Guid id)
        {
            return _context.Registrations.Find(id);
        }

        public Room GetRoom(Guid id)
        {
            return _context.Rooms.Find(id);
        }
    }
}
```

ДОДАТОК Б

Приклад DTO (data transfer object) – об’єкту, який відображає сутність з бази даних та створений для серіалізації і передачі між компонентами.

```
using System;

namespace coldel.Persistance.Models.DTOS
{
    public class ProductDTO
    {

        public ProductDTO(Product product)
        {
            Id = product.Id;
            Client = product.Client;
            SKU = new SKUDTO(product.SKU);
            Start = product.CheckInDate;
            End = product.CheckOutDate;
            Title = $"{Client.Id} ({Client.LPN})";
        }

        public Guid Id { get; set; }

        public Client Client { get; set; }

        public SKUDTO SKU { get; set; }

        public DateTime Start { get; set; }

        public DateTime End { get; set; }

        public string Title { get; set; }
    }
}
```

ДОДАТОК В

Приклад автогенерованого класу, який відображає міграцію Entity Framework.

```
using System;
using Microsoft.EntityFrameworkCore.Migrations;

namespace coldel.Migrations
{
    public partial class NumberOfDaysreplacedwithCheckOutDate : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropColumn(
                name: "NumberOfDays",
                table: "OrderIn");

            migrationBuilder.AddColumn<DateTime>(
                name: "CheckOutDate",
                table: "OrderIn",
                type: "date",
                nullable: false,
                defaultValue:   new   DateTime(1,   1,   1,   0,   0,   0,   0,
DateTimeKind.Unspecified));
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropColumn(
                name: "CheckOutDate",
                table: "OrderIn");

            migrationBuilder.AddColumn<int>(
                name: "NumberOfDays",
                table: "OrderIn",
                nullable: false,
                defaultValue: 0);
        }
    }
}
```

ДОДАТОК Г

Приклад ASP.NET WEB API контролера, який дозволяє отримувати, створювати, редагувати та видаляти запити на додання продуктів у систему.

```
using System;
using System.Collections.Generic;
using System.Linq;
using coldel.Persistance.Core;
using coldel.Persistance.Models;
using coldel.Persistance.Models.DTOS;
using Microsoft.AspNetCore.Mvc;

namespace coldel.Controllers
{
    [Route("api/registrations")]
    [ApiController]
    public class RegistrationsController : ControllerBase
    {
        private readonly IProductRepository _repository;

        public RegistrationsController(IProductRepository repository)
        {
            _repository = repository;
        }

        // GET api/registrations
        [HttpGet]
        public ActionResult<IEnumerable<RegistrationDTO>> Get()
        {
            var registrations = _repository.GetRegistrations();
            return  Ok(registrations);
        }

        // POST /api/registrations
        [HttpPost]
        public            ActionResult<RegistrationDTO>            Add([FromBody]
AddRegistrationResource resource)
        {
            var    client    =    _repository.GetOrAddClient(resource.ClientName,
resource.lpn);
            var sku = _repository.GetSKU(resource.SKU);

            var reg = new Registration()
            {
                Id = new Guid(),
```

```
                SKU = sku,
                SKUDetails = sku.Details,
                Client = client,
                CheckInDate = resource.Start,
                CheckOutDate = resource.End
            };

            _repository.AddRegistation(reg);

            return Ok(new RegistrationDTO(reg));
        }
        // PUT /api/registrations
        [HttpPut]
        public            ActionResult<RegistrationDTO>            Edit([FromBody]
AddRegistrationResource resource)
        {
            var reg = _repository.GetRegistration(resource.RegistrationId);

            var     client     =     _repository.GetOrAddClient(resource.ClientName,
resource.lpn);

            var sku = _repository.GetSKU(resource.SKU);

            reg.ClientId = client.Id;
            reg.SKU = sku;
            SKUDetails = sku.Details;
            reg.CheckInDate = resource.Start;
            reg.CheckOutDate = resource.End;

            _repository.SaveChanges();

            return Ok(new RegistrationDTO(reg));
        }

        // DELETE /api/registrations
        [HttpDelete]
        public ActionResult Delete([FromBody] DeleteRegistrationResource resource)
        {
            _repository.DeleteRegistration(resource.Id);

            return Ok();
        }
    }
}
```

ДОДАТОК Д

Приклад Angular компоненту, який відображає форму для створення та редагування Recon запитів до системи.

```
import * as _ from 'underscore';
import { SaveRecon, Recon } from './../../models/recon';
import { Observable } from 'rxjs/Observable';
import { ActivatedRoute, Router } from '@angular/router';
import { ReconService } from './../../services/recon.service';
import { Component, OnInit } from '@angular/core';
import { ToastyService } from "ng2-toasty";
import 'rxjs/add/Observable/forkJoin';

@Component({
  selector: 'app-recon-form',
  templateUrl: './recon-form.component.html',
  styleUrls: ['./recon-form.component.css']
})
export class reconFormComponent implements OnInit {
  makes: any[];
  models: any[];
  features: any[];
  recon: Saverecon = {
    id: 0,
    makeId: 0,
    modelId: 0,
    isRegistered: false,
    features: [],
    contact: {
      name: '',
      email: '',
      phone: '',
    }
  };

  constructor(
    private route: ActivatedRoute,
    private router: Router,
    private reconService: ReconService,
    private toastyService: ToastyService) {

      route.params.subscribe(p => {
        this.recon.id = +p['id'] || 0;
      });
    }
```

```
ngOnInit() {
  var sources = [
    this.reconService.getDates(),
    this.reconService.getFeatures(),
  ];

  if (this.recon.id)
    sources.push(this.reconService.getRecon(this.recon.id));

  Observable.forkJoin(sources).subscribe(data => {
    this.makes = data[0];
    this.features = data[1];

    if (this.recon.id) {
      this.setRecon(data[2]);
      this.populateModels();
    }
  }, err => {
    if (err.status == 404)
      this.router.navigate(['/home']);
  });
}

private setRecon(v: Recon) {
  this.recon.id = v.id;
  this.recon.makeId = v.make.id;
  this.recon.modelId = v.model.id;
  this.recon.isRegistered = v.isRegistered;
  this.recon.contact = v.contact;
  this.recon.features = _.pluck(v.features, 'id');
}

onMakeChange() {
  this.populateModels();

  delete this.recon.modelId;
}

private populateModels() {
  var selectedMake = this.makes.find(m => m.id == this.recon.makeId);
  this.models = selectedMake ? selectedMake.models : [];
}

onFeatureToggle(featureId, $event) {
  if ($event.target.checked)
```

```
        this.recon.features.push(featureId);
      else {
        var index = this.recon.features.indexOf(featureId);
        this.recon.features.splice(index, 1);
      }
    }

    submit() {
      var result$ = (this.recon.id) ? this.reconService.update(this.recon) :
this.reconService.create(this.recon);
      result$.subscribe(recon => {
        this.toastyService.success({
          title: 'Success',
          msg: 'Data was sucessfully saved.',
          theme: 'bootstrap',
          showClose: true,
          timeout: 5000
        });
        this.router.navigate(['/recons/', recon.id])
      });
    }
  }
```

ДОДАТОК Е

Схема бази даних для Order In. Містить інформацію про замовлення, які ввозитимуться на склад та про продукти, які входять в ці замовлення.