Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Інформаційна система підприємства по виготовленню адаптивного одягу
Назва теми

КвРІСТ 200187.20.01.12 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *126 «Інформаційні системи та технології»*
Шифр, назва

Освітня програма *«Інформаційні системи та технології»*
Назва

Виконав: студент IV курсу, група ІСТ-20-1 _____ Я. С. Степанчук
Підпис Ініціали, прізвище

Керівник _____ І. О. Засорнова
Підпис, дата Ініціали, прізвище

Нормоконтролер _____ І. О. Засорнова
Підпис, дата Ініціали, прізвище

До захисту допускаю:
Зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Т.О. Говорущенко
Підпис Ініціали, прізвище

«14» червня 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Освітній рівень Бакалавр

Галузь знань 12 Інформаційні технології

Спеціальність 126 Інформаційні системи та технології

Освітня програма «Інформаційні системи та технології»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

" 10 " 01 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Степанчуку Ярославу Сергійовичу

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Інформаційна система підприємства по виготовленню адаптивного одягу

Керівник проекту (роботи) Засорнова І.О., к.т.н., доцент

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 15.02.2024 р. № 8

2. Строк подання студентом проекту (роботи) на кафедру 01.06.2024 р.

3. Вихідні дані до проекту (роботи) Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Дослідження предметної області та постановка задачі

Проєктування інструментальних засобів створення і використання інформаційних систем та технологій

Реалізація та тестування розробленої інформаційної системи

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Блок-схема принципу роботи ІС

UML-діаграма взаємодії користувача та адміністратора ІС

Структура бази даних ІС підприємтсва по виготовленню адаптивного одягу

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Засорнова І.О., доцент кафедри КІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІС		

7. Дата видачі завдання «_10_»_____01_____2024 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2024	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2024	виконано
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.03.2024	виконано
4	Робота над розділом 2 – проєктування інструметальних засобів створенняі використання інформаційних систем та технологій	01.04.2024	виконано
5	Робота над розділом 3 – реалізація та тестування розробленої інформаційної технології	29.04.2024	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2024	виконано
7	Попередній захист ВКР	30.05.2024	виконано
8	Захист ВКР на засіданні ЕК	Червень 2023 року	

Студент _____ Я. С. Степанчук
Підпис Ініціали, прізвище

Керівник роботи _____ І. О. Засорнова
Підпис Ініціали, прізвище

_____(підпис)

АНОТАЦІЯ

Тема кваліфікаційної роботи: Інформаційна система підприємства по виготовленню адаптивного одягу.

Автор роботи: Степанчук Ярослав Сергійович.

Керівник роботи: Засорнова Ірина Олександрівна.

Пояснювальна записка: 66 с., 19 рис., 3 дод., 49 джерел.

Графічна частина: 3 креслення.

ІНФОРМАЦІЙНА СИСТЕМА, API, REST, MICROSERVICES, ENTITY FRAMEWORK, DOTNET CORE, POSTGRESQL, MVC, DATABASE, HTTPS, JWT, SECURITY, GENERIC REPOSITORY

Метою роботи є забезпечення обміну даними між різними підрозділами підприємства в межах однієї інформаційної системи.

Об'єктом дослідження є процес розробки інформаційної системи підприємства по виготовленню адаптивного одягу.

Предметом дослідження є інформаційна система мікросервісної архітектури для організації процесу виробництва адаптивного одягу.

Практичне значення організація взаємодії між різними підрозділами підприємства в рамках однієї інформаційної системи, а також контроль та управління виробничими процесами.

Підпис студента

30.05.2024

Дата

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

IDE – Integrated Development Environment

CRUD – create read update delete

SQL – Structured Query Language

CRM – Customer Relationship Management

MES – Manufacturing Execution System

UI – User Interface

UX – User Experience

ІШІ – штучний інтелект

CAD – Computer Assisted Design

CAM – Computer Aided Manufacturing

ВСТУП

Сучасне підприємство, яке виготовляє адаптивний одяг, стикається з численними викликами в організації виробничих процесів, управлінні запасами та контролі якості продукції. Проблеми включають неефективне використання ресурсів, затримки у виробництві через недостатній контроль за матеріалами та готовою продукцією, а також проблеми з координацією між підрозділами. Відсутність інтегрованої інформаційної системи призводить до низької продуктивності, збільшення витрат та зниження конкурентоспроможності.

Розробка ефективної інформаційної системи є актуальною, оскільки дозволяє автоматизувати та оптимізувати виробничі процеси, покращувати управління ресурсами та контроль якості продукції. Це особливо важливо для підприємств, які виробляють адаптивний одяг, оскільки вони потребують точного контролю за якістю матеріалів та кінцевим продуктом.

Метою роботи є забезпечення обміну даними між різними підрозділами підприємства в межах однієї інформаційної системи.

Об’єктом дослідження є процес розробки інформаційної системи підприємства по виготовленню адаптивного одягу.

Предметом дослідження є інформаційна система мікросервісної архітектури для організації процесу виробництва адаптивного одягу.

Практичне значення полягає в організації взаємодії між різними підрозділами підприємства в межах однієї інформаційної системи, а також контроль та управління виробничими процессами.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Адаптивний одяг в даний час оцінюється в різних контекстах і для різних цільових груп користувачів. Одним з найбільш очевидних видів адаптивного одягу є одяг, призначений для людей з інвалідністю. Такий одяг розробляється з урахуванням потреб і фізичних обмежень цільової групи [1]. Він може включати різні елементи, такі як зручні застібки, широкі дверцята, регульовані розміри та інші адаптивні рішення, які легко носити і використовувати людям з різними фізичними обмеженнями.

Другий тип адаптивного одягу призначений для тих, хто працює в складних умовах. Це спеціалізований одяг для працівників, які працюють у суворих умовах, наприклад, у небезпечних або екстремальних умовах [2]. Такий одяг може включати захисні елементи, водонепроникні матеріали, ергономічний крій та інші технології, що забезпечують комфорт і безпеку працівників у різних ситуаціях.

Крім того, з’являється модний адаптивний одяг для загального споживчого ринку. Такий одяг поєднує в собі стиль і функціональність, дозволяючи людям з різними потребами відчувати себе комфортно і безпечно. Вони можуть включати такі елементи, як регульовані розміри, адаптивні застібки та інші інноваційні рішення, що забезпечують комфорт і стиль у повсякденному житті.

Кожен такий адаптивний одяг має унікальні особливості, які відповідають потребам конкретних користувачів. Враховуючи різноманітність та потреби цільових груп, розробники адаптивного одягу продовжують розробляти нові технології та рішення, щоб забезпечити комфорт та зручність для всіх користувачів.

Адаптивний одяг – це інноваційне модне рішення, спрямоване на покращення якості життя різних груп споживачів. Основні вимоги до адаптивного

одягу є не тільки естетичними, але й функціональними та практичними, і відіграють важливу роль у забезпеченні комфорту та зручності для користувача.

Першою і, мабуть, найважливішою вимогою є комфорт. Адаптивний одяг повинен бути не тільки модним, але й зручним у носінні, не сковувати рухів і бути комфортним протягом тривалого періоду часу. Це особливо важливо для людей з різними видами інвалідності, для яких комфорт в одязі є запорукою мобільності та незалежності. Друга вимога - безпека. Адаптивний одяг повинен бути безпечним для користувача. Це означає, що матеріал одягу повинен бути гіпоалергенним і не містити шкідливих речовин. Крім того, елементи дизайну, такі як застібки-блискавки та шви, також повинні бути безпечними та зручними, особливо для користувачів з порушеннями опорно-рухового апарату або слуху. Третя вимога - функціональність. Адаптивний одяг повинен відповідати конкретним потребам користувача. Це включає в себе різні функціональні елементи, які сприяють комфорту і практичності одягу, такі як застібки на липучках, корисні додаткові кишені і регульовані розміри. Четверта вимога - стиль і дизайн. Адаптивний одяг має бути модним та естетично привабливим, а також функціональним. Це допомагає користувачам відчувати себе впевнено та стильно, незалежно від їхніх потреб та обмежень.

Крім того, вимоги до адаптивного одягу можуть відрізнятися залежно від специфічних потреб різних груп споживачів. Наприклад, людям з інвалідністю може знадобитися спеціалізований одяг з елементами, що полегшують їхнє вдягання та пересування, тоді як інші споживачі можуть надавати перевагу модним і стильним елементам. Тому важливо враховувати різноманітність потреб користувачів при розробці адаптивного одягу, щоб зробити його максимально зручним і задовольняючим.

При розробці адаптивного одягу використання передових технологій та інноваційних матеріалів відіграє важливу роль у забезпеченні максимального комфорту та функціональності для користувача. Однією з ключових складових цього процесу є використання еластичних тканин. Стретч-тканини сьогодні

можуть бути розроблені з використанням передових технологій і можуть відповідати високим вимогам з точки зору гнучкості та адаптивності. Ці тканини не тільки безперешкодно пристосовуються до рухів людини, але й забезпечують надзвичайний комфорт під час носіння. Такий підхід дозволяє адаптивному одягу легко пристосовуватися до потреб різних користувачів і забезпечувати оптимальний рівень комфорту, що є важливим фактором підвищення якості життя.

Застібки-блискавки на сучасному адаптивному одязі не тільки зручні для одягання та знімання, але й є важливим елементом, що полегшує носіння для людей з інвалідністю та порушеннями рухливості. Такі застібки-блискавки можуть мати вбудовані датчики, які розпізнають рух і певні сигнали для активації автоматичного відкриття і закриття застібки, що полегшує користування для людей з інвалідністю. Крім того, деякі моделі одягу оснащені системами дистанційного керування, які дозволяють користувачам самостійно керувати застібками за допомогою додаткових пристроїв, таких як смартфони або пульти дистанційного керування. Ці технології значно полегшують одягання та роздягання людей з обмеженими фізичними можливостями та роблять повсякденне життя більш незалежним і комфортним.

Для досягнення більш високого рівня функціональності та індивідуального підходу в адаптивному одязі широко використовуються різні датчики, які можуть бути вбудовані безпосередньо в тканину або розміщені в спеціальних кишенях або вставках. Ці датчики відкривають нові можливості для вимірювання різних параметрів тіла і моніторингу фізіологічних показників. Наприклад, датчики можна налаштувати на безперервне вимірювання частоти серцевих скорочень, температури тіла, рівня активності та інших ключових фізіологічних показників. Ця інформація дозволяє користувачам контролювати стан свого здоров'я, а також налаштовувати адаптивний одяг для оптимального комфорту та фізичної форми. Завдяки датчикам адаптивний одяг може реагувати на зміни фізичного стану користувача, наприклад, автоматично регулюючи температуру або надаючи

додаткову підтримку в потрібний момент. Це особливо корисно для людей з інвалідністю та людей з різними захворюваннями, які потребують постійного моніторингу та підтримки. Тому використання різних датчиків в адаптивному одязі не тільки підвищує функціональність і комфорт, але і відкриває нові перспективи в області здоров'я і благополуччя власника.

Новітні технології та матеріали, що використовуються в адаптивному одязі, відображають значні досягнення в забезпеченні комфорту та функціональності для користувачів. Ці інновації не тільки враховують фізичні особливості кожної людини, але й прагнуть задовольнити різні потреби, наприклад, людей з інвалідністю або особливі потреби в одязі. При розробці адаптивного одягу важливо враховувати різні фізіологічні особливості та потреби користувача. Використовувані технології включають еластичні тканини, які дозволяють одягу адаптуватися до рухів тіла і дарують відчуття комфорту і свободи. Застібки-блискавки та системи застібок розроблені відповідно до потреб користувача і можуть мати спеціальні функції для полегшення одягання та роздягання. Датчики, вбудовані в тканину, вимірюють різні параметри, такі як частота серцевих скорочень і температура тіла, дозволяючи користувачам стежити за своїм самопочуттям і підтримувати оптимальний стан здоров'я під час носіння одягу. З огляду на це, останні розробки в галузі адаптивного одягу спрямовані на створення виробів, які не тільки відповідають індивідуальним потребам кожного користувача, але й підвищують комфорт і якість життя. Такий підхід покликаний зробити моду доступною і зручною для кожного, незалежно від його особливостей і можливостей.

В рамках аналізу предметних областей, що охоплюють виробництво адаптивного одягу, важливо ретельно розглянути основні інформаційні потреби компаній. Специфіка сектору визначається не тільки різноманітністю матеріалів та конструкцій, але й необхідністю швидкого моніторингу та аналізу великих обсягів даних на всіх етапах виробництва. Компаніям, що спеціалізуються на адаптивному одязі, необхідно мати ефективні інструменти для моніторингу та

контролю виробничих процесів, управління складом, контролю якості та маркетингових стратегій. У виробництві адаптивного одягу інформаційні потреби є важливим елементом успіху компанії. Моніторинг виробничого процесу, вибір найкращих матеріалів, управління запасами, контроль якості, розробка та реалізація маркетингових стратегій - всі ці аспекти вимагають правильної інформаційної інфраструктури та інструментів управління. Тому аналіз інформаційних потреб є важливим кроком у створенні ефективної інформаційної системи, спрямованої на оптимізацію всіх аспектів бізнесу.

При аналізі потреб в системі управління виробництвом для компанії, що спеціалізується на виробництві адаптивного одягу, необхідно враховувати різноманітність матеріалів і конфігурацій виробів. Одяг може бути адаптований до різних розмірів і стилів і може бути виготовлений з різних тканин і матеріалів, таких як трикотаж, текстиль і шкіра. Тому виробничий процес повинен бути гнучким до змін і забезпечувати ефективне планування та координацію роботи на кожному етапі. Інструменти управління виробництвом повинні бути здатні керувати різними аспектами виробничого процесу від закупівлі сировини до випуску готової продукції, включаючи моніторинг та оптимізацію запасів, планування виробничих операцій та ефективний розподіл ресурсів для максимізації продуктивності. Крім того, система управління повинна також враховувати індивідуальні потреби кожного замовлення, такі як варіанти, додаткові опції та інші особливі вимоги клієнтів. Оптимізація процесів складання та пошиття є важливим аспектом адаптивного управління швейним виробництвом. Передові технології та методи можуть скоротити час виробництва, покращити якість продукції та підвищити гнучкість виробництва. Важливо враховувати індивідуальні вимоги кожного замовлення, автоматизувати процеси складання та пошиття і мати системи, які гарантують ефективну координацію між різними відділами та виробничими підрозділами.

Розглядаючи вимоги до систем складського обліку в контексті адаптивного швейного виробництва, важливо враховувати різноманітність і складність

складських процесів. Такі склади можуть бути складними екосистемами, де зберігаються не тільки різні матеріали, а й компоненти для виробництва різних моделей одягу, а також готова продукція різних розмірів і варіантів. Тому важливо мати систему, яка забезпечує ефективне управління, а також правильний облік усіх цих складських активів. Системи складського обліку повинні відстежувати рух матеріалів і готової продукції від постачальника до замовника. При цьому слід враховувати не лише кількість і розмір партій, але й дати поставок, термін придатності матеріалів, а також характеристики деталей і готової продукції. Така детальна інформація дозволяє підтримувати оптимальні рівні запасів, уникати надлишкових і недостатніх запасів, а також оптимізувати процеси виробництва і доставки. Крім того, важливо мати систему контролю за переміщенням матеріалів і готової продукції на складі. Це передбачає розміщення товарів на складі відповідно до категорій продукції та вимог до зберігання, а також використання найоптимальніших маршрутів для забезпечення швидкості та ефективності виконання замовлень. Такі системи допомагають уникнути заторів та заощадити цінний час і ресурси. Загалом, системи складського обліку та управління є невід'ємною частиною інформаційної інфраструктури підприємств, що спеціалізуються на виробництві адаптованого одягу. Ця система не тільки забезпечує коректний облік і контроль руху матеріалів і готової продукції, а й відіграє важливу роль у забезпеченні ефективності та конкурентоспроможності підприємств на ринку.

Третій елемент аналізу – вимога до системи управління якістю - має велике значення у виробництві протезів та ортезів. Особливістю цього сектору є те, що виготовлені вироби повинні відповідати не тільки загальним стандартам якості, але й індивідуальним потребам кожного клієнта. У цьому контексті ефективна система управління якістю повинна забезпечити контроль якості виробництва на кожному етапі, від вибору матеріалу до готового виробу. Це означає, що кожна стадія виробництва повинна ретельно контролюватися, починаючи від надходження сировини і матеріалів на склад, через процеси пошиття та

оздоблення, до остаточної перевірки готового виробу перед відправкою замовнику. Крім того, система управління якістю повинна бути здатна виявляти та усувати можливі дефекти у виробі. Це включає в себе як ручний контроль кваліфікованими фахівцями, так і використання різних технологій і автоматизованих методів контролю, які допомагають своєчасно виявити дефекти і помилки у виробництві.

Адаптивний одяг часто є продуктом для певної цільової групи, яка включає людей з різними фізичними характеристиками та потребами, наприклад, людей з інвалідністю або тих, хто шукає одяг з підвищеною функціональністю. Це ускладнює для компаній ефективну адаптацію своїх стратегій продажу та маркетингу для досягнення успіху на ринку. Перш за все, необхідно ретельно спланувати систему дистрибуції, щоб адаптований одяг був максимально доступним для цільової групи. Це включає в себе розгляд різних каналів дистрибуції, таких як інтернет-магазини, спеціалізовані магазини або роздрібні торговці, що спеціалізуються на продажі адаптивного одягу. Крім того, оскільки адаптивний одяг часто вимагає індивідуальної примірки та модифікації, важливо мати механізми для прийняття та обробки індивідуальних замовлень. Крім того, маркетингові системи повинні бути спрямовані на ефективне охоплення цільових споживачів і створення інтересу до продукції. Це включає в себе маркетингові дослідження для розуміння потреб та вподобань цільових споживачів, а також дослідження конкурентного середовища. На основі цих даних можуть бути розроблені стратегії просування, включаючи використання різних каналів комунікації, рекламні кампанії, участь у спеціальних виставках і заходах, а також розробку унікального брендингу для залучення цільової аудиторії та диференціації продукту від конкурентів. Важливо також постійно моніторити ринок і адаптувати маркетингові стратегії до змін попиту та конкурентного середовища. Компанії, що спеціалізуються на виробництві адаптивного одягу, стикаються з унікальними викликами та потребами в інформаційній підтримці. Аналіз цих потреб показує, що вони стосуються всіх аспектів бізнесу, починаючи

з етапу розробки нової моделі і закінчуючи моментом, коли продукт випускається на ринок і пропонується споживачам. Починаючи з ранніх етапів планування виробництва, інформаційні системи повинні мати можливість ефективно аналізувати ринковий попит, передбачати модні тенденції та прогнозувати вартість і доступність ресурсів і матеріалів. Це важливо для того, щоб система могла постійно реагувати на мінливі потреби клієнтів і ринкові умови. Крім того, на етапі виробництва система повинна ефективно керувати процесами складання, пошиття та контролю якості. Це означає, що потрібні інструменти для відстеження робочого часу, контролю якості виробленої продукції, управління запасами та виробничими потужностями. Управління складом - ще один важливий аспект, який потребує інформаційної підтримки. Система повинна надавати інформацію в режимі реального часу про наявність сировини і готової продукції на складі, автоматично оновлювати дані про виробничі партії і забезпечувати надійний механізм контролю запасів і логістики дистрибуції. Звісно, не менш важливою є інформаційна підтримка у сфері продажів та маркетингу. Це аналіз ринку, планування маркетингових кампаній, управління замовленнями та лояльністю клієнтів. Ефективна система повинна забезпечувати швидкий обмін даними між різними підрозділами компанії для забезпечення безперебійного та успішного ведення бізнесу в цілому. Таким чином, аналіз інформаційних потреб компанії показує, що для успішного ведення бізнесу в адаптивній швейній промисловості необхідна комплексна та інтегрована інформаційна система, яка забезпечує оптимальне функціонування всіх аспектів бізнесу та його конкурентоспроможність на ринку.

1.2 Аналіз наявних інформаційних технологій предметної області

Аналіз існуючих інформаційних технологій у сфері адаптивного одягу показує, що існує широкий спектр інструментів, які використовуються для підтримки та оптимізації виробничих процесів та досліджень і розробок у цій

галузі. Інформаційні технології в цьому контексті включають як програмні, так і апаратні рішення, технічні процеси і методи, спрямовані на підвищення ефективності виробництва, якості та конкурентоспроможності адаптивного одягу. Програмне забезпечення в цій сфері включає CAD/CAM системи (Computer Aided Design/Computer Aided Manufacturing), які використовують комп'ютерні технології для проектування та виробництва адаптивного одягу. Ці системи дозволяють розробникам і дизайнерам створювати і візуалізувати моделі одягу та автоматизувати виробничий процес, допомагаючи підвищити продуктивність і скоротити час розробки. Паралельно з CAD/CAM-системами в індустрії адаптивного одягу активно використовуються технології штучного інтелекту (ШІ) та машинного навчання. Ці технології використовуються для аналізу споживчих даних і вимог до одягу, вибору найкращих рішень у виробничому процесі та прогнозування тенденцій в індустрії моди.На апаратному рівні у виробництві адаптивного одягу може використовуватися низка сучасних технологій, таких як 3D-принтери для створення прототипів та виготовлення окремих частин одягу, а також інтерактивні технології для створення "розумного" одягу з інтегрованими датчиками та електронікою.

Технологічні процеси в секторі виробництва адаптивного одягу включають інноваційні методи, спрямовані на підвищення якості та швидкості виробництва, такі як термоусадка, ультразвукове склеювання та лазерне різання. Загальний стан сучасних інформаційних технологій в секторі виробництва адаптивного одягу свідчить про постійний прогрес та інновації в цьому секторі і надає нові можливості компаніям, що займаються виробництвом адаптивного одягу, розробляти і впроваджувати новітні інформаційні системи і технології.

Системи управління взаємовідносинами з клієнтами (CRM) є невід’ємною частиною підтримки бізнесу в секторі адаптивного одягу. Вони дозволяють компаніям ефективно взаємодіяти з клієнтами, розуміти їхні потреби та вподобання, а також будувати і розвивати довгострокові відносини. Наприклад, за допомогою CRM-системи компанії можуть збирати та аналізувати дані про

минулі замовлення та взаємодію з клієнтами, розуміти їхні вподобання та створювати персоналізовані пропозиції.

Прикладом CRM-системи для адаптивної індустрії одягу є Salesforce CRM. Ця система надає рішення для збору та обробки даних про клієнтів, автоматизації комунікації та створення персоналізованих пропозицій. Вона дозволяє компаніям ефективно управляти взаємодією з клієнтами, враховуючи їхні індивідуальні потреби та вимоги. Salesforce CRM - це інтегрована система управління взаємовідносинами з клієнтами (CRM), яка надає широкий спектр інструментів для збору, аналізу та управління даними про клієнтів. Платформа призначена для підвищення ефективності взаємодії з клієнтами та їхньої задоволеності. Salesforce CRM відома своєю високою функціональністю та гнучкістю, що дозволяє компаніям адаптувати систему до своїх конкретних потреб та процесів. Основні функції Salesforce CRM включають збір і зберігання даних про клієнтів, включаючи контактну інформацію, історію взаємодії, інформацію про замовлення та інші важливі дані. Ця інформація зберігається в центральній базі даних, що дозволяє легко відстежувати та аналізувати взаємодію з клієнтами. Крім того, Salesforce CRM надає інструменти для автоматизації різних бізнес-процесів, таких як маркетингові кампанії, продажі та обслуговування клієнтів. Використовуючи ці інструменти, компанії можуть ефективно управляти робочими процесами і виконувати завдання швидко і точно. Платформа підтримує розробку та інтеграцію додаткових функцій і модулів, що дозволяє компаніям налаштовувати систему відповідно до своїх унікальних потреб і вимог.

Іншим прикладом є HubSpot CRM, яка пропонує рішення для збору та аналізу даних про клієнтів, автоматизації процесів маркетингу та продажів, а також створення персоналізованих пропозицій. HubSpot CRM - це інтегрована система управління взаємовідносинами з клієнтами, яка забезпечує комплексне рішення для ефективного управління взаємовідносинами з клієнтами. Вона включає в себе ряд функцій, спрямованих на збір, аналіз та оптимізацію даних про клієнтів. Вона спрямована на збір, аналіз та оптимізацію даних про клієнтів,

автоматизацію процесів маркетингу та продажів, а також створення персоналізованих пропозицій. Однією з найважливіших особливостей HubSpot CRM є її інтуїтивно зрозумілий і простий у використанні інтерфейс. Без складних налаштувань або додаткового навчання користувачі можуть швидко освоїти систему та ефективно використовувати всі її функції.

Однією з основних функцій HubSpot CRM є збір та обробка даних про клієнтів. Система може збирати інформацію про людей, їхню взаємодію з людьми, а також про їхні інтереси та вподобання. Це дозволяє компаніям краще розуміти своїх клієнтів і надавати їм більш персоналізовані послуги та пропозиції. Крім того, HubSpot CRM автоматизує процеси маркетингу та продажів. Вона дозволяє компаніям ефективно управляти маркетингом і продажами, надаючи інструменти для автоматичного відстеження та управління контактами з клієнтами, створення та відстеження маркетингових кампаній, а також автоматизації рутинних завдань. Ще однією важливою особливістю HubSpot CRM є можливість створювати персоналізовані пропозиції для клієнтів. Система дозволяє компаніям створювати та відстежувати персоналізовані пропозиції для кожного клієнта на основі його потреб та вподобань, що сприяє підвищенню ефективності продажів та задоволеності клієнтів. Крім того, компанії можуть використовувати системи управління складом, бухгалтерські системи, системи аналізу даних і звітності для спрощення операцій і прийняття обґрунтованих рішень.

Аналіз інформаційних систем провідних компаній-виробників адаптивного одягу дозволяє виділити кілька ключових підходів і рішень: Система SAP ERP, яку використовує SmartWear, є провідною платформою в галузі інтегрованих інформаційних систем. Система пропонує комплексний підхід до автоматизації всіх аспектів виробництва адаптивного одягу, від планування до фінансового обліку. Використовуючи SAP ERP, SmartWear може ефективно управляти запасами матеріалів. Це має вирішальне значення у виробництві адаптивного одягу, де важливо мати достатній запас різних типів тканин, фурнітури та інших

матеріалів. Інтегрована система планування виробництва забезпечує оптимізацію виробничого процесу з урахуванням різноманітності асортименту продукції та потреб клієнтів. Інтеграція з системами управління взаємовідносинами з клієнтами (CRM) гарантує ефективне отримання замовлень, контакт з клієнтами та своєчасне здійснення поставок. Це забезпечує високу задоволеність клієнтів і зміцнює позиції SmartWear на ринку адаптивного одягу.

Siemens MES (Manufacturing Execution System), яку використовує Adaptive Fashion – це комплексна система, спеціально розроблена для оптимізації виробничих процесів у секторі адаптивного одягу. Система забезпечує повний контроль над усіма етапами виробництва, від отримання замовлень до відвантаження готової продукції. Одним з ключових функціональних елементів Siemens MES є управління якістю виробництва. Система автоматизує процес контролю якості матеріалів, комплектуючих та готової продукції. За допомогою спеціальних модулів і датчиків MES може автоматично виявляти дефекти і невідповідності, дозволяючи виробничому процесу реагувати і виправляти проблему без затримок. Крім того, Siemens MES також забезпечує моніторинг обладнання. Це означає, що система відстежує роботу обладнання в режимі реального часу для визначення його ефективності та робочого стану. У разі виникнення проблеми або несправності MES надає операторам і технічним фахівцям детальну інформацію про стан обладнання та рекомендації щодо вирішення проблеми. Ще одним важливим аспектом Siemens MES є моніторинг прогресу у виконанні замовлень. Система автоматично відстежує кожен етап виробництва, від обробки замовлення до остаточного пакування та відправки. Це дозволяє керівництву компанії отримувати звіти про стан виробництва в режимі реального часу і приймати швидкі та обґрунтовані бізнес-рішення.

1.3 Визначення вимог до інструментальних засобів створення та використання інформаційних систем та технологій, розробка технічного завдання

При розробці інформаційної системи для компанії, що спеціалізується на виробництві адаптивного одягу, важливо адаптувати вимоги до інструменту до особливостей галузі та потреб користувачів. Аналіз цільового сектору допоможе визначити найбільш підходящий інструмент для досягнення цілей системи. Однією з найважливіших вимог є використання Web API ASP.NET Core 8 для розробки серверної частини інформаційної системи. З огляду на потреби в надійності, масштабованості і простоті управління, ASP.NET Core 8 Web API є найкращим вибором. Фреймворк дозволяє створювати потужні та гнучкі інтерфейси прикладного програмування (API), які сприяють ефективній взаємодії між клієнтськими додатками та серверною частиною системи. Використання ASP.NET Core 8 Web API також забезпечує сумісність з новітніми технологіями та інтеграцію з іншими системами.

Для забезпечення повної функціональності та ефективної взаємодії з користувачем для реалізації клієнтської частини системи використовується ASP.NET Core 8 MVC. За допомогою цього фреймворку можна створювати веб-додатки з високою швидкістю відгуку та зручною архітектурою. Використання ASP.NET Core 8 MVC забезпечує зручний та інтуїтивно зрозумілий інтерфейс для користувачів системи, а також дозволяє легко налаштовувати та змінювати функціонал у майбутньому. У той же час, важливо переконатися, що використовується інтегроване середовище розробки (IDE), яке підтримує ASP.NET Core 8, щоб забезпечити легкість і ефективність процесу розробки. Вибір конкретного IDE може залежати від внутрішніх процесів і переваг команди розробників, але важливо переконатися, що у них є всі необхідні інструменти для роботи з ASP.NET Core 8.

З огляду на важливість безпеки даних, обов'язковою умовою також є використання сучасних інструментів, таких як механізми шифрування та

аутентифікації для захисту інформації. Це забезпечує конфіденційність і цілісність даних та запобігає несанкціонованому доступу до них.

ASP.NET Core 8 MVC - це потужний інструмент для розробки веб-додатків, який не тільки швидко реагує, але й забезпечує ефективну та масштабовану архітектуру. Заснований на моделі Model-View-Controller (MVC), фреймворк спрощує розробку, тестування та підтримку коду, дозволяючи розбивати додатки на логічні компоненти. ASP.NET Core 8 MVC забезпечує гнучкість у виборі технології та підходу до розробки, дозволяючи розробникам використовувати різні шаблони проектування та інші методи для досягнення своїх цілей. Крім того, фреймворк має вбудовану підтримку для створення користувацьких інтерфейсів, які підтримують багатофункціональність, наприклад, асинхронні запити, перевірку даних і безпеку. Завдяки вбудованій інтеграції з іншими компонентами ASP.NET Core, такими як Entity Framework Core для інтеграції з базами даних та Identity для аутентифікації та авторизації, розробники можуть легко створювати повноцінні веб-додатки, що відповідають сучасним вимогам користувачів та бізнесу. Загалом, використання ASP.NET Core 8 MVC для реалізації клієнтської частини інформаційної системи гарантує якість, продуктивність та масштабованість додатків [3-21].

Враховуючи специфіку індустрії адаптивного одягу, де конфіденційність даних про дизайн, матеріали та замовлення відіграє важливу роль, інформаційна безпека є ключовим елементом інформаційної системи компанії. Однією з основних вимог є використання сучасних інструментів захисту даних. Криптографічні механізми допомагають забезпечити конфіденційність і цілісність інформації, що зберігається в базах даних і передається між серверами і клієнтськими додатками [22, 23]. Використання різних рівнів шифрування, таких як шифрування баз даних і каналів зв'язку, може допомогти запобігти несанкціонованому доступу до конфіденційної інформації. Механізми автентифікації гарантують, що тільки авторизовані користувачі можуть отримати доступ до системи та її функцій. Використання різних методів автентифікації,

таких як паролі, двофакторна автентифікація та біометричні дані, забезпечує надійний контроль доступу до системи. Загалом, впровадження сучасних засобів захисту інформації є необхідним кроком для забезпечення високого рівня безпеки даних в інформаційних системах компаній-розробників програмного забезпечення. Такий підхід допомагає запобігти можливим загрозам безпеки та зберегти довіру клієнтів до компанії.

Основні вимоги, визначені в цьому підрозділі, є наступними:

1. Система буде розроблена на основі ASP.NET Core 8 Web API для серверної частини та ASP.NET Core 8 MVC для клієнтського інтерфейсу.

2. MediatR та СУБД PostgreSQL для обробки інформації та вміщувати основну логіку обробки інформації та взаємодії з СУБД PostgreSQL.

3. В цілому система використовує підхід з розділеною архітектурою, де основний API відповідає за обробку даних та зберігання інформації в базі даних, в той час як окремі проекти кожної майстерні використовуються як інтерфейси для взаємодії з основним API.

4. Окремі проекти для кожного воркшопу повинні бути підготовлені у форматі ASP.NET Core MVC для взаємодії з основним API через визначені точки доступу

5. Система повинна бути розділена на модулі з чітко визначеною структурою, щоб її можна було легко розширити новими функціями.

На основі виявлених вимог та аналізу технологій розроблено технічне завдання з такими складовими:

- створення API для керування та обробки інформації про адаптивний одяг;

- реалізація можливості реєстрації користувачів та авторизації через JWT [24];

- розробка інтерфейсу для адміністрування системи та управління користувачами;

- забезпечення можливості взаємодії з окремими цехами для підправки та отримання запитів через відповідний інтерфейс.

Нефункціональні вимоги:

- висока продуктивність та масштабованість системи для обробки великого обсягу дани;

- забезпечення безпеки даних через механізми аутентифікації та авторизації;

- інтерфейс повинен бути інтуїтивно зрозумілим та зручним у використанні для користувачів;

- система повинна бути легко розширюваною для можливості додавання нового функціоналу у майбутньому.

Архітектура системи:

- мікросервісна архітектура з використанням ASP.NET Core Web API для головного сервісу та окремих проектів для кожного цеху;

- застосування шаблону Mediator для керування подіями та обробки запитів;

- зберігання даних в базі даних PostgreSQL.

Основною платформою для розробки інформаційних систем є ASP.NET Core 8. На серверній стороні системи використовується Web API ASP.NET Core 8 у поєднанні з шаблонами MediatR для кращої організації обробки запитів та управління виконанням [25]. Для зберігання даних було обрано базу даних PostgreSQL, що відповідає вимогам продуктивності та надійності.

Крім того, відділи компанії мають власні проекти на базі ASP.NET Core 8 MVC. Ці проекти використовуються як інтерфейс до основного API і не обробляють дані. Це забезпечує зручність та гнучкість при роботі з інформаційними системами та дозволяє різним відділам працювати незалежно.

Враховуючи вищезазначені технічні вимоги та архітектурні особливості системи, можна зробити висновок, що обрані інструменти мають високий потенціал для успішного створення та використання адаптивної готової до використання корпоративної інформаційної системи.

1.4 Висновки. Постановка задачі

У першому розділі було проведено комплексне дослідження предметної області та визначено основні вимоги до розробки інформаційної системи для підприємства з виготовлення адаптивного одягу.

Під час аналізу предметної області було виявлено структурні та функціональні особливості виробничого процесу підприємства. Визначено ключові підрозділи, включаючи швейний, виробничий та підготовчий цехи, а також складські приміщення. Описано процеси, що відбуваються в кожному з підрозділів, такі як приймання матеріалів, перевірка їх якості, виробництво товарів, перевірка якості готової продукції та управління запасами. Такий аналіз дозволив визначити основні функціональні потреби підприємства, які мають бути враховані при розробці інформаційної системи.

Було проведено аналіз існуючих інформаційних технологій, які можуть бути використані для автоматизації процесів на підприємстві. З'ясовано, що сучасні технології, такі як PostgreSQL для управління базами даних, ASP.NET Core Web API для серверної частини, Entity Framework для роботи з базами даних та HTML/CSS/JS для розробки клієнтського додатку, найбільш ефективно відповідають вимогам підприємства.

На основі проведеного аналізу було сформульовано вимоги до інструментальних засобів створення та використання інформаційних систем і технологій. Визначено, що система має забезпечувати інтеграцію всіх підрозділів підприємства, автоматизацію управління матеріалами, контроль якості продукції та можливість масштабування і гнучкої адаптації до змін ринкових умов. На основі цих вимог було розроблено технічне завдання, яке стало основою для подальшої розробки інформаційної системи. Поставлено наступні задачі:

- дослідити предметну область;
- спроєктувати інструментаьні засоби створення і використання ІС;
- розробити базу даних та реалізувати ІС.

2 ПРОЄКТУВАННЯ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ СТВОРЕННЯ І ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

2.1 Проєктування бази даних

Інформаційна система включає в себе три основні компоненти:

- база даних;
- сервер;
- клієнт.

При проєктуванні інформаційної, головним компонентом являється саме база даних. Визначення дизайну бази даних має відображення на сервер та на клієнті, оскільки поняття бази даних являється первинним для будь якої інформаційної системи. База даних повинна забезпечувати структуроване збереження даних та доступ до них, а також можливість взаємодії із ними різними методами. Визначення архітектури бази даних призводить до різної обробки їх на рівні серверу та відображення на клієнті, а також перелік інтерактивних компонентів для взаємодії з базою даних.

Чому база даних являється головним та ключовим компонентом в інформаційній системі? Тому що вона забезпечує безпечне збереження даних на рівні операційної системи або хмарних сховищ, які розгортаються на відповідних операційних системах. База даних це структуроване місце для збереження даних, де самі дані зберігаються в хеш-таблицях на рівні операційної системи. Сучасна база даних повинна забезпечувати доступ до даних в режимі читання та запису, а також можливість побудови запитів для взаємодії з ними. Такі запити пишуться на мові SQL, яка буде розглянута пізніше. Також база даних повинна забезпечувати можливість зв’язку між таблицями, яких є три наступних варіанти [26]:

- один до одного;
- один до багатьох;
- багато до багатьох.

Збереження відповідних даних на рівні таблиць не дає явного розуміння базі даних про їх зв'язок, тому для цього необхідне налаштування конфігурації бази даних та відповідних таблиць. В разних базах даних цю операцію можна виконати за допомогою різних дій, основної з яких є конфігурація за допомогою мови запитів SQL.

SQL (Structured Query Language – мова структурованих запитів) – це мова програмування, яка використовується для роботи з реляційними базами даних; SQL дозволяє виконувати різні операції з даними, такі як додавання, видалення, модифікація та пошук інформації в базі даних. Простіше кажучи, SQL – це спосіб взаємодії з базою даних за допомогою команд. Він забезпечує стандартний спосіб виконання запитів до бази даних незалежно від використовуваної системи управління базами даних (СКБД) [27].

Проєктування бази даних складається з декількох етапів, які ми розберемо послідовно. Перший етап включає в себе загальне розуміння структури системи та усіх підсистем, що є досить важливим нюансом в проєктуванні бази даних. Це потрібно для того, щоб було чітке загальне розуміння структур, які необхідно зберігати в самій базі даних. У нашому випадку, уся система буде поділена на малі підсистеми, за кожну з яких відповідає окрема таблиця в базі даних для збереження поточного стану підсистеми. Конкретний стан підсистеми виражається його внутрішніми даними. Для прикладу, для складу його поточний стан буде виражатись в кількості певних матеріалів та товарів на цьому ж складі у певний момент часу. Надходження нових товарів, їх переміщення в середині системи між підсистемами або їх вихід за межі системи буде призводити до зміни стану цієї підсистеми. Загальний стан системи включає в себе сукупність станів усії внутрішнії об'єктів системи у певний момент часу.

База даних повинна відображати сукупний стан усієї системи, відповідно, містити інформацію про всі матеріали та продукти, а також інформації про їх стан, місце знаходження та відношення до певного цеху.

Проєктування самої бази даних розпочинається з головної таблиці, яка може бути вибрана довільно. Для прикладу, для інформаційної системи підприємства виготовлення адаптивного одягу міститиме наступні властивості:

- ідентифікатор;
- назва товару;
- опис товару;
- короткий опис товару;
- валюта;
- ціна;
- кількість;
- рейтинг;
- кількість замовлень.

Без врахування допоміжних даних, які можуть зберігатись в інших таблицях накшталт фотографій, нам потрібно зробити зв'язок з іншими таблицями, зокремо, потрібно вказати, що цей продукт належить певній одиниці системи, тобто, цеху, складу або магазину. Для цього опишемо таблицю, яка міститиме дані про цех або іншу фізичну підсистему. Дана таблиця міститиме наступні дані:

- ідентифікатор;
- назва;
- тип цеху;
- місто;
- вулиця;
- номер будинку;
- номер корпусу;
- додаткова адреса.

Варто зауважити, що даний список описує лише дані які необхідно зберігати на рівні бази даних. Насправді ж схема даних матиме приблизно наступний вигляд як на рисунку 2.1:

Рисунок 2.1 – Схема даних бази даних

На цьому рисунку зображена первинна схема даних, яка може бути змінине в майбутньому згідно з потребами функціоналу системи. До схеми даних можуть бути додані нові таблиці або ж модифіковані попередні, для опису наступного:

- переміщення товару в середині системи;
- дії користувача;
- інтернет магазин та його залежні дані;
- фінансування відділів;
- виплата заробітньої плати;
- поєднання матеріалів;
- результати подій.

Після проєктування бази даних з’являється необхідність в проєктуванні серверної частини інформаційної системи для забезпечення функціональної частини обробки даних в системі.

2.2 Проєктування сервера

Інформаційна система підприємтва з пошиття адаптивного одягу використовує мікросервісну архітектуру для серверної взаємодії разом із різними

підходами [28]. Розглянемо рисунок 2.2, на якому зображена схема взаємодії кінцевого кориистувача із базою даних.

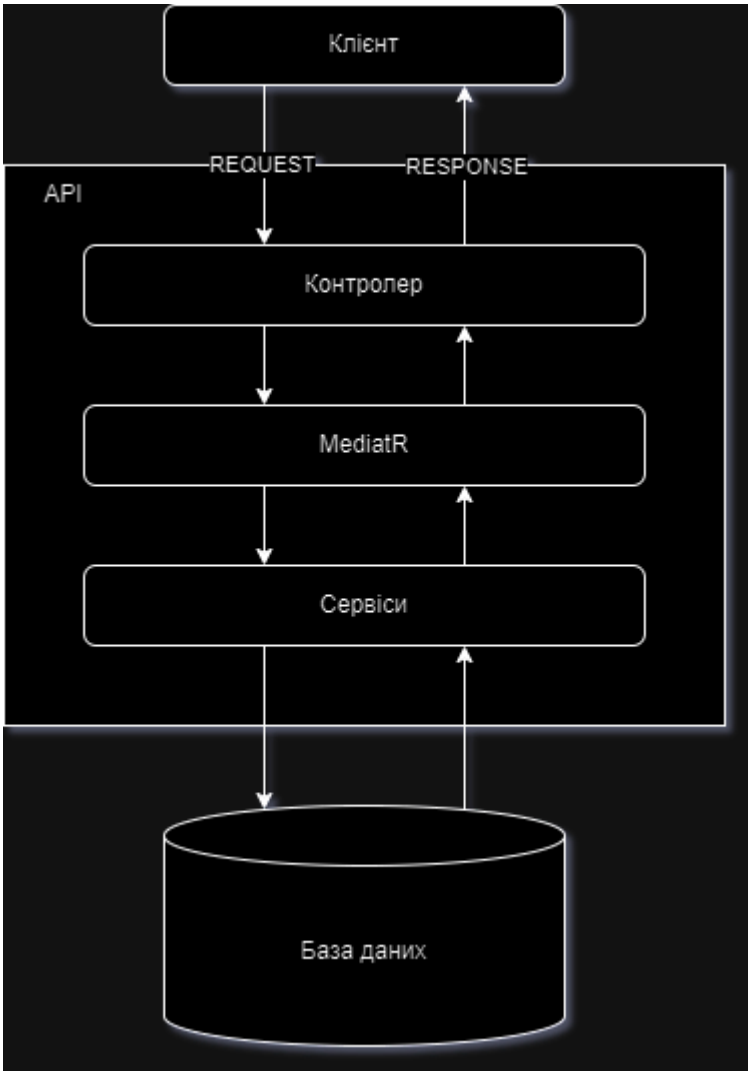


Рисунок 2.2 – Обробка запитів

Цей приклад показує, що кожен запит, надісланий користувачем, повинен пройти через важливий рівень API сервера, який відповідає за обробку та управління інформацією. Цей рівень API взаємодіє з різними сервісами, які функціонують на рівні транзакцій, показаних на відповідній схемі. Спочатку HTTP-запит надсилається до контролера, який забезпечує точку входу до певної кінцевої точки сервера [29]. Для забезпечення чистоти архітектури додатку та оптимальної продуктивності сервісу, контролер перенаправляє запит через

MediatR [30]. Цей інструмент дозволяє ефективно управляти потоками даних і спрощує взаємодію між компонентами додатку: звертаючись до MediatR, контролер передає управління відповідним сервісам, які викликають відповідні методи обробки даних і дії з ними у відповідь на запити користувачів. Така структура запитів, що проходять через різні рівні сервера, забезпечує ефективну та організовану обробку даних у системі.

На цьому етапі важливо зазначити, що між сервісним рівнем і базою даних є невеликий, але важливий прошарок, а саме реалізація паттерну Generic Repository. Використання відповідного патерну не тільки спрощує взаємодію з різними компонентами системи, але й забезпечує максимально зручний та масштабований код. Після звернення до бази даних отримується набір даних у вигляді результатів запитів, для перетворення яких у відповідні класи використовується фреймворк Entity Framework Core [31]. Фреймворк автоматично відстежує дані і перетворює їх у відповідний тип. Отриманий тип, який представляє об'єкт у базі даних, називається "сутність" або "entity". Це дозволяє легко маніпулювати даними в коді програми і поводитися з ними як з програмними об'єктами.

Сервер використовує кінцеві точки для реагування на HTTP запити. Усі ці кінцеві точки знаходяться в середині контролерів та мають назву «екшени» або «дії». Кожний контролер повинен мати за свою «область відповідальності» для забезпечення чистоти коду [32]. На сервері будути визначені наступні контролери:

1. AuthenticationController.
2. UserController.
3. ProductController.
4. WorkshopController.
5. MaterialController.
6. SupplyController.

Ці контролери являються базовими для забезпечення мінімального функціоналу системи.

Після рівня контролера запит має бути перенаправлений на рівень MediatR, який відіграє важливу роль у цій архітектурі. Цей рівень забезпечує централізовану обробку інформації та перенаправляє запити до інших сервісів системи. По суті, роль MediatR полягає в тому, щоб відокремити логіку обробки запитів від рівня контролера [33]. У випадку такої побудови інформаційної системи рівень MediatR стає залежним від рівня сервісів. Це пов'язано з тим, що саме сервіси відповідають за спеціалізовану обробку та маніпулювання інформацією, що надходить через MediatR. Така структура дозволяє системі залишатися гнучкою і розширювати свою функціональність без значних змін у коді. На рівні сервісів можуть бути розгорнуті різні функціональні модулі для виконання різних завдань. Це можуть бути сервіси, що взаємодіють з базами даних, файловими системами та хмарними сервісами, або сервіси, що виконують рутинну обробку інформації, таку як аналіз даних, обчислення та взаємодію з іншими мікросервісами. Кожен сервіс відповідає за певну задачу і має власний інтерфейс для взаємодії з іншими компонентами системи. Такий підхід забезпечує високий рівень масштабованості та підтримки системи в цілому.

Для оптимального управління базами даних сервіс використовує ефективну реалізацію моделі Generic Repository. Такий підхід забезпечує однаковий уніфікований доступ незалежно від типу та джерела даних [34]. Крім того, для забезпечення надійної взаємодії між різними компонентами програми використовується бібліотека Result: Бібліотека Result забезпечує просту і потужну реалізацію моделі Result, яка надає дані та повідомлення про помилки з їхніми описами у разі виникнення помилки Патерн Result є дуже корисним у програмуванні, забезпечуючи відмовостійку систему та зручний спосіб передачі інформації про помилки користувачеві. Різні реалізації цього патерну можуть також включати інформацію про коди стану, так звані "Коди стану". Ці коди стану відіграють важливу роль у веб-системах. Це пов'язано з тим, що всі відповіді HTTP-сервера містять ці коди і є основним засобом зв'язку між сервером і клієнтом. Завдяки цим кодам клієнт може по-різному реагувати на отриману

відповідь і відображати інформацію користувачеві найбільш ефективним способом.

MediatR працює у якості концентратора для поступаючих до нього запитів та переадресовує їх на відповідні обробники, тобто, ми завжди будемо працювати лише з одним екземпляром шару MediatR [35]. На рівні контролерів відбувається створення об’єкту запиту який надсилається в MediatR. Взаємодія цих компонентів схематично зображена на рисунку 2.3 нижче.

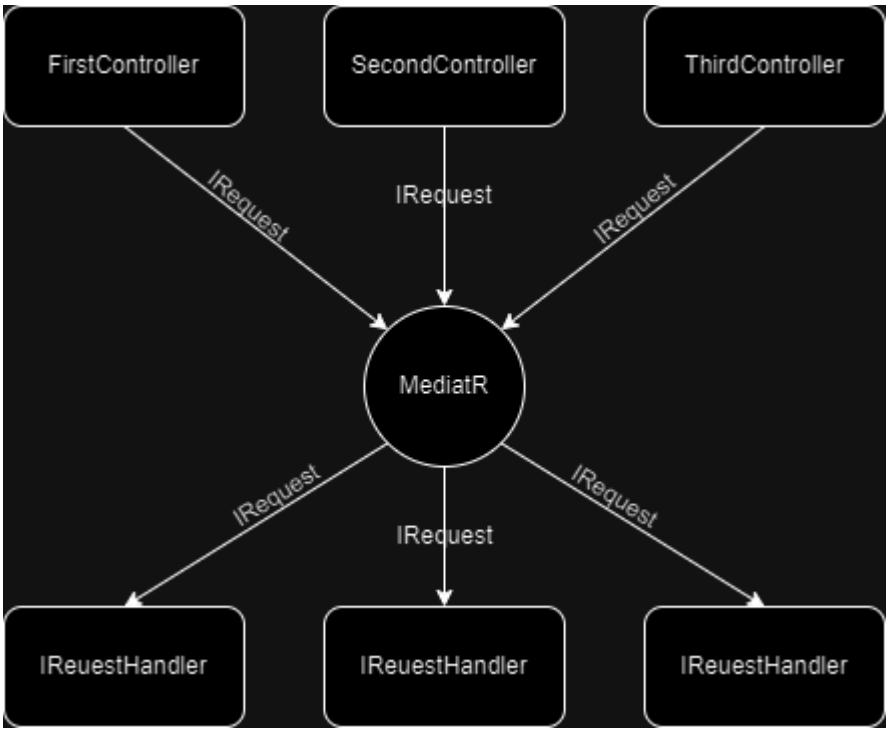
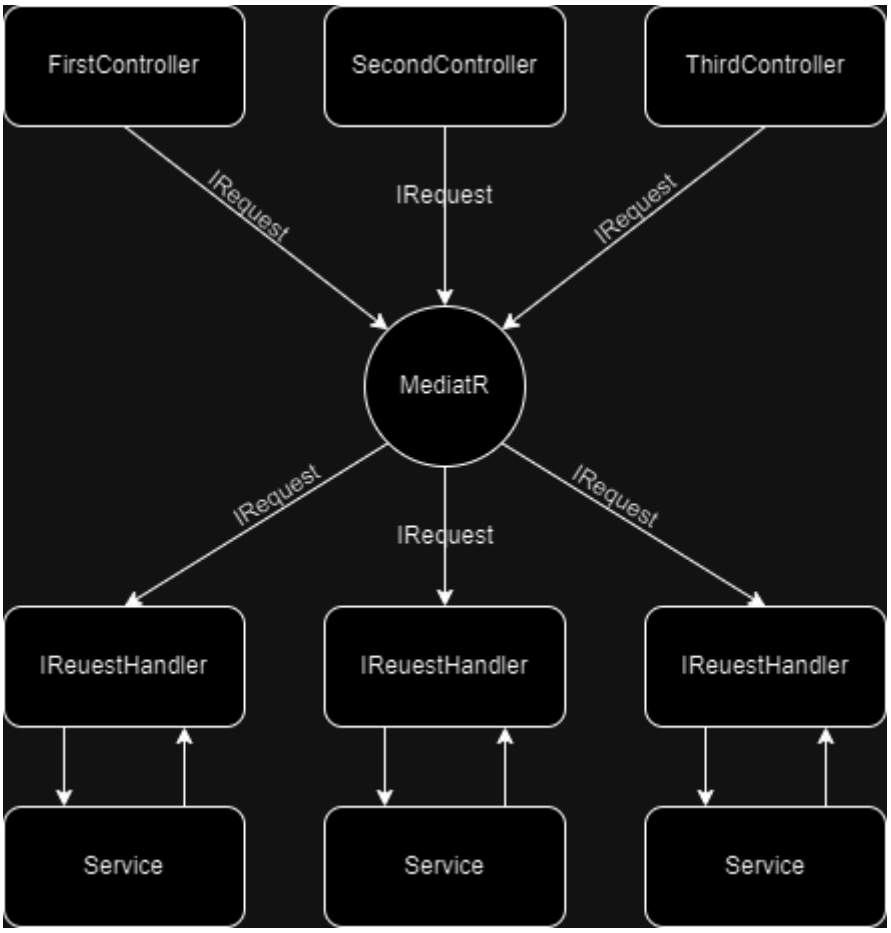


Рисунок 2.3 – Взаємодія контролерів із MediatR

Як у цьому випадку обробники повинні взаємодіяти із сервіним шаром? Кожний сервіс реалізує так званий сервісний клас. Це зв’язку інтерфейсу, який надає відповідні методи для обробки даних, та його реалізуючого класу. Використовуючи можливості вбудованого Dependency Injection конвейеру, ми можемо забезпечити максимально зручний функціонал для переадресації запитів в інші класи та сервіси. Схематично це зображено на рисунку 2.4, де відображається взаємодія між MediatR та деякими сервісами.



Риснок 2.4 – Взаємозв’язок MediatR та сервісних класів

Такий підхід для реалізації архітектури серверу є досить важким для підготовки та конфігурації, оскільки для забезпечення чистого коду та зрозумілої архітектури потрібно мати деякий досвід із роботою з подібними системами та їх розробкою. Забезпечення якісної архітектури це не проста задача, оскільки потрібне широке розуміння системи та ймовірні варіанти підходу до проєктування, задля цього ми використовуємо різні бібліотеки та реалізації патернів, як архітектурних, так і локальних.

Зв'язок між сервісним рівнем і базою даних визначає фундаментальний аспект архітектури програмного забезпечення. У цьому контексті фреймворк Entity Framework Core є справжнім витвором мистецтва. Цей фреймворк був створений, щоб спростити та оптимізувати роботу програмістів, які мають справу з базами даних, надаючи зручний та ефективний інтерфейс взаємодії. Завдяки Entity Framework Core розробники можуть ігнорувати деталі SQL-запитів і

маніпуляцій з даними і працювати з базами даних на більш високому рівні. рівень абстракції для роботи з базами даних. Це спрощує розробку та підтримку додатків за рахунок автоматизації багатьох рутинних завдань, таких як трансляція C#-коду у відповідні SQL-запити. Крім того, Entity Framework Core надає можливість використовувати концепцію "спочатку код", коли бази даних автоматично створюються на основі класів C# без необхідності вручну визначати безпосередньо схему бази даних [36]. Це дає розробникам більшу гнучкість в управлінні структурами баз даних і прискорює процес розробки. Entity Framework Core також забезпечує відображення даних між об'єктами C# і таблицями бази даних, дозволяючи розробникам працювати з даними більш інтуїтивно зрозумілим і зручним способом. Такий підхід дозволяє розробникам зосередитися на поліпшенні функціональності програми замість того, щоб витрачати час на вирішення технічних деталей, пов'язаних з базою даних.

Ще одним рівнем, який забезпечує безперебійну роботу програмного забезпечення та ефективне управління даними, є реалізація моделі Generic Repository. Інтеграція з ядром Entity Framework Core та використання Generic Repository виявилися дуже корисними для спрощення рутинних операцій з базами даних, а також для підвищення гнучкості та масштабованості процесу розробки. Переваги такого підходу полягають не лише у скороченні часу розробки, але й у зменшенні ймовірності помилок. Оскільки для кожного запиту до бази даних не потрібно писати однотипний код, ризик людської помилки знижується, що сприяє більшій одноманітності та узгодженості кодової бази. Завдяки публічному репозиторію розробники можуть швидше впроваджувати зміни та підтримувати систему. Варто також відзначити, що такий підхід дозволяє уникнути багатьох помилок і спрощує процес виявлення та виправлення системних проблем, які впливають на стабільність і надійність програмних продуктів. Схематичне зображення взаємодії сервісів з базою даних зображено на рисунку 2.5.

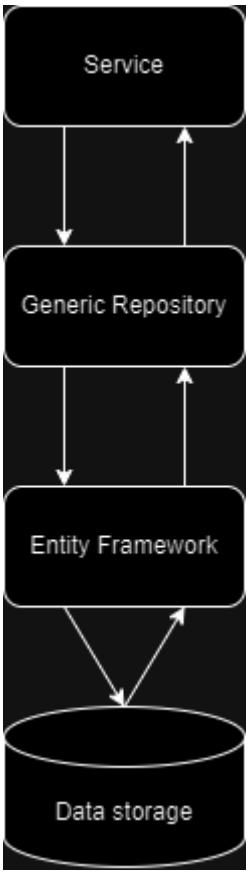


Рисунок 2.5 – Шар Generic Repository

Після успішного проєктування серверу, завершимо розділ проєктування клієнтської частини та розберемось, як вона взаємодіє із сервером.

2.3 Проєктування клієнтської частини

Проєктування клієнтської частини займає більшу частину з проєктування всієї системи, оскільки потребує професійних або спеціалізованих навичок з сфери розробки користувацьких інтерфейсів, тобто, UI/UX дизайн [37, 38]. Для спрощення роботи з клієнтською частиною в цьому проєкті буде використано мікросервісну архітектуру, де клієнтом буде інший проєкт Asp.NET Core MVC [39]. Проєктування клієнтської сторони включає в себе, окрім розробки користувацького інтерфесу, методи обробки запитів, більшість з яких буде виконуватись на стороні серверу MVC та авторизаційні методи, а також доступність тих чи інших дій відносно можливостей користувача. Основною

задачею клієнтської частини правильно корегувати інтерфейс користувача та набір відповідних дій, які він може виконувати. Для прикладу, користувач, який авторизований як робітник складу не може виконувати дії щодо вироблення товарів, перевірки їх якості, аналогічних або подібних дій щодо матеріалів тощо.

Для реалізації цього процесу в проєкті клієнтської сторони системи будуть наявні різні сервіси, зокрема сервіс авторизації який повинен отримувати актуального користувача від центрально API серверу. Центральний API сервер перенаправляє запит на Identity Server який, в свою чергу, працює з користувачами, їх персональними даними та має доступ до зв’язаних з ним таблицями, зокрема, місцем роботи. Первинна задумка полягає в отриманні повного користувача з зв’язаним з ним цехом або складом та перевірці дії, яка планується виконатись.

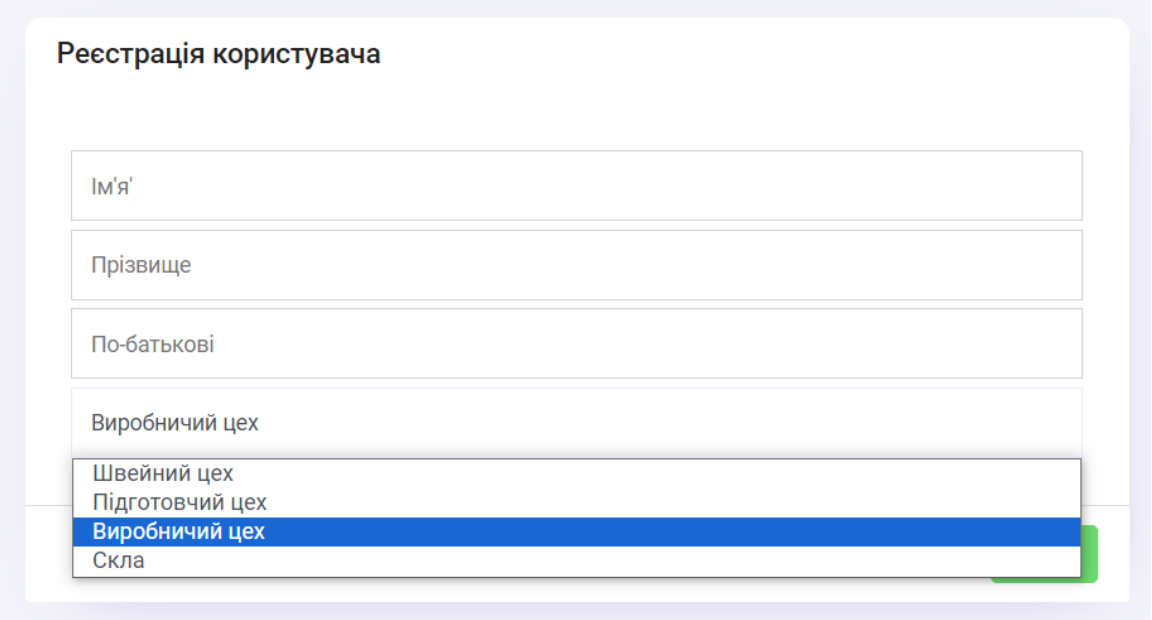


Рисунок 2.6 – Сторінка реєстрації

Проєктуваня клієнтської сторони також включає в себе проєктування користувацького інтерфейсу. Першим, що має побачити користувач, який не є авторизованим в системи це сторінка авторизації де користувача має можливість авторизуватися або зареєструватися. Реєстрація користувача здійснюється в два

кроки. Перший крок – це введення персональних даних користувача, таких як: ім’я, прізвище та вибір цеху в якому користувач бажає працювати. Відповідна сторінка зображена на рисунку 2.6. Реалізація інтерефейсу користувача буде розглянута в наступному розділі.

Процес авторизація та реєстрація досить схожий, оскільки вони використовують один процес авторизації користувача. Процес авторизації передбачає процес відправки запиту на сервер самого клієнта, оскільки клієнтська частина системи реалізована за допомогою фреймворку asp.net core mvc. Після цього, за допомогою методів мови програмування C#, виконується запит до серверу авторизації. Схема алгоритму запиту зображена на рисунку 2.7.

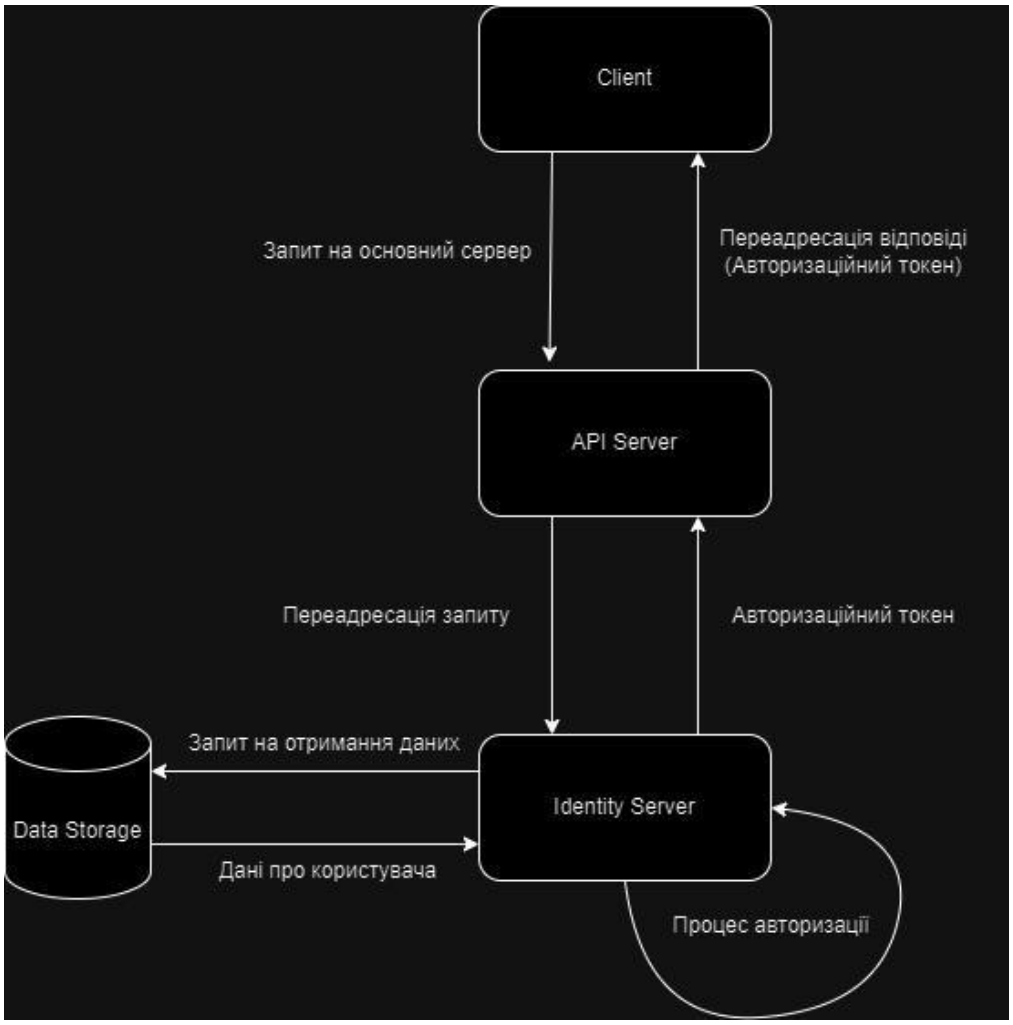


Рисунок 2.7 – Отримання токену при авторизації

Після авторизації користувач отримає доступ до основної системи та переадресовується на головну сторінку системи. Дана сторінка має деякі графічні користувацькі компоненти, зокрема, бічна панель навігації, верхня панель та головна частина. Голона частина головної сторінки містить діаграму ефективності авторизованого користувача. Після реєстрації діаграми мають найнижчі показники, але для користувача, який виконував роботу протягом деякого часу, діаграми будуть приблизними до рисунку 2.8. Також можилва наявність інших діаграм, зокрема діаграми щотижневого та щомісячного виробництва та інших. Дані діаграм реалізуються шляхом збору відповідної статистики.

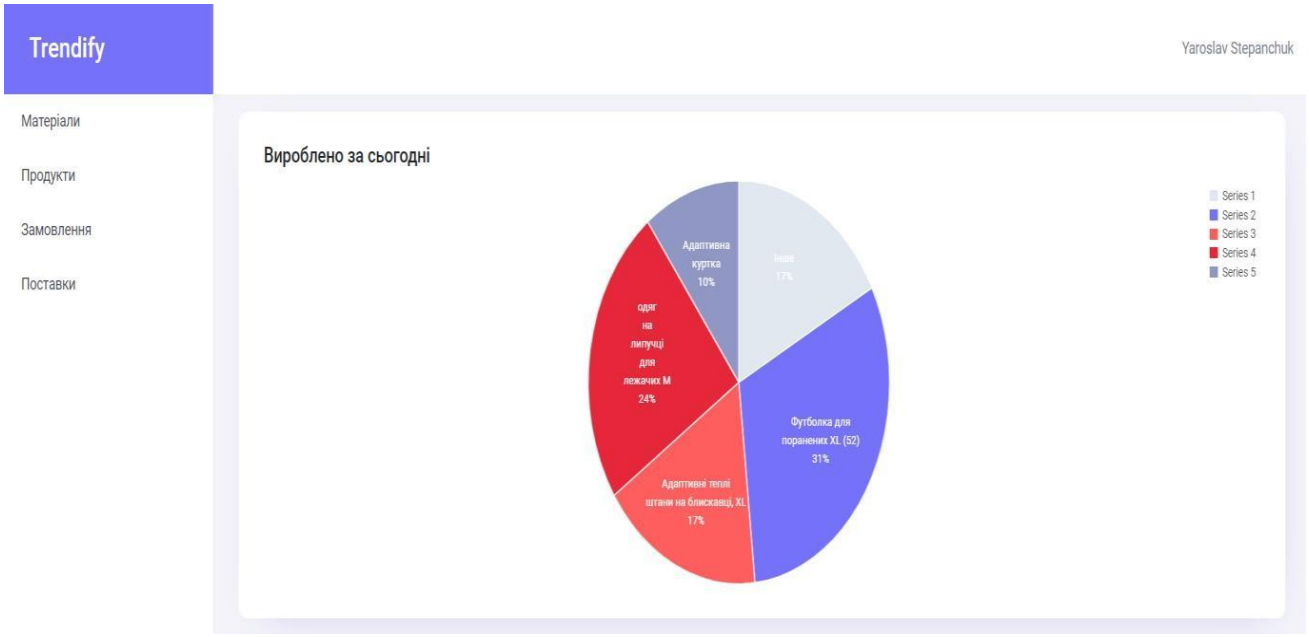


Рисунок 2.8 – Діаграма виробництва

Наявність та доступність інших сторінок повинна бути зумовлена можливість користувача робити або переглядати деякі дані відповідно до його місця роботи. На рисунку 2.8 зображено чотири пункти меню, які доступі для виробничого цеху, оскільки він має відповідні матеріали, яки використовуються для виробництва та самі товари, які були виготовлені в результаті деякої роботи. Для інших цехів даний набір може бути іншим. Для прикладу, підготовчий цех не має доступу до матеріалів, оскільки не може з ними працювати, а виконує лише

функцію перевірки готових товарів на наявність браку та інших нюансів. Кожна сторінка має відповідний функціонал відносно цеху, з якого робиться запит. Це повинно бути реалізовано за рахунок отримання даних та динамічної розмітки клієнта.

2.4 Висновок

У другому розділі детально пророблено проєктування ключових компонентів інформаційної системи для підприємства з виготовлення адаптивного одягу, використовуючи сучасні технології. Основна увага була зосереджена на створенні бази даних, серверної частини і клієнтського додатку, кожен з яких відіграє ключову роль у забезпеченні ефективної роботи підприємства.

Була розроблена структура бази даних, яка включала основні таблиці для зберігання інформації про матеріали, виробничі процеси, готову продукцію та управління запасами. Кожна таблиця містила необхідні атрибути та була взаємозв'язана з іншими таблицями за допомогою зовнішніх ключів. Це створило зручну та масштабовану структуру даних, яка дозволяє ефективно керувати виробничими процесами та забезпечити необхідну інформацію для управлінських рішень.

Для забезпечення взаємодії між базою даних і клієнтським додатком було використано ASP.NET Core Web API. Це рішення дозволяє створювати надійні, безпечні та ефективні API для обміну даними. Було розроблено RESTful API, яке підтримує операції CRUD (створення, читання, оновлення та видалення даних), що дозволяє клієнтському додатку ефективно взаємодіяти з базою даних через стандартні HTTP запити.

3 РЕАЛІЗАЦІЯ РОЗРОБЛЕНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Розробка бази даних

Розробка будь якої система розпочинається з етапну перенесення спроєктованного архітектурного плану на рівень коду. Розробка повинна виконуватись з самого низу архітектурного дерева до самої вершини, завершуючи рішення проблеми взаємодії кінцевого користувача з системою обробки даних. Виходячи із плану проєктування, ніжнім шаром системи являється шар бази даних, який використовується для збереження даних. Опис бази даних та важливість її наявності було описано в попередньому розділі, тому перейдемо до етапу розробки бази даних та розробки шару доступу та обробки даних.

В якості бази даних використовується база даних PostgreSql останньою версії [38-44]. Це безкоштовна база даних, яка не має обмежень по об’єму. Потрібно також зазначити, що розробка виконується на операційній системі Windows 10 останньої версії. Її можна встановити із офіційного сайту бази даних, вибравши платформу для розробки та відповідну версії, а також архітектуру системи. Після встановлення можемо переходити далі.

Наступний крок для переходу до процесу розробки це встановлення середовища розробки, налаштування бібліотек, залежностей, мови програмування, її версії, та відповідних фреймворків. Для цього потрібно встановити середовище розробки Visual Studio із офіційного сайту Microsoft. Для виконання завдання нам повністю підійде Visual Studio Community, яка є безкоштовною та не потребує активації, але якщо є можливість використання інших версій, вони також підійдуть для розробки. У процесі розробки буде використана Visual Studio Enterprice версії 17.8.0 [37]. На рисунку 3.1 виділено базові компоненти, які необхідні для розробки інформаційної системи.

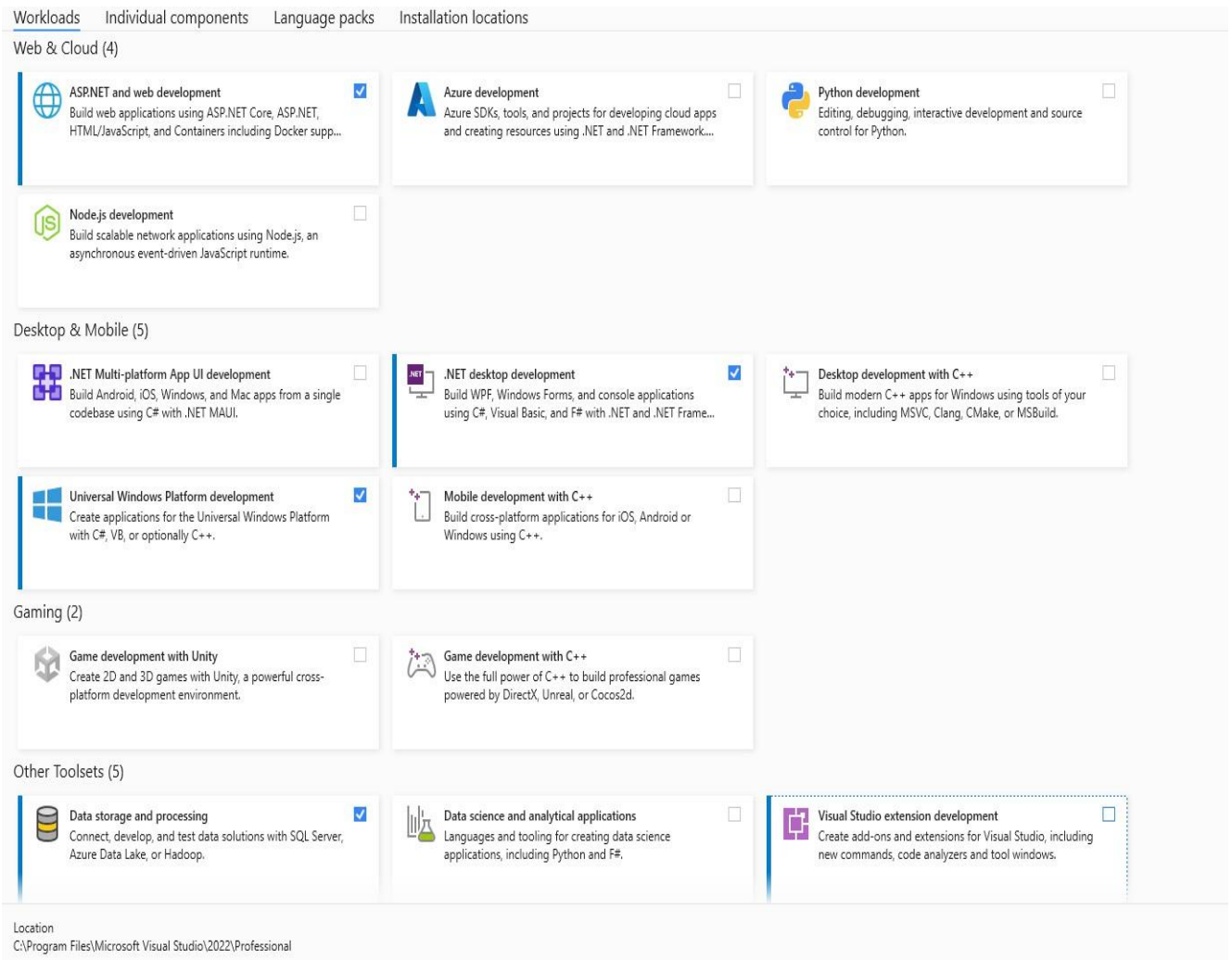


Рисунок 3.1 – Встановлені компоненти

Після встановлення середовиша, можна переходити до процесу створення та налаштування проєкту. Проєкт матиме назву Trendify, а для розробки використовуватиметься платформа .Net версії 8 [45]. До створеного проєкту додаємо бібліотку з префіксом Database що слугуватиме орієнтиром про призначення та зміст даної бібліотеки. Це робиться шляхом виклику контекстного меню правою кнопкою миші по рішенню та вибором відповідних пунктів меню як на рисунку 3.2 нижче.

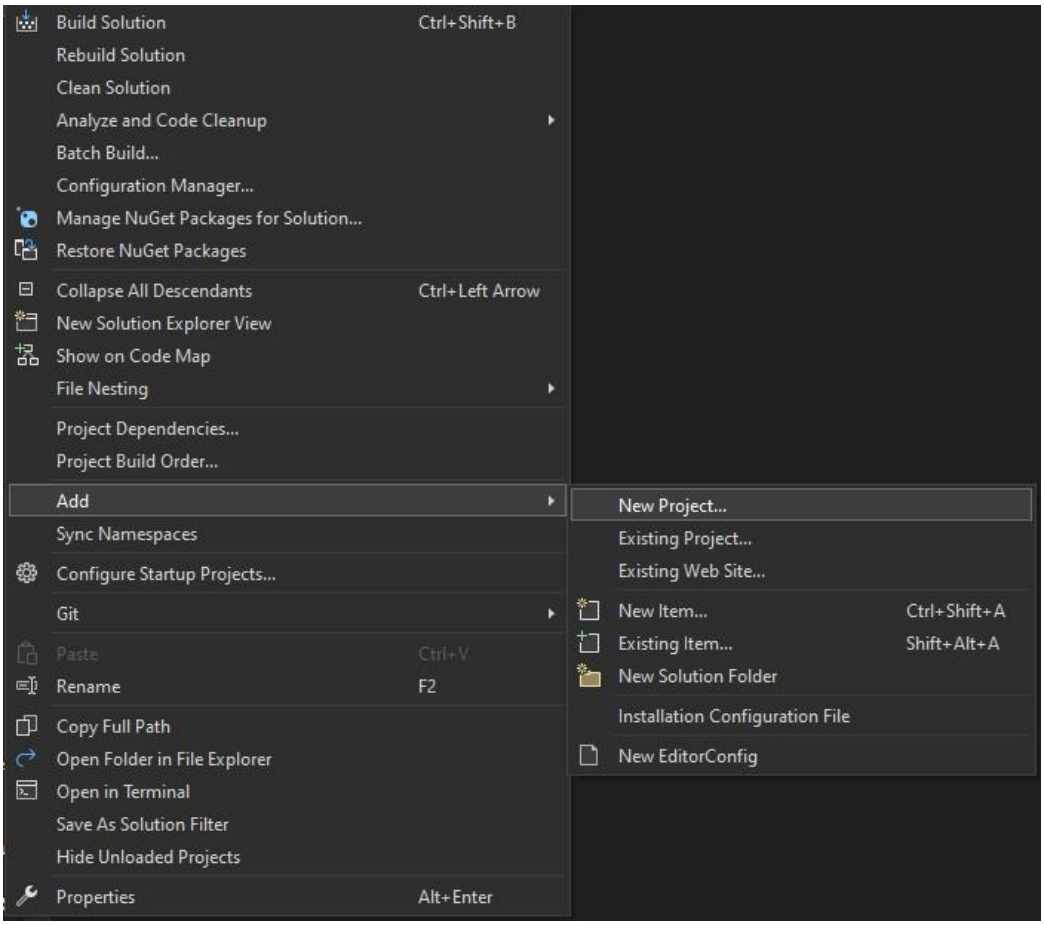


Рисунок 3.2 – Додавання нового прокту

Після вибору відповідного пункту потрібно знайти шаблон проєкт Class Library та дати йому відповідну назву. Після додавання проєкту визначимо архітектуру бібліотеки. Даний проєкт міститиме лише дві папки для розподілення програмних компонентів по логічним директоріям.

На даному етапі проєкт повнісю готовий виконувати свою функцію, а його головною функцією буде збереження сутностей, якими ми зможемо описати базу даних, її таблиці та зв’язки між ними. В попередньому розділи вже згадувалось про те, що для розробки доступу до бази даних буде використовуватись фреймворк Entity Framework Core 8. У контексті Entity Framework є термін «Сутність» [46]. Це звичайний .NET клас, який використовується для опису таблиці в базі даних та співставлення даних між базою даних та програмним типом даних. Винесимо загальні властивості всіх сутностей в окрему сутність під назвою BaseEntity, яка міститиме наступні властивості:

1. Id – має тип даних Guid та використовується для індексації запису в таблиці бази даних.

2. CreatedAt – має тип даних DateTimeOffset та використовується для позначення дати та часу створення (Додавання) запису в базу даних

3. UpdatedAt – має тип даних DateTimeOffset та використовується для позначення дати та часу останнього оновлення запису.

4. DeletedAt – має тип даних DateTimeOffset? та використовується для реалізації так званого м’ягкого видалення та позначення дати та часу видалення.

Важливо зазначити, що усі інші типи наслідуються від класу BaseEntity що забезпечує наявність вказаних властивостей для кожної сутності [47]. На етапі проєктування бази даних описувалась схема даних для бази даних на рисунку 2.1. Тепер, в ході розробки, доповнимо її відповідними таблицями для приведення бази даних у вигляд відповідний до рисунку 3.3.

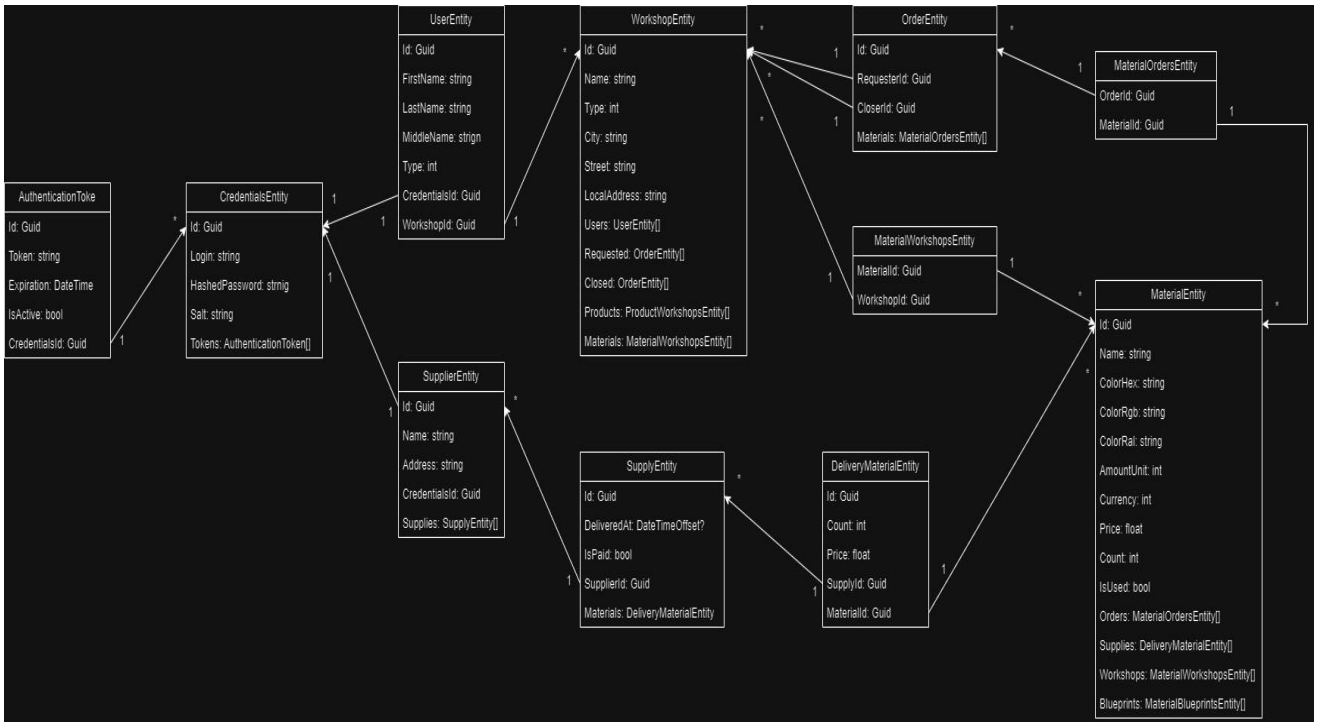


Рисунок 3.3 – Схема даних

Розробка шару бази даних включає в себе перенесення схеми даних на рівень коду, тому файлова стуктура матиме наступний вигляд, як на рисунку 3.4.

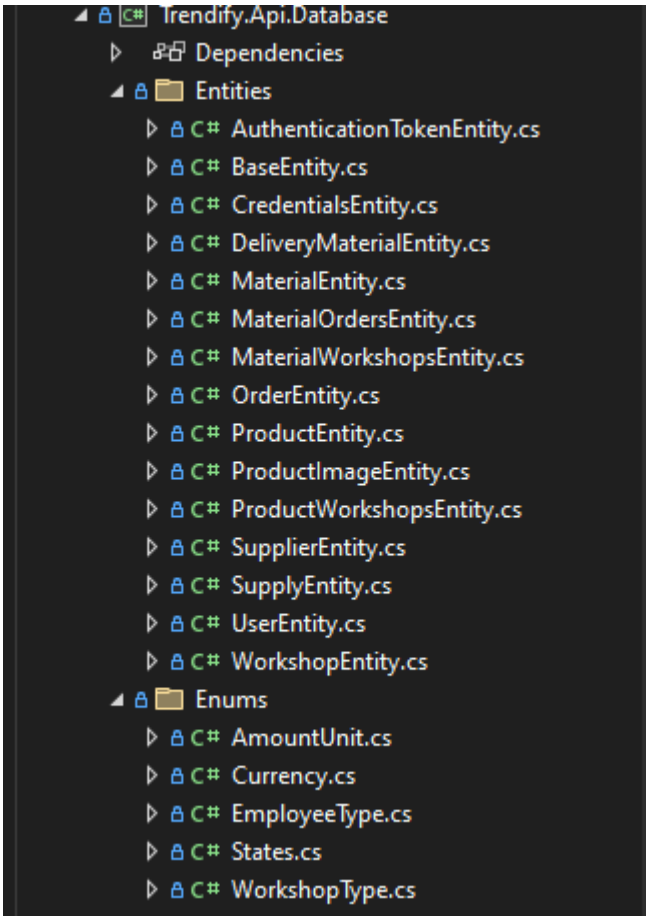


Рисунок 3.4 – Сутності бази даних

Усі вказані типи мають відповідну структуру до схеми даних та наслідування від базового типу BaseEntity. Ці типи використовуються для співставлення записів у базі даних та кодом на мові програмування C#.

Наступний етап розробки це шар доступу до даних. Так як було написано раніше, для доступу до даних в інформаційній системі використовується фреймвор Entity Framework Core 8. Це базовий фреймворк який використовує технологію ADO.NET для підключення та маніпуляцією даних та співставлення їх з відповідними типами [48]. Для конфігурації Entity Framework використовує базовий тип DbContext. Для налаштування бази даних та доступу до неї, при розробці шару доступу до даних, використовується наслідування типу DbContext та перевизначення методів конфігурації бази даних [49]. При розробці даної інформаційної системи використовується підхід винесення конфігурацій таблиць в окремі файли. Для реалізації окремої конфігурації, потрібно створити клас

відповідної конфігурації, який реалізує інтерфейс IEntityTypeConfiguration<T> [49]. Цей інтерфейс дозволяє налаштовувати різні аспекти конкретної таблиці в базі даних, зокрема такі властивості як:

– назва таблиці;

– первинний ключ таблиці;

– зовнішні ключі;

– звя’зки між таблицями;

– назви колонок таблиці;

– обмеження колонок таблиці.

На лістингу 3.1 зображено приклад конфігурації допоміжної таблиці типу при зв’язку багато до багатьох відношення між матеріалами по їх поставками DeliveryMaterialEntity.

Лістинг 3.1 – конфігурація DeliveryMaterialEntity

```
public sealed class DeliveryMaterialConfiguration : IEntityTypeConfiguration<DeliveryMaterialEntity>
{
   public void Configure (EntityTypeBuilder <DeliveryMaterialEntity> builder)
   {
      builder.ToTable("Delivery materials").HasKey(dm => new { dm.SupplyId, dm.MaterialId });

      builder.HasOne<MaterialEntity>(dm => dm.Material)
         .WithMany(material => material.Supplies)
         .HasForeignKey(dm => dm.MaterialId);

      builder.HasOne<SupplyEntity>(dm => dm.Supply)
         .WithMany(supply => supply.Materials)
         .HasForeignKey(dm => dm.SupplyId);
   }
}
```

На цьому лістингу можна побачити як відбувається наслідування та перевизначення методу Configure. Об’єкт тип EntityTypeBuilder дозволяє нам налаштовувати відповідні аспекти конфігурації таблиці, які були описані вище. Далі можна звернути увагу на два блоки коду які відповідають за налаштування зв’язку між таблицями. Дана таблицям виконує співтавлення між основними таблицями за рахунок зв’язку багато до багатьох відносно них. Розберемо основні методи та їх призначення.

Метод HasOne<T> вказує на те, що таблиця, відносно якої ми налаштовуємо конфігурацію зв’язку тає лише одне посилання на іншу таблицю. Це означає, що ми может мати один з зв’язків – один до одного або один до багатьох. Далі в дужках ми вказуємо на посилання на цей тип. Наступна стрічка це виклик методу WithMany який використовується для конкретизації зв’язку, а саме вказуючи на сторону відношення як «Багато». Тепер дана частина конфігурації забезпечує зв’язок багато до багатьох. Наступний рядок це виклик методу HasForeignKey, де нам потрібно вказати на властивість яка буде виступати зовнішнім ключем у цьому відношенні. Після цих дій конфігурація між таблицями вважається завершеною.

Але не завжди потрібно вказувати посилання на допоміжний зв’язок. Розглянемо приклад конфігурації зв’язку між сутностями CredentialsEntity та UserEntity. На лістингу 3.2 можна побачити конфігурації CredentialsEntity та на лістингу 3.3 конфігурацію UserEntity відповідно.

Лістинг 3.2 – Конфігурація CredentialsEntity

```
internal sealed class CredentialsConfiguration : IEntityTypeConfiguration<CredentialsEntity>
{
    public void Configure(EntityTypeBuilder<CredentialsEntity> builder)
    {
        builder.ToTable("Credentials").HasKey(credentials => credentials.Id);
    }
}
```

На цьому лістингу ми можемо побачити, що зв’язок не встановлюється явним чином, що може викликати деякі питання. Але якщо розглянути конфігурацію, відповідно, таблиці UserEntity, ми зможемо прийти до деяких висновків щодо зв’язку.

Лістинг 3.3 – Конфігурація UserEntity

```
public sealed class UserConfiguration : IEntityTypeConfiguration<UserEntity>
{
    public void Configure(EntityTypeBuilder<UserEntity> builder)
    {
        builder.ToTable("Users").HasKey(user => user.Id);
```

```
        builder.HasOne<WorkshopEntity>(user => user.Workshop)
            .WithMany(workshop => workshop.Users)
            .HasForeignKey(user => user.WorkshopId);

        builder.HasOne<CredentialsEntity>(user => user.Credentials)
            .WithOne()
            .HasForeignKey<UserEntity>(user => user.CredentialsId);
    }
}
```

На цьому лістингу нас цікавить лише блок налаштування зв'язку між даю сутністю та сутністю CredentialsEntity. Зверність увагу на другий рядок надого блоку коду, де при виклику методу WithOne не було вказано посилання на відповідний тип даних який використовується для зв'язку між типами. На лістингу 3.4 написано код самої сутності CredentialsEntity, яку ми можемо проаналізувати.

Лістинг 3.4 – Сутність CredentialsEntity

```
public sealed record CredentialsEntity : BaseEntity
{
    public string Email { get; set; } = string.Empty;
    public string Login { get; set; } = string.Empty;
    public string HashedPassword { get; set; } = string.Empty;
    public string Salt { get; set; } = string.Empty;
    public string Scope { get; set; } = string.Empty;

    public List<AuthenticationTokenEntity> AuthenticationTokens { get; set; } = new
List<AuthenticationTokenEntity>();
}
```

Можна звернути увагу на відсутність типу для зв'язку між таблицями. Чому так виходить і чому це допустимо при роботі з Entity Framework? Відповідь криється у самому способі роботи з базою даних. База даних зберігає лише примітивні типи даних у своїх таблицях, зокрема, первинні та зовнішні ключі для реалізації зв'язків між таблицями. Оскільки різні сутності мають також і різні типи .NET на рівні коду C#, не всі вони можуть відображатись у таблицях на рівні бази даних, зокрема, і посилання на інші таблиці. Замість цього ми можемо вказати лише зовнішні ключі, при конфігурації, та типи даних з якими потрібно робити співставлення даних з бази даних. Entity Framework автоматично конвертує C# код з SQL запити.

Після налаштування конфігурацій усіх сутностей, потрібно налаштувати головни клас для зв'язку з базою даних та її конфігурації. Цей тип повинен наслідуват базовий тип DbContext та відповідати певній структурі, яка описана в документації Entity Framework Core відповідно. На лістингу 3.5 описаний клас контексту бази даних, який використовується для розробки інформаційної системи.

Лістинг 3.5 – Контекст бази даних

```
public sealed class ApplicationDbContext : DbContext
{
    public DbSet<DeliveryMaterialEntity> DeliveryMaterials { get; set; }
    public DbSet<MaterialEntity> Materials { get; set; }
    public DbSet<MaterialWorkshopsEntity> MaterialWorkshops { get; set; }
    public DbSet<OrderEntity> Orders { get; set; }
    public DbSet<ProductEntity> Products { get; set; }
    public DbSet<ProductImageEntity> ProductImages { get; set; }
    public DbSet<ProductWorkshopsEntity> ProductWorkshops { get; set; }
    public DbSet<SupplierEntity> Suppliers { get; set; }
    public DbSet<SupplyEntity> Supplies { get; set; }
    public DbSet<UserEntity> Users { get; set; }
    public DbSet<WorkshopEntity> Workshops { get; set; }
    public DbSet<CredentialsEntity> Credentials { get; set; }
    public DbSet<AuthenticationTokenEntity> AuthenticationTokens { get; set; }
    public DbSet<MaterialOrdersEntity> MaterialOrders { get; set; }

    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
    {
        Database.Migrate();
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.ApplyConfiguration(new DeliveryMaterialConfiguration());
        modelBuilder.ApplyConfiguration(new MaterialConfiguration());
        modelBuilder.ApplyConfiguration(new MaterialWorkshopsConfiguration());
        modelBuilder.ApplyConfiguration(new OrderConfiguration());
        modelBuilder.ApplyConfiguration(new ProductConfiguration());
        modelBuilder.ApplyConfiguration(new ProductImageConfiguration());
        modelBuilder.ApplyConfiguration(new ProductWorkshopsConfiguration());
        modelBuilder.ApplyConfiguration(new SupplierConfiguration());
        modelBuilder.ApplyConfiguration(new SupplyConfiguration());
        modelBuilder.ApplyConfiguration(new UserConfiguration());
        modelBuilder.ApplyConfiguration(new WorkshopConfiguration());
        modelBuilder.ApplyConfiguration(new CredentialsConfiguration());
        modelBuilder.ApplyConfiguration(new AuthenticationTokenConfiguration());
        modelBuilder.ApplyConfiguration(new MaterialOrdersConfiguration());
    }
}
```

З наведеного вище коду, можна зробити деякий висновок щодо структури контексту бази даних усі типи, які повинна являти собою таблиці бази даних, повинні бути «обгорнуті» типом DbSet<T>, що забезпечує розуміння фреймворка щодо таблиці та їх структури. У самому низу перевизначено метод OnModelCreating, у якому, за допомогою типу ModelBuilder використовуються конфігурації, які були реалізовані раніше. Конструктор контексту приймає тип DbContextOptions в якості параметру, що надає можливість для передачі опцій контексту ззовні. Це потрібно для реалізації доступному контексту бази даних за допомогою контейнеру ін'єкції залежностей, реалізації якого буде представлено згодом.

Передостаннім кроком є реєстрація контексту в, названому раніше, контейнері ін'єкції залежностей. Для цього у відповідному файлі Program.cs потрібно звернутсь до типу IServiceCollection та викликати метод AddDbContext<T>, який приймає параметром опції для налаштування контексту бази даних, як це представлено на лістингу коду 3.6.

Лістинг 3.6 – Реєстрація контексту бази даних

```
services.AddDbContext<ApplicationDbContext>(options =>
{
   options.UseNpgsql(builder.Configuration.GetSection("ConnectionStrings:DefaultConnection").Value);
});
```

При передачі об'єкту опцій використовується метод UseNpgsql, який забезпечує доступ до конкретної СУБД, а саме PostgreSql. Цей метод являється методом розширення та підключається NuGet пакетом Npgsql.EntityFrameworkCore.PosgreSQL, який встановлений на проєкт EntityFramework.

Після реєстрації контексту бази даних, останнім етапом є міграція бази даних, що забезпечить процес її створення та доступу до неї. Для використання цієї функції потрібно відкрити відповідну консоль яка має назву «Консоль управління пакетами» та доступна за вказаним шляхом, як на рисунку 3.5.

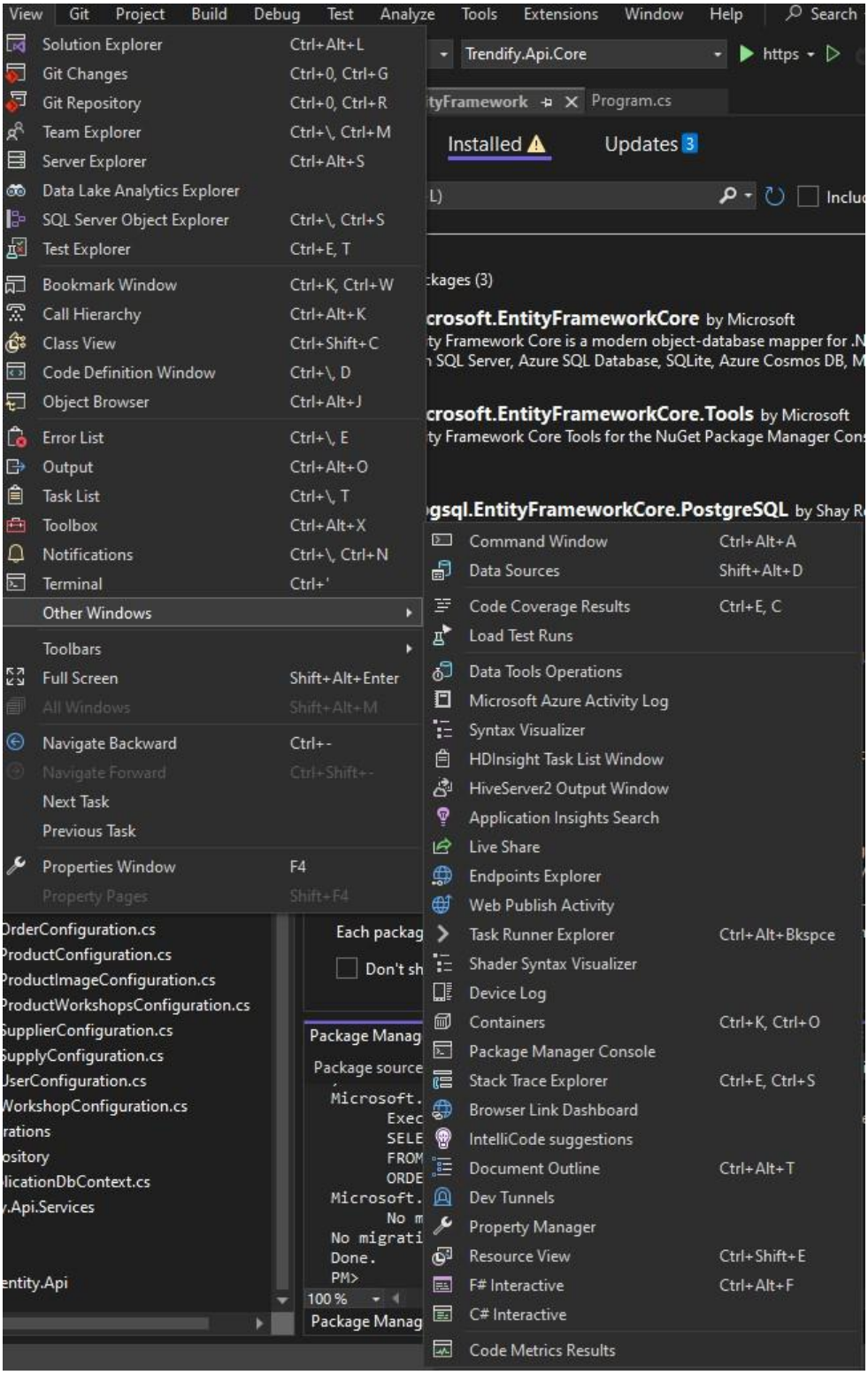


Рисунок 3.5 – Консоль управління пакетами

Для виконання міграції у консолі потрібно написати команду add-migration «Migration» де Migration є назвою міграції. Потірбно зауважити, що назва міграції

повинна мати назву одним словом та нести сенс її використання. Також, у вікні «Проєкт за замовчування» потрібно вибрати саме той проєкт, у якому реалізовано контекст бази даних, як це зображено на рисунку 3.6.

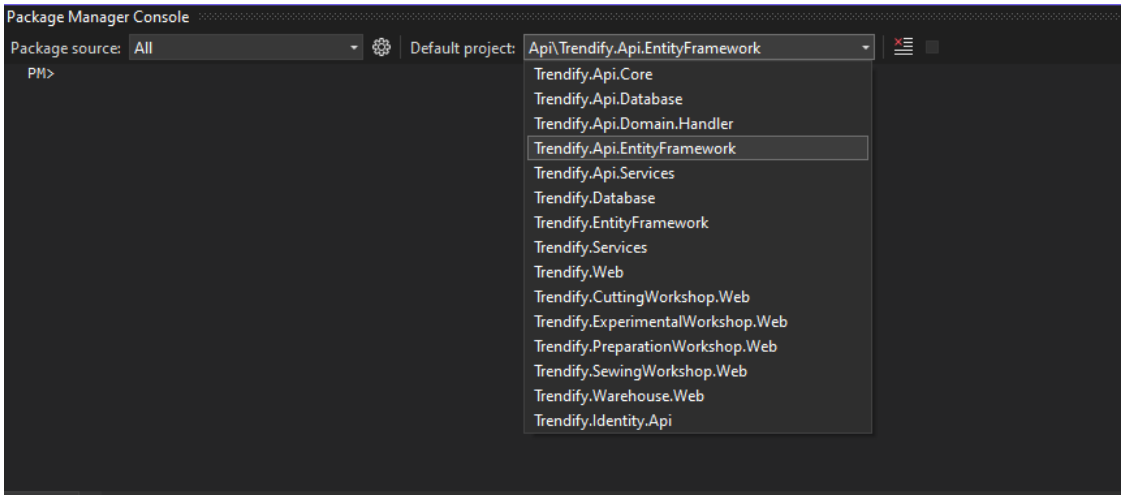


Рисунок 3.6 – Проєкт для міграції

Після виконанян міграції, у вибраному проєкт буде створена директорія з назвою Migrations, в яку будуть додаватись усі міграції для реалізації поточного стану бази даних. Останнім етапом є виконання команди update-database, що забезпечить оновлення наявної бази даних відповідно до наявних міграції. Міграції містять автогенеровний код, який не потрібно редагувати вручну, а лише з використанням спеціальних команд Entity Framework. Команди для виконання міграції та оновлення бази даних підключаються пакетом Microsoft.EntityFrameworkCore.Tools до відповідного проєкту.

3.2 Реалізація серверної частини

Сервер – це мозок будь якої системи включаючи дану інформаційну систему. Сервер може використовувати різні стороні або користувацькі бібліотеки, які містять деякі готові рішення. У попередньому розділі було розглянути розробку шару даних та шару доступу до даних. У цьому розділі буде

розглянуто розробку шару обробки та маніпулації даними, шар сервісів та розширень, а також шар веб-доступу до ресурсів. Розпочнемо з оптимізації зручності процесу розробки, а саме, шару доступу до даних. Оптимізація полягає в реалізації патерну Generic Repository. Для обмеження типів виберемо тип BaseEntity. Реалізація представлена на лістингу 3.7.

Лістинг 3.7 – Реалізація паттерну.

```
public sealed class GenericRepository<T> : IGenericRepository<T> where T : BaseEntity
{
    private readonly ApplicationDbContext _context;
    private readonly DbSet<T> _table;

    public GenericRepository(ApplicationDbContext context)
    {
        _context = context;
        _table = _context.Set<T>();
    }

    public async Task Create(T entity, CancellationToken cancellationToken = default)
    {
        entity.Id = Guid.NewGuid();
        entity.CreatedAt = DateTime.Now.ToUniversalTime();
        entity.UpdatedAt = DateTime.Now.ToUniversalTime();

        await _table.AddAsync(entity, cancellationToken);
        await _context.SaveChangesAsync();
    }

    public async Task DeleteSoft(T entity)
    {
        entity.DeletedAt = DateTime.Now.ToUniversalTime();

        _table.Update(entity);
        await _context.SaveChangesAsync();
    }

    public IQueryable<T> GetAll() =>
        _table.Where(entity => !entity.DeletedAt.HasValue).AsNoTracking();

    public IQueryable<T> GetAllBy(Expression<Func<T, bool>> predicate) =>
        _table.Where(entity => !entity.DeletedAt.HasValue)
            .Where(predicate)
            .AsNoTracking();

    public async Task<T?> GetBy(Expression<Func<T, bool>> predicate, CancellationToken cancellationToken
= default) =>
        await _table
            .Where(entity => !entity.DeletedAt.HasValue)
            .FirstOrDefaultAsync(predicate, cancellationToken);

    public async Task<T?> GetById(Guid id, CancellationToken cancellationToken = default) =>
        await _table.FirstOrDefaultAsync(entity =>
            entity.Id == id &&
```

Продовження лістингу коду 3.7.

```
            !entity.DeletedAt.HasValue,
            cancellationToken);

    public async Task Update(T entity)
    {
        _table.Update(entity);
        await _context.SaveChangesAsync();
    }

    public async Task DeleteHard(T entity)
    {
        _table.Remove(entity);
        await _context.SaveChangesAsync();
    }

    public async Task UpdateRange(List<T> entities)
    {
        _table.UpdateRange(entities);
        await _context.SaveChangesAsync();
    }
}
```

Основна ідея реалізації полягає у наявності методу Set<T> класу DbContext, який повертає об’єкт DbSet<T>. Це дозволяє викликати реалізації таблиці вказаного типу. Це являється дуже зручним методом, який дозволяє реалізувати патерн Generic Repository. Ключовою особливістю даної реалізації є обмеження по типу узагальненого типу BaseEntity. Ключовою особливістю такого обмеження стає доступ до загальних полів базової сутності BaseEntity, що дозволяє нам автоматично ініціювати ідентифікатор, тобто, первинний ключ, та дати створення та оновлення сутності при виклику методу створення. Аналогічні дії ми можемо проводити під час виклику методів оновлення сутності та видалення. Метод для видалення сутності має особливість в тому, що він не проводить фактичного видалення, а лише ставить так звану мітку видалення симулюючи видалення. Такий тип видалення має назву м’ягке видалення або «Soft Delete». Результат цього ми можемо бачити при виклику методів GetById або GetAll, де перед поверненням значення робиться вибірка на виключення записів, у яких властивість DeletedAt була встановлена.

В рішенні також присутній проєкт з префіксом .Services який відповідає за різну обробку даних, управління даними та інші допоміжні методи роботи над

проєктом. У цьому наявні дві директорії із відповідними типами розширення, які описані нижче. Розпочнемо огляд з типу Result, лістинг коду якого представлено на лістингу 3.8.

Лістинг 3.8 – Тип Result

```
public sealed class Result
{
    public Error Err { get; }
    public string? ErrorMessage => Err?.ErrorMessage;
    public bool IsError => !string.IsNullOrEmpty(ErrorMessage);

    private Result(string errorMessage) => Err = Response.Error.New(errorMessage);
    private Result(Error error) => Err = Response.Error.Copy(error);
    private Result(Exception ex) => Err = Response.Error.New(ex);

    public static Result Success() => new(string.Empty);
    public static Result Error(string errorMessage) => new(errorMessage);
    public static Result Error(Error error) => new(error);
    public static Result Error(Exception ex) => new(ex);
}

public sealed class Result<T>
{
    public Error Err { get; }
    public string? ErrorMessage => Err?.ErrorMessage;
    public bool IsError => !string.IsNullOrEmpty(ErrorMessage);
    public T Value { get; }

    private Result(T value) => Value = value;
    private Result(string errorMessage) => Err = Response.Error.New(errorMessage);
    private Result(Error error) => Err = Response.Error.Copy(error);
    private Result(Exception ex) => Err = Response.Error.New(ex);

    public static Result<T> Success(T value) => new(value);
    public static Result<T> Error(string errorMessage) => new(errorMessage);
    public static Result<T> Error(Error error) => new(error);
    public static Result<T> Error(Exception ex) => new(ex);
}
```

Основна ідея даного класу полягає в тому, що будь яка обробка даних має лише два стани – успіх та помилка. Проблема полягає у тому, що помилка виконання може відрізнятись в залежності від типу обробки та її причини, а це, в свою чергу, впливає на користувацьку помилку. Обробку цієї ситуації забезпечує наявність властивості IsError, але залишається проблема в описі помилки. Для прикладу, розглянемо ситуацію, коли нам потрібно зареєструвати користувача, але помилки можуть бути різні, такі як неправильно введені логін, пароль, номер телефону, електрона адреса тощо. Для вирішення цієї проблеми даний клас має

властивість ErrorMessage, яка використовуєтся для збереження повідомлення про саму помилку. Також на лістингу видно використання типу Error. Це користувацький клас, який використовується для більш детального опису помилки.

Одним із основних класів розширення який найчастіше використовується при розробці системи є клас із назвою Mapper [42], реалізацію якого можна побачити на лістингу 3.9.

Лістинг 3.9 – Метод розширення

```
public static class Mapper
{
    public static async Task<IEnumerable<TOut>> Map<TIn, TOut>(this Task<IEnumerable<TIn>> asyncResult,
Func<TIn, TOut> mapper)
    {
        IEnumerable<TIn> list = await asyncResult;
        return list.Select(mapper);
    }

    public static async Task<TOut> Map<TIn, TOut>(this Task<TIn> asyncResult, Func<TIn, TOut> mapper)
    {
        TIn result = await asyncResult;
        return mapper(result);
    }

    public static TOut Map<TIn, TOut>(this TIn result, Func<TIn, TOut> mapper) => mapper(result);
}
```

Це надзвичайно зручна і водночас проста реалізація для швидкої та лаконічної обробки результатів, яку потім буде представлено не один раз. Інші аспекти системи будуть розглянуті в ході потреби. В процесі розробки будуть розглянути основні аспекти, щоб не зациклюватись по фрагментах, які можуть повторювати багаторазово.

Робота системи заключається в забезпеченні виготовлення адаптивного одягу з деяких матеріалів. Спершу, опишемо процес появи матеріалів в системі. Загалом, системи повинна працювати з уже готовими матеріалами, тому ми повинні надати їй можливіть їх реєстрації в середині системи. На верхніх рівня серверу таку можливість надає відповідний контроллер MaterialController. На

лістингу 3.10 представлено код цього контролеру для більш детального ознайомлення.

Лістинг 3.10 – Контроллер матеріалів

```
[IdentityAuthorize]
[Route(MaterialControllerRoute)]
public class MaterialController(IMediator mediator) : BaseController(mediator)
{
    [HttpPost(RegisterBaseNew)]
    public async Task<IActionResult> Create([FromBody] RegisterMaterialApiRequest request,
        CancellationToken cancellationToken = default) =>
        await SendRequest(new RegisterNewMaterialRequest(request.Name), cancellationToken)
            .Map(result => result.IsError ?
                AsError(result.ErrorMessage!) :
                AsSuccess(result.Value));

    [HttpGet(GetByIdBaseRoute)]
    public async Task<IActionResult> GetById([FromRoute]Guid id, CancellationToken cancellationToken =
default) =>
        await SendRequest(new GetMaterialByIdRequest(id), cancellationToken)
            .Map(result => result.IsError ?
                AsError(result.ErrorMessage!) :
                AsSuccess(result.Value));

    [HttpGet(GetAllBaseRoute)]
    public async Task<IActionResult> GetAll(CancellationToken cancellationToken = default) =>
        await SendRequest(new GetMaterialsRequest(), cancellationToken)
            .Map(list => list.Any() ?
                AsSuccess(list) :
                AsSuccess());

    [HttpDelete(DeleteBaseRoute)]
    public async Task<IActionResult> Delete([FromRoute] Guid id, CancellationToken cancellationToken =
default) =>
        await SendRequest(new DeleteMaterialRequest(id), cancellationToken)
            .Map(result => result.IsError ?
                AsError(result.ErrorMessage!) :
                AsSuccess());
}
```

Тут потрібно виділити деякі ключові аспекти, які не було описані раніше та являють виключно користувацьким доповненням, тобто, атрибути HttpGet, HttpPost, FromRoute та інші, а також типи Task<T>, IActionResult, Guid та інші не будуть розглядатись, оскільки вони являються базовими типами для фреймворку asp.net core web api та мови програмування C#.

Даний контролер наслідуються від базового контролеру BaseController, лістинг коду якого представлено в лістингу 3.11. У цей контролер винесено основі методи обробки даних перед їх поверненням клієнту. Усі маршрути винесені в

окремі константи для уникнення такого явища як «Магічні рядки». Серед методів повернення результату є декілька перевантажених методів для забезпечення комфортної роботи з ними та передбачення усіх можливих варіантів використання [41].

Лістинг 3.11 – Базовий контроллер

```
[ApiController]
public abstract class BaseController(IMediator mediator) : ControllerBase
{
    protected const string SupplierControllerRoute = "api/supplier";
    protected const string UpdateSupplierNameRoute = "{supplierId:guid}/update/name";
    protected const string UpdateSupplierAddressRoute = "{supplierId:guid}/update/address";
    protected const string RemoveSupplierRoute = "{supplierId:guid}/remove";

    protected const string SupplyControllerRoute = "api/supplier/{supplierId:guid}/supply";
    protected const string AppendMaterialFromSupplyRoute = "{id:guid}/materials/append";
    protected const string RemoveMaterialFromSupplyRoute = "{id:guid}/materials/remove";
    protected const string CompleteSupplyRoute = "{id:guid}/complete";
    protected const string PaySupplyRoute = "{id:guid}/pay";

    protected const string MaterialControllerRoute = "api/material";

    protected const string WorkshopControllerRoute = "api/workshop";
    protected const string WorkshopMaterialsRoute = "{id:guid}/materials";
    protected const string UpdateWorkshopNameRoute = "{id:guid}/update/name";
    protected const string UpdateWorkshopInfoRoute = "{id:guid}/update/info";
    protected const string RemoveWorkshopRoute = "{id:guid}/remove";

    protected const string OrderControllerRoute = "api/workshop/{workshopId:guid}/order";

    protected const string AuthenticationControllerRoute = "api/authentication";
    protected const string SignInRoute = "sign-in";
    protected const string SignUpRoute = "sign-up";

    protected const string NewBaseRoute = "new";
    protected const string DeleteBaseRoute = "{id:guid}/delete";
    protected const string RegisterBaseNew = "register-new";
    protected const string GetAllBaseRoute = "get/all";
    protected const string GetByIdBaseRoute = "get/{id:guid}";

    protected async Task<T> SendRequest<T>(IRequest<T> request, CancellationToken cancellationToken =
default) =>
        await mediator.Send(request, cancellationToken);

    protected object ToErrorResponse(string errorMessage) => new { errorMessage = errorMessage };

    protected IActionResult AsError(object responseObject) =>
        BadRequest(responseObject);
```

Продовження лістингу коду 3.11.

```
    protected IActionResult AsError(string errorMessage) =>
        BadRequest(ToErrorResponse(errorMessage));
```

```
    protected IActionResult AsNotFound(string errorMessage) =>
        NotFound(ToErrorResponse(errorMessage));


    protected IActionResult AsSuccess(object value) => Ok(value);

    protected IActionResult AsSuccess() => NoContent();
}
```

Конструктор цільового контролера приймає об'єкт IMediator. Даний інтерфейс встановлюється бібліотекою MediatR. Даний пакет включає в себе програмну реалізацію патерну Медіатор. Його наявність має широкий вплив на всю систему, але основне місце обробки даних знаходить в іншій бібліотеці, реалізацію якої буде розглянуто пізніше. На рівень базового контролеру винесено метод для спрощення виклику медіатора, це метод можна побачити в лістингу 3.11 вище. При надходженні запиту контролер викликає відповідний метод, куди передає запит, тобто, реалізацію інтерфейсу IRequest. Усі подібні реалізації реалізовані в бібліотеці з префіксом Domain.Handler. Відповідь від медіатора аналізується викликом методу Map<TIn, TOut> який є методом користувацьким розширення, призначеним спеціально для аналізу подібних результатів. Тип повртаємого значення медіатором є Result<T>, який було описано раніше у цьому розділі. В залежності від стану результату робиться висновок щодо значення, яке потрібно повертати клієнту, тобто, виклик відповідного методу із BaseController.

Методи для обробки даних за допомогою MediatR мають назву обробники запитів кожен з яких має в залежностях об'єкт IGenericRepository для зв'язку з базою даних. На лістингу 3.12 представлено приклад такого обробника для реєстрації матеріалу в системі.

Лістинг 3.12 – Обробник реєстрації матеріалу

```
public sealed class RegisterNewMaterialHandler(
    IGenericRepository<MaterialEntity> repository,
    ILogger<RegisterNewMaterialHandler> logger)
    : IRequestHandler<RegisterNewMaterialRequest, Result<Guid>>
```

Продовження лістингу коду 3.12.

```
{
    private const string WasCreatedError = "Material was created.";
```

```
    public async Task<Result<Guid>> Handle(RegisterNewMaterialRequest request, CancellationToken
cancellationToken)
```

Продовження лстингу коду 3.12.

```
    {
        if (await repository.GetBy(entity => entity.Name == request.Name)
                .Map<MaterialEntity, bool>(entity => entity is not null))

        {
            return Result<Guid>.Error(WasCreatedError);
        }

        MaterialEntity entity = new MaterialEntity()
        {
            Name = request.Name,
        };

        try
        {
            await repository.Create(entity);
        }
        catch (Exception ex)
        {
            logger.LogError($"Cant' register new material: {ex.Message}");
            return Result<Guid>.Error(ex);
        }
        return Result<Guid>.Success(entity.Id);
    }
}
```

Опираючись на вище представлениий код, можна зробити висновок щодо простоти обробки даних та легкість архітектурного рішення, зокрема, завдяки наявності бібліотеки MediatR та користувацьким методам розширення. Наразі, це є основною логікою обробки даних та взаємодії із системою обробки даних. Далі будуть розглянуті методи для виконання основної логіки роботи програми.

3.3 Реалізації клієнтської частини

Для реалізації клієнтської частини інформаційної системи було використано фреймворк asp.net core mvc. Даний вибір обгрунтований в першу чергу простотою створення користувацького інтерфейсу за рахунок особливості asp.net core mvc – шаблонізації. Шаблонізація дозволяє розділяти розробку різних частин користувацького інтерфейсу під час розробки та багаторазово використовувати його під час роботи з сторінками.

Для реалізації користувацього інтерфейсу використовується CSS фреймворк Bootstrap версії 5.3, яка являється останньою версією на момент розробки та написання кваліфікаційної роботи. Для підключення цього фреймворку до проєкту використовується технологія CDN, яка дозволяє використовувати різні бібліотеки та стилі, які розміщені віддалено вказавши посилання на них. Для цього потрібно з відповідної сторінки на офіційному сайті взяти вілповідне посилання та використати його наступним чином, як це показано на рисунку 3.7.

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="utf-8" />
5       <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6       <title>@ViewData["Title"] - Trendify.Client</title>
7       <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
8       <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
9       <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
10      <link rel="stylesheet" href="~/Trendify.Client.styles.css" asp-append-version="true" />
11  </head>
12  <body>
13      <header>…</header>
34      <div class="container">…</div>
39
40      <footer class="border-top footer text-muted">
41          <div class="container">
42              &copy; 2024 - Trendify.Client - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
43          </div>
44      </footer>
45      <script src="~/lib/jquery/dist/jquery.min.js"></script>
46      <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
47      <script src="~/js/site.js" asp-append-version="true"></script>
48      @await RenderSectionAsync("Scripts", required: false)
49  </body>
50  </html>
```

Рисунок 3.7 – Підключення Bootstrap

На рядках 8, 46 та 47 видно зміни в коді, які відрізнаються від стандартного шаблону. Стандартний шаблон asp.net core 8 mvc має стандартну бібліотеку, яка йде «З коробки», але вона не завжди являється останньою версією. У цьому випадку, на момент написання кваліфікаційної роботи, фреймворк asp.net core 8 mvc підключає Bootstrap версії 5.1 за замовчуванням. Після додавання усіх стилів, після запуску проєкту, головна сторінка проєкта виглядатиме як на рисунку 3.8.

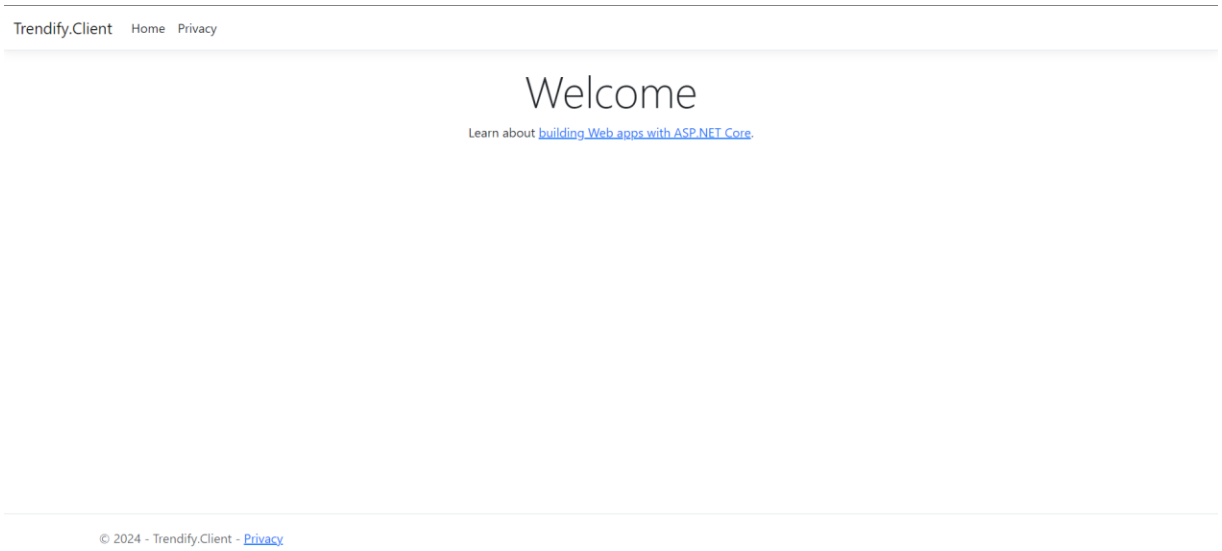


Рисунок 3.8 – Головна сторінка

Для реалізації користувацького інтерфейсу використаємо безкоштовний Bootstrap шаблон типу адміністративної панелі щоб спросити процес написання ромітки для користувацього інтерфейсу. З цього шаблону нам потрібно скопіювати деякі деталі, які можуть бути корисними при реалізації користувацього інтерфейсу. Для початку роботи скопіюємо з відповідного шаблону бокову напель на верхню панель та переробимо їх відповідні до потореби системи. Спершу потрібно розбити усі частини шаблону на відповідні макети та зв’язати їх між собою. Усі стилі винесені в макет _Layout, загальні частини, які вляється спільними для більшості сторінок, оскрім сторінок реєстрації та авторизації, в макет _HeaderLayout.

Макета _Layout та _HeaderLayout використвуються для механізму шаблонізації. Даний механізм передбачає розбиття загальних частин HTML розмітки на окремі файли для повторного використання. Загальні частини, які являються спільними для більшості сторінок мають назву «Макети», а їх відповідні файли мають приставку у вигляді символу нижного підкреслення. Використання макетів для певної сторінки вказується з самого зверху.

Процес авторизації частково виконується за рахунок використання методів ajax бібліотеки jQuery. На лістингу коду 3.13 представлено скрипт, який

відправляє запит на реєстрацію користувача надсилаючи відповідні дані на сервер.

Лістинг 3.13 – Запит на реєстрація користувача

```
$.ajax({
    url: 'https://localhost:7054/api/workshop/get/list',
    method: 'get',
    success: (response) => {
        let select = document.getElementById('workshop-type')
        response.items.forEach(item => {
            select.innerHTML += "<option value='" + item.id + "'>" + item.name + "</option>"
        })
    },
    error: (response) => {
        console.log(response)
    }
})
.done(() => {
    $('#preloader').fadeOut()
})

$('#next-page-button').on('click', () => {
    $('#first-step-page').addClass('d-none')
    $('#first-step-control').addClass('d-none')
    $('#second-step-page').removeClass('d-none')
    $('#second-step-control').removeClass('d-none')
})

$('#prev-page-button').on('click', () => {
    $('#first-step-page').removeClass('d-none')
    $('#first-step-control').removeClass('d-none')
    $('#second-step-page').addClass('d-none')
    $('#second-step-control').addClass('d-none')
})

$('#login-button').on('click', () => {
    let data = {
        firstName: $('#first-name').val(),
        lastName: $('#last-name').val(),
        middleName: $('#middle-name').val(),
        workshopId: $('#workshop-type').val(),
        login: $('#login').val(),
        password: $('#password').val()
    }

    let jsonData = JSON.stringify(data)

    $.ajax({
        url: 'https://localhost:7054/api/authentication/sign-up',
        method: 'post',
        data: jsonData,
        dataType: 'json',
        contentType: 'application/json; charset=utf-8',
        success: (response) => {
            console.log(response)
            localStorage.setItem('token', response.value)

            let token = localStorage.getItem('token')
            if (token == undefined || token == null) {
```

```
            console.log("Error")
        }

        console.log(token)
```

Продовження лістингу коду 3.13.

```
        let item = document.getElementById('user-name')

        $.ajax({
            url: 'https://localhost:7054/api/authentication/current-user',
            method: 'get',
            headers: {
                "Authentication": "Bearer " + token
            },
            success: (response) => {
                console.log(response)
                localStorage.setItem('username', response.firstName + " " + response.lastName)
                location.href = "@Url.Action("Index", "Home")"
            },
            error: (response) => {
                console.log("Error")
            }
        })
    },
    error: (response) => {
        console.log(response)
        return;
    }
})
        });
```

Даний код збирає дані з форми для реєстрації в JSON об’єкт та відправляє на сервер, на кінцеву точку для виконання реєстрації, а згодом отримує відповідь. Ця відповідь містить в собі авторизаційний токен, який видається при виконанні авторизації або реєстрації в системі та дозволяє використовувати інші кінцеві точки для отримання або зміни деякої інформації. Настпуний запит, який виконується це запит на отримання користувацьких даних з цього ж токену. Після отримання користувацьких даних, його прізвище та ім’я зберігаються в локальному сховищі дял подального використання.

Макет _HeaderLayout використовується для сторінок, які призначені для внутрішнього використання в інформаційній системі [40]. Шаблони також можуть містити певні скрипти, і дана можливість також використовується в даній реалізації. Макет _HeaderLayout містить головний скрипт, який представлено на лістингу 3.14, для перевірки авторизації користувача. Він виконується щоразу при будь-якому спрацюванні рендрингу сторінки.

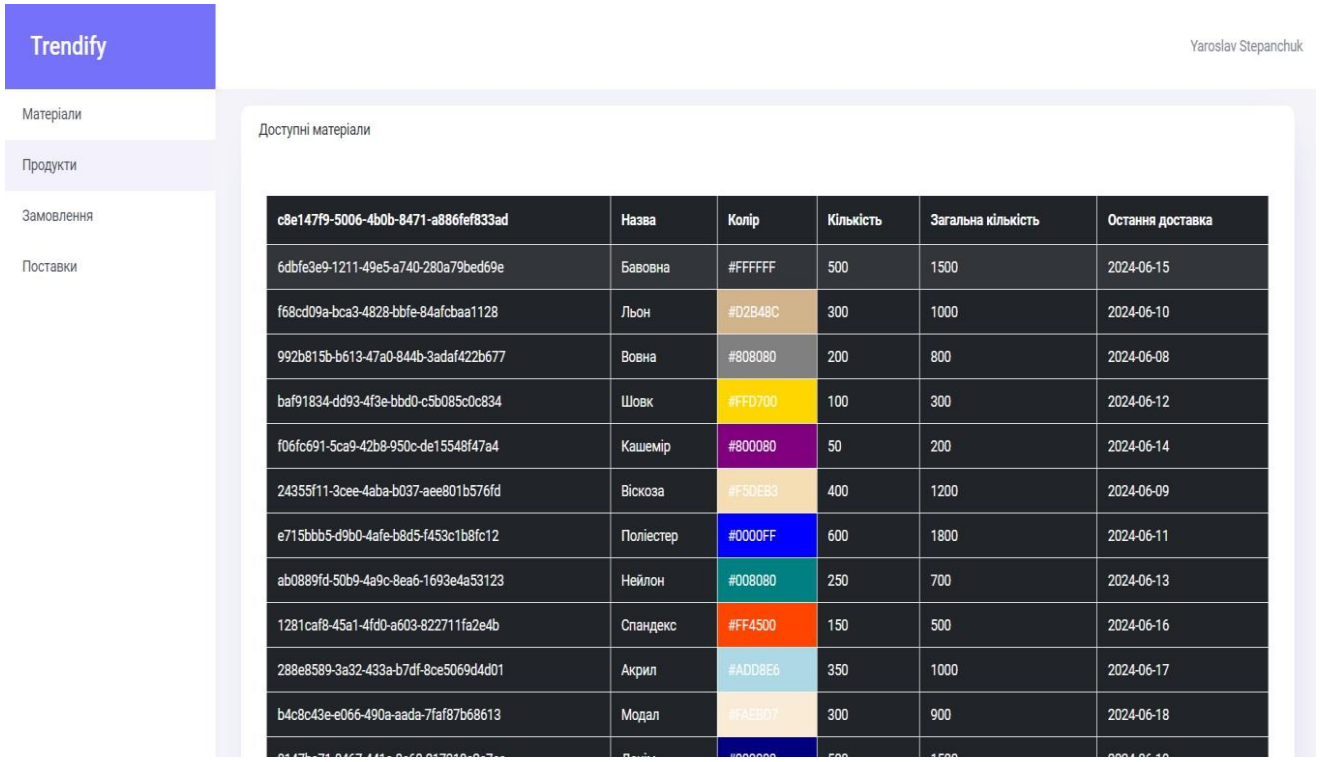
Лістинг коду 3.14 – Перевірки авторизованого користувача.

```
let username = localStorage.getItem('username');
if (username == undefined || username == null) {
    location.href = "@Url.Action("SignIn", "Authentication")"
}

let token = localStorage.getItem('token')
if (token == undefined || token == null) {
    location.href = "@Url.Action("SignIn", "Authentication")"
}

$.ajax({
    url: 'https://localhost:7054/api/authentication/current-user',
    method: 'get',
    headers: {
        "Authentication" : "Bearer " + token
    },
    success: (response) => {
        document.getElementById('user-name').innerHTML = username
        $('#preloader').fadeOut();
    },
    error: (response) => {
        location.href = "@Url.Action("SignIn", "Authentication")"
    }
})
```

Головними даними, які вважаються необхідними для роботи системи являються ім’я користувача, оскільки воно завжди виводить на верхій панелі програми та токен, який був виданий в процесі авторизації. Токен завжди передається на сервер при будь якому запиті і вважається ключем доступу, оскільки він виданий конкретному користувачу в конкретний момент часу на містиить персональну ідентифікаційну інформацію про користувача. Вважається, що підібрати токен неможливо, оскільки він представлений у форматі base64 та підписаний цифровим підписом конкретного серверу з використанням секретних ключів.

При реалізації клієнтської частини не буде розглядатись спосіб отримання інформації від серверу MVC або API окрім тих, що вже були розглянути, а буде зосереджена увага на реалізації користувацького інтерфейсу та виведенні отриманих наборів даних, які також будуть розглянуті. Першою сторінкою, на якій буде зосереджена увага, це сторінка вироблення товарів. Дана сторінка має деякий набір даних, який включає в себе доступні матеріали та продукти із міткою про можливість їх створення, а також уже наявних продуктів на цьому

цеху. Таблиця із наявними матеріалами зображена на рисунку 3.8, а таблиця із товарами для виготовленян на рисунку 3.9 відповідно.



c8e147f9-5006-4b0b-8471-a886fef833ad	Назва	Колір	Кількість	Загальна кількість	Остання доставка
6dbfe3e9-1211-49e5-a740-280a79bed69e	Бавовна	#FFFFFF	500	1500	2024-06-15
f68cd09a-bca3-4828-bbfe-84afcbaa1128	Льон	#D2B48C	300	1000	2024-06-10
992b815b-b613-47a0-844b-3adaf422b677	Вовна	#808080	200	800	2024-06-08
baf91834-dd93-4f3e-bbd0-c5b085c0c834	Шовк	#FFD700	100	300	2024-06-12
f06fc691-5ca9-42b8-950c-de15548f47a4	Кашемір	#800080	50	200	2024-06-14
24355f11-3cee-4aba-b037-aee801b576fd	Віскоза	#F5DEB3	400	1200	2024-06-09
e715bbb5-d9b0-4afe-b8d5-f453c1b8fc12	Поліестер	#0000FF	600	1800	2024-06-11
ab0889fd-50b9-4a9c-8ea6-1693e4a53123	Нейлон	#008080	250	700	2024-06-13
1281caf8-45a1-4fd0-a603-822711fa2e4b	Спандекс	#FF4500	150	500	2024-06-16
288e8589-3a32-433a-b7df-8ce5069d4d01	Акрил	#ADD8E6	350	1000	2024-06-17
b4c8c43e-e066-490a-aada-7faf87b68613	Модал	#FAEBD7	300	900	2024-06-18

Рисунок 3.8 – Таблиця з товарами

Як зазначалось раніше, усі товари отримуються вибіркою із доставок відповідного цеху. При завантажені сторінки MVC сервер робить запит на відповідний API сервер та отримує дані про матеріали та їх кількість. Після отримання необхідних даних формується модель, як містить дані для опису відповідної сторінки. MVC сервер виводить це все на клієнтську частину у вигляді готової HTML розмітки та використанння Razor Page синтаксису. Разом із цим, також отримуються дані про доступні товари, зокрема, аналізується необхідна кількість матеріалів на даному цеху. Усі дані збираються в одну модель та виводяться на сторінку.

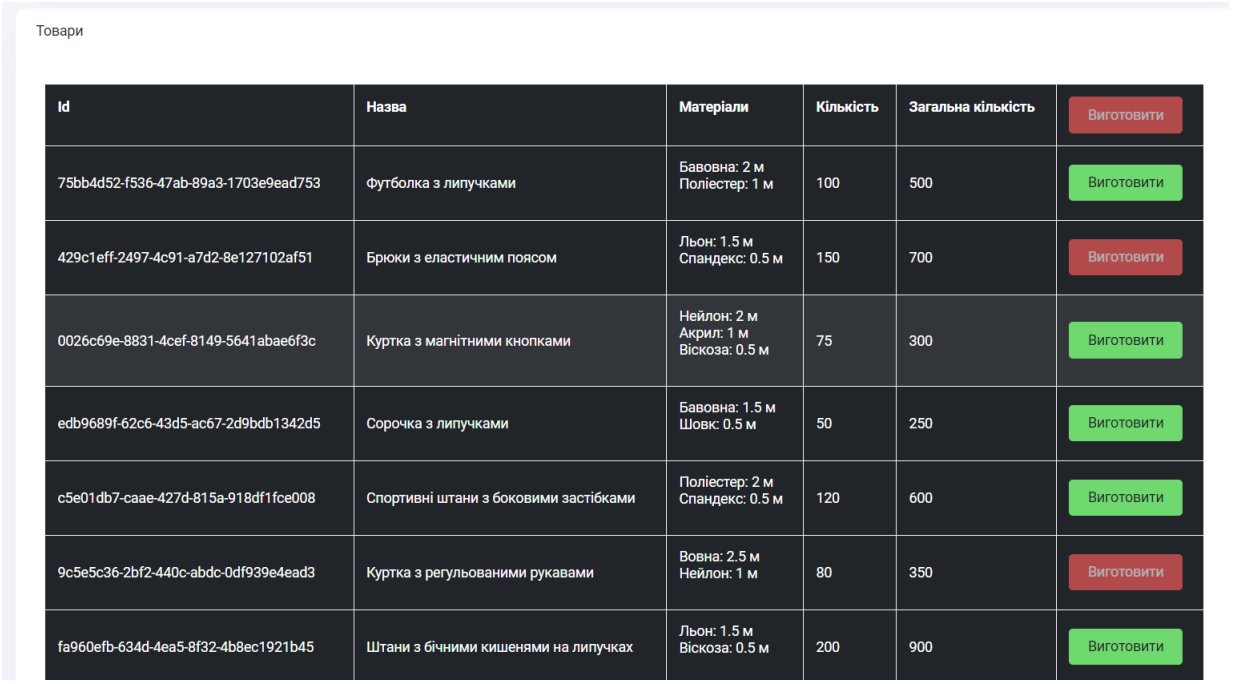


Рисунок 3.9 – Панель виготовлення товарів

Даний функціонал доступний для користувачів виробничого цеху, оскільки тільки вони мають доступ до відповідного функціоналу. Натомість користувачі інших цехів мають інший користувацький інтерфейс та, відповідно, фунціонал. Для прикладу, підготовчий цех має справу з вже готовим одягом, а до його обов'язків відноситься перевірка якості виробленої продукції. На рисунку 3.10 зображено відповідну функціональну панель.

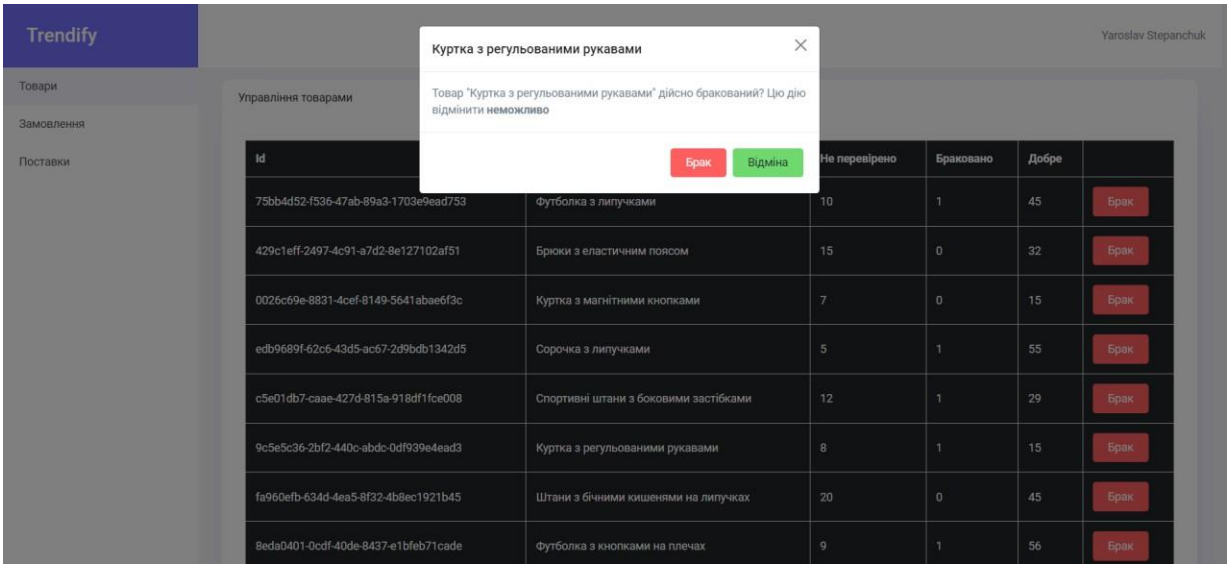


Рисунок 3.10 – Підготовка товару

Будь які внутрішні об’єкти попадають в цех через системи замовлень. У кожного цеху є відповідний функціонал, де робітник може замовити деякий набір необхідних матеріалів або товарів які йому необхідні. Цехи такі як виробничий або підготовчий можуть запросити лише матеріали, оскільки вони являються цільовим предметом для роботи цього цеху, натомість швейний або закрійний цехи можуть запросити лише матеріали. На рисунку 3.11 зображено таблицю виробничого цеху із доставками запрошених матеріалів.

Запити

Id	Матеріали (Кількість)	Дата запиту	Статус
1	Бавовна (50) Поліестер (60)	2024-06-28	Запрошено
2	Вовна (20) Віскоза (40)	2024-06-29	Відправлено
3	Льон (30) Нейлон (20) Шовк (10)	2024-06-30	Відправлено
4	Спандекс (15) Денім (50) Трикотаж (45)	2024-07-01	Відхилено
5	Акрил (35) Жаккард (10) Органза (25)	2024-07-02	Доставлено (2024-07-03)
6	Модал (30) Кашемір (5)	2024-07-03	Доставлено (2024-07-04)
7	Фліс (20) Оксамит (5)	2024-07-04	Відхилено

Рисунок 3.11 – Таблиця запитів матеріалів

Розробка користувацького інтерфейсу є досить важливою в функціональному плані, оскільки через нього кінцевий користувач може взаємодіяти із сервером та системою в цілому. Якісна подача інформації сприяє швидкому аналізу та зменшує вірогідність помилок зумовлених людським фактором. UX відноситься до проєктування та розробки користувацького інтерфейсу який базується на досвіді користувача в використанні подібного програмного забезпечення. Основна ціль при проєктуванні та розробці користувацького інтерфейсу в ході виконання кваліфікаційної роботи має

виключно функціональних характер для забезпечення мінімального інтерфейсу для взаємодії із системою та простотою прийняття відповідних рішень.

3.4 Висновок

У третьому розділі було виконано реалізацію зв'язку з базою даних, серверної частини та клієнтського додатку для інформаційної системи підприємства з виготовлення адаптивного одягу.

Використовуючи Entity Framework Core, було забезпечено ефективний та зручний доступ до бази даних. Було реалізовано механізм Generic Repository, який дозволяє спростити та уніфікувати операції з даними. Це забезпечило зменшення дублювання коду, підвищення його якості та зручність подальшої підтримки і розширення системи.

Серверна частина була реалізована на основі ASP.NET Core Web API. Для внутрішньої маршрутизації запитів було використано MediatR, що дозволяє організувати чітку та зрозумілу архітектуру обробки запитів. Це сприяє покращенню підтримуваності та масштабованості коду. Ін'єкція Generic Repository в MediatR забезпечила зручну роботу з даними, що підвищило ефективність розробки та продуктивність системи.

Клієнтська частина була реалізована за допомогою ASP.NET Core MVC, що дозволило створити зручний та інтерактивний інтерфейс для користувачів. Для зв'язку з серверною частиною було використано AJAX, що забезпечило асинхронну взаємодію та швидкий обмін даними без необхідності перезавантаження сторінок. Це значно покращило користувацький досвід та підвищило зручність роботи з системою.

Загалом, у третьому розділі було виконано комплексну реалізацію основних компонентів інформаційної системи, включаючи зв'язок з базою даних, серверну та клієнтську частини.

ВИСНОВКИ

Результатом виконаної кваліфікаційної роботи стало створення ефективної інформаційної системи для підприємства з виготовлення адаптивного одягу, яка дозволяє автоматизувати та оптимізувати основні виробничі та логістичні процеси. Система інтегрує всі підрозділи підприємства, включаючи швейний, виробничий та підготовчий цехи, а також складські приміщення, що забезпечує прозорість і контроль на всіх етапах виробництва.

У першому розділі досліджено предметну область та виконано постановку задач.

У другому розділі виконано проєктування інструментальних засобів створення і використання ІС.

У третьому розділі виконано реалізацію розробленої ІС.

Мета роботи по забезпеченню обміну даними між різними підрозділами підприємства в межах однієї інформаційної системи досягнута.

Практичне значення полягає в організації взаємодії між різними підрозділами підприємства в межах однієї інформаційної системи, а також контроль та управління виробничими процесами.

Таким чином, впровадження розробленої інформаційної системи дозволило не тільки вирішити основні задачі підприємства, але й закласти основу для подальшого розвитку та вдосконалення виробничих процесів, що сприяє підвищенню якості продукції та задоволенню потреб споживачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Adaptive apparel for people with disabilities: A systematic literature review and future research agenda. URL: https://www.researchgate.net/publication/380897329_Adaptive_apparel_for_people_with_disabilities_A_systematic_literature_review_and_future_research_agenda (дата звернення 15.02.2024).
2. Chapter 1. The Adaptive Apparel Designer’s Guide to Research. URL: https://iastate.pressbooks.pub/adaptiveapparel/chapter/adaptive-apparel-designers-guide-to-research/ (дата звернення 15.02.2024).
3. Blockchain on MSP430 with IEEE 802.15.4 URL: https://ieeexplore.ieee.org/abstract/document/9314805 (дата звернення 02.03.2024).
4. EEPROM endurance degradation at different temperatures: State of the art TCAD simulation URL: https://www.sciencedirect.com/science/article/abs/pii/S0026271422002414 (дата звернення 02.03.2024).
5. Challenges and Trends of SRAM-Based Computing-In-Memory for AI Edge Devices URL: https://ieeexplore.ieee.org/abstract/document/9382915 (дата звернення 12.03.2024).
6. Design and Implementation of High-Speed Universal Asynchronous Receiver and Transmitter (UART) URL: https://ieeexplore.ieee.org/abstract/document/9070856 (дата звернення 15.03.2024).
7. A Flexible Hardware Architecture for Slave Device of I2C Bus URL: https://ieeexplore.ieee.org/abstract/document/8991113 (дата звернення 02.01.2024).
8. Interfacing of light sensor with FPGA using I2C bus URL: https://ieeexplore.ieee.org/abstract/document/9074372 (дата звернення 02.03.2023).
9. A review of multiple input DC-DC converter topologies linked with hybrid electric vehicles and renewable energy systems, URL:

https://www.sciencedirect.com/science/article/abs/pii/S1364032120304767 (дата звернення 14.03.2023).

10. GPIO Based Super-Twisting Sliding Mode Control for PMSM URL: https://ieeexplore.ieee.org/abstract/document/9139259 (дата звернення 14.03.2023).

11. Advanced DSP for coherent optical fiber communication URL: https://www.mdpi.com/2076-3417/9/19/4192 (дата звернення 14.03.2023).

12. Master-slave synchronization via dynamic control URL: https://www.sciencedirect.com/science/article/abs/pii/S1007570419302965 (дата звернення 14.03.2023).

13. The Real-Time Clock and the DS3231 URL: https://link.springer.com/chapter/10.1007/978-1-4842-9051-4_10 (дата звернення 14.03.2023).

14. The Icarus Mote: Employing Off-Chip RTC to Attain 22 nA Sleep Current in Duty-Cycled IoT Devices URL: https://dl.acm.org/doi/abs/10.1145/3423423.3423469 (дата звернення 14.03.2023).

15. Quantum Dot Light-Emitting Diodes URL: https://pubs.acs.org/doi/abs/10.1021/acs.chemrev.2c00695 (дата звернення 14.03.2023).

16. Multiphase PWM characteristics in the EER transmitter envelope path URL: https://ieeexplore.ieee.org/abstract/document/9619166 (дата звернення 20.03.2023).

17. Zigbee Control Design of Pixelated Light Strips URL: https://pure.tue.nl/ws/portalfiles/portal/139037233/h (дата звернення 20.03.2023).

18. Створення низькочастотного навчального осцилографа на базі Arduino UNO R3 та персонального комп'ютера URL: http://eir.zp.edu.ua/handle/123456789/7762 (дата звернення 20.03.2023).

19. Confocal Microscopy: Principles and Modern Practices URL: https://currentprotocols.onlinelibrary.wiley.com/doi/abs/10.1002/cpcy.68 (дата звернення 20.03.2023).

20. Implementation of arduino pro mini and ESP32 cam for temperature monitoring on automatic thermogun IoT-based URL: https://pdfs.semanticscholar.org/c66c/510162661c8b3e93399ada5274134560c782.pdf (дата звернення 20.03.2023).

21. Motion Detector Using NE555 Timer URL: https://mechanical.journalspub.info/index.php?journal=JIEGT&page=article&op=view&path%5B%5D=1354 (дата звернення 20.03.2023).

22. Projektiranje podesivog izvora napajanja s LM317 integriranim krugom URL: https://zir.nsk.hr/islandora/object/vuka%3A1521/datastream/PDF/view (дата звернення 20.03.2023).

23. Система групового керування світлодіодами для дослідження методів змішування кольорів URL: http://umj.metrology.kharkov.ua/index.php/2617-5509/article/view/200676 (дата звернення 20.03.2023).

24. Автоматизована система для контролю багатошарових конструкцій методом вільних коливань URL: https://ela.kpi.ua/bitstream/123456789/38198/1/Tasazh_magistr.pdf (дата звернення 20.03.2023).

25. Mechanical single-molecule potentiometers with large switching factors from ortho-pentaphenylene foldamers URL: https://www.nature.com/articles/s41467-020-20311-z (дата звернення 20.03.2023).

26. Real Time Clock (RTC) URL: https://www.renesas.com/us/en/products/gadget-renesas/reference/gr-peach/library-rtc (дата звернення 25.03.2023).

27. Інфрачервоні світлодіоди - потужні, працездатні, довговічні URL: http://stroyka-gid.com.ua/kerivniztv/14785-infrachervoni-svitlodiody.html (дата звернення 25.03.2023).

28. Історія розвитку розумних годинників: 4 етапи і думки про майбутнє URL:https://www.moyo.ua/ua/news/istoriya_razvitiya_umnyh_chasov_4_etapa_i_mysli_o_budushchem.html (дата звернення 25.03.2023).

29. Інноваційний смарт-годинник для активного способу життя URL: https://dailylviv.com/news/osvita-i-nauka/innovatsiinyi-smart-hodynnyk-dlya-aktyvnoho-sposobu-zhyttya-111305 (дата звернення 25.03.2023).

30. Денніс Габор URL: https://www.wikiwand.com/uk/%D0%94%D0%B5%D0%BD%D0%BD%D1%96%D1%81_%D0%93%D0%B0%D0%B1%D0%BE%D1%80 (дата звернення 25.03.2023).

31. Управління сталим розвитком URL: https://enpuir.npu.edu.ua/bitstream/handle/123456789/33254/Sustainable%20development%20management.pdf?sequence=1 (дата звернення 25.03.2023).

32. GARMIN FENIX 5S URL: https://www.navionika.com/shop/garmin-fenix-5s.html (дата звернення 25.03.2023).

33. Розробка програмного забезпечення для визначення показників екосистеми в приміщенні URL: https://er.knutd.edu.ua/bitstream/123456789/23152/3/Dyplom122_K%D0%BEchuk_Astistova.pdf (дата звернення 25.03.2023).

34. Аналого-цифровий перетворювач URL: https://vue.gov.ua (дата звернення 25.03.2023).

35. Схема ЦАП. Цифро-аналогові перетворювачі: типи, класифікація, принцип роботи, призначення URL: https://hi-news.pp.ua/tehnka-tehnologyi/15906-shema-cap-cifro-analogov-peretvoryuvach-tipi-klasifkacya-princip-roboti-priznachennya.html (дата звернення 25.03.2023).

36. Мікропроцесори та мікроконтролери URL: https://ela.kpi.ua/bitstream/123456789/40858/1/Mikroprotsesory-ta- mikrokontrolery (дата звернення 05.04.2023).

37. Вивчення властивостей мікроконтролерів і електронних систем на базі платформи Ардуіно URL: https://openarchive.nure.ua/server/api/core/bitstreams/7349b613-9f68-4edf-9f1d-c35baac25c76/content (дата звернення 05.04.2023).

38. Програмування мікроконтролерів AVR URL: http://pdf.lib.vntu.edu.ua/books/IRVC/2021/Tsirulnik_2018_111.pdf (дата звернення 05.04.2023).

39. Процесори цифрової обробки сигналів URL: http://ni.biz.ua/15/15_6/15_65379_protsessori-tsifrovoy-obrabotki-signalov.html (дата звернення 14.04.2023).

40. Архітектура мікроконтролерів aTmega8 URL: https://studfile.net/preview/9076005/page:3/ (дата звернення 14.04.2023).

41. Гарвардська архітектура: походження, модель, як це працює URL: https://uk.warbletoncouncil.org/arquitectura-harvard-8234 (дата звернення 14.04.2023).

42. Архітектура фон Неймана URL: https://vue.gov.ua (дата звернення 14.04.2023).

43. Як створити дзеркальний том в Windows 10 URL: https://uk.begin-it.com/6006-create-mirrored-volume-instant-hard-drive-backup-windows-10 (дата звернення 22.04.2023).

44. Парктронік стрічковий URL: https://uk.warbletoncouncil.org/arquitectura-harvard-8234 (дата звернення 22.04.2023).

45. Kingbright Electronic Co., LTD URL: https://www.sea.com.ua/ua/producer/kingbright-electronic-co-ltd/ (дата звернення 04.05.2023).

46. Урок 12 Електродвигуни. Електровимірювальні прилади. Гучномовець URL: https://naurok.com.ua/urok-12-elektrodviguni-elektrovimiryuvalni-priladi-guchnomovec-313713.html (дата звернення 04.05.2023).

47. Акрилова тканина URL: https://next-buy.com.ua/ua/blog/264-akrilova-tkanina (дата звернення 04.05.2023).

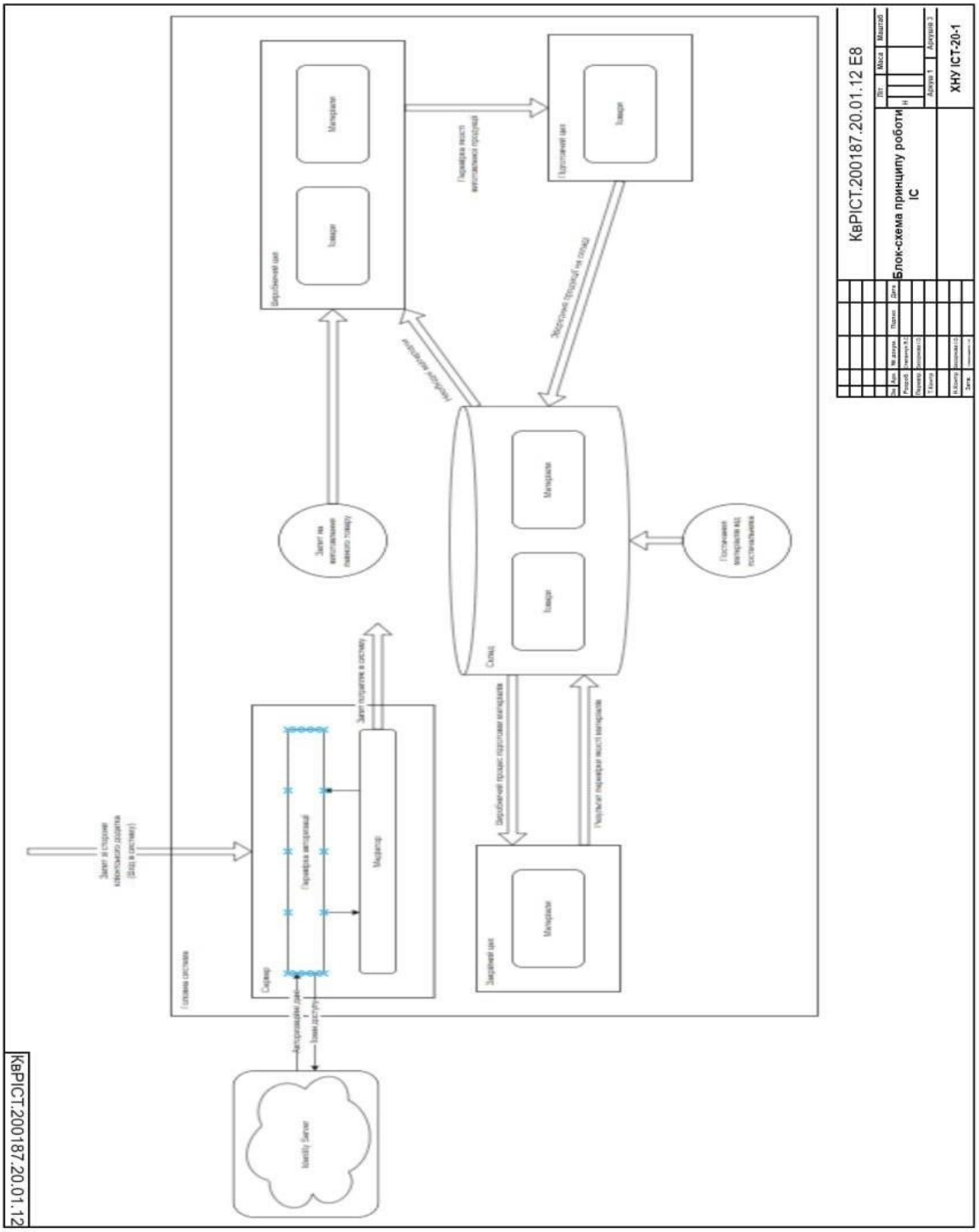
48. Світлодіодна матриця URL: https://vela.com.ua/ (дата звернення 04.05.2023).

49. Що означає RGB в освітленні? URL: https://www.svit-lamp.ua/stati/co-znamena-rgb-u-svitidel/ (дата звернення 04.05.202

Додаток А

(обов'язковий)

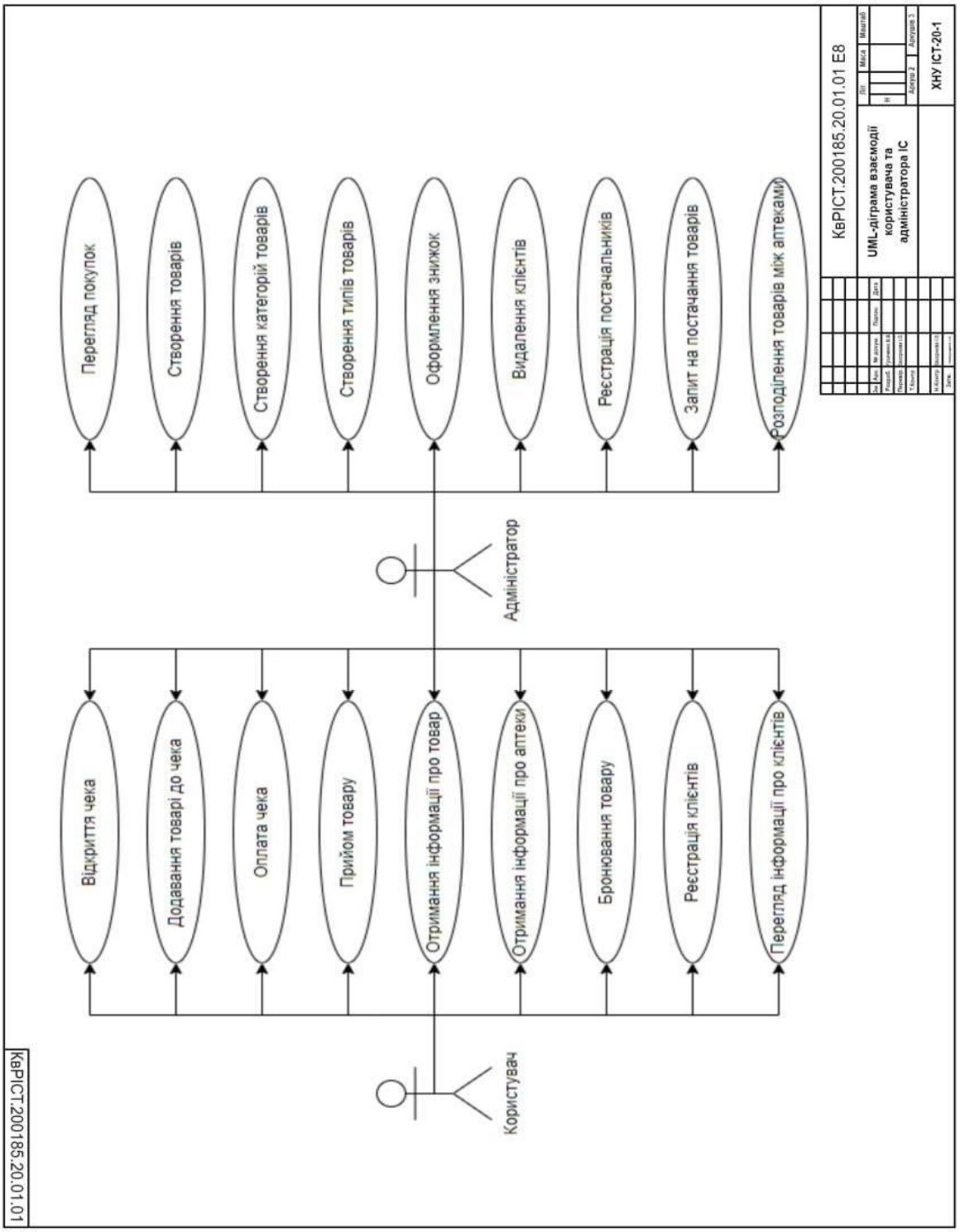
Копія креслення «Блок-схема принципу роботи ІС»



Додаток Б

(обов'язковий)

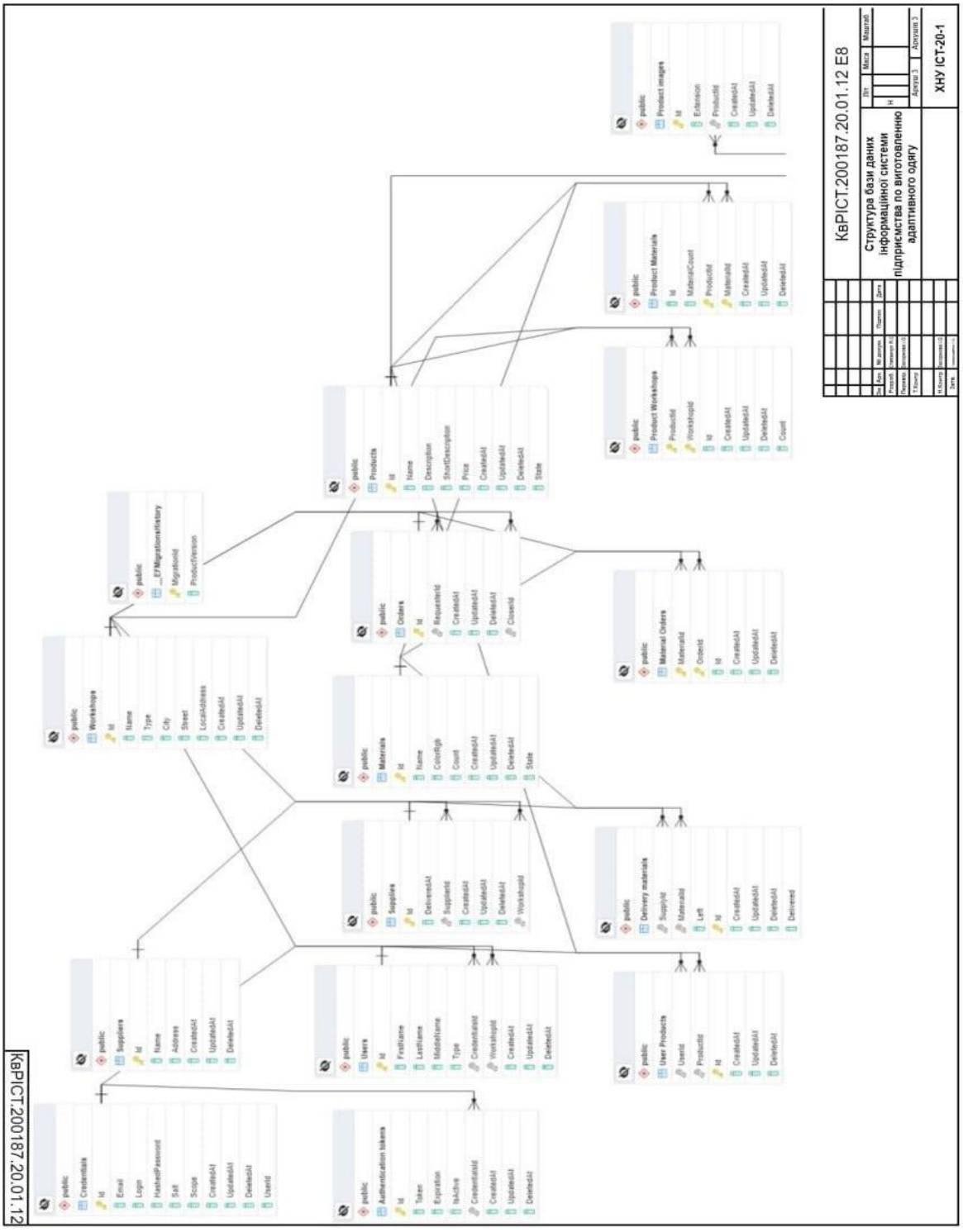
Копія креслення «UML-діаграма взаємодії користувача та адміністратора ІС»



Додаток В

(обов'язковий)

Копія креслення «Структура бази даних ІС підприємтсва по виготовленню адаптивного одягу»



UNICHECK
by Turnitin

Ім'я користувача:
Кафедра КІ

ID перевірки:
1016383160

Дата перевірки:
23.06.2024 09:29:29 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
23.06.2024 09:30:03 EEST

ID користувача:
100005591

Назва документа: **Степанчук_Інформаційна система підприємства по виготовленню адаптивного одягу**

Кількість сторінок: **79** Кількість слів: **13762** Кількість символів: **110703** Розмір файлу: **1.61 MB** ID файлу: **1016193418**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

11.2%
Схожість

Найбільша схожість: **8.57%** з джерелом з Бібліотеки (ID файлу: **1015058379**)

10.3% Джерела з Інтернету	758	Сторінка 81
10.1% Джерела з Бібліотеки	220	Сторінка 82

1.56% Цитат

Цитати	13	Сторінка 83

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	10
Підозріле форматування	14 сторінок

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 8.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 12%**

	Документ		Сумарний збіг по Базі Даних	
ID: 132225 Назва: БКР Інформаційна система підприємства по виготовленню адаптивного одягу Додано в БД: 2024-06-23 Автора: Я. С. Степанчук Керівники: І. О. Засорнова Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	94634	769	8151 (9%)	56 (7%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Степанчук Ярослав Сергійович

Тема: Інформаційна система підприємства по виготовленню адаптивного одягу

Спеціальність: 126 «Інформаційні системи та технології»

Обсяг кваліфікаційної роботи:

Кількість листів креслень ___3___ Кількість сторінок записки ___66___

1. Короткий зміст роботи та прийнятих рішень: Метою роботи є забезпечення обміну даними між різними підрозділами підприємства в межах однієї інформаційної системи

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі досліджено предметну область та виконано постановку задач. У другому розділі виконано проєктування інструментальних засобів створення і використання ІС. У третьому розділі виконано реалізацію розробленої ІС

4. Позитивні сторони роботи: висока практична цінність роботи.

5. Негативні сторони роботи: система не повністю закрита, не було проведено тестування з розгортання.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на належному науково-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: 3,5 (D) задовільно

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) _____

Бедрашюк Леонід Петрович, зав. кафедри ІПЗ, ХНУ

"24" грудня 2024 р. _____ (підпис)

Завідувачу кафедри КІІС
д-р.техн.наук, проф. Говорущенко Т. О.

Степанчук Ярослав Сергійович
ПІБ здобувача вищої освіти

ФІТ, 4 курсу, групи ІСТ-20-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

30 травня 2024 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ
КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Інформаційна система підприємства по виготовленню адаптивного одягу

Автор: Степанчук Ярослав Сергійович

Спеціальність: 126-Інформаційні системи та технології

Освітня програма: освітньо-професійна

Науковий керівник: Засорнова Ірина Олександрівна, к.т.н, доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1Вони розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи.

2)Усі запозичення фрагментарні або мають належним чином оформлені посилання.

3)Окремі збіги є загальновживаними фразами або виразами, що збігаються з багатьма джерелами одночасно.

4)Зафіксовані системою ознаки модифікації тексту стосуються комбінування латинських символів з україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 11.2% і адресується до 978 першоджерела, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ І.О. Засорнова

Гарант ОП _____ Є.Г. Гнатчук

Завідувач кафедри КІІС _____ Т. О. Говорущенко