

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ДЛЯ
УПРАВЛІННЯ МЕРЕЖЕЮ МАГАЗИНІВ ОДЯГУ
RESEARCH AND SOFTWARE IMPLEMENTATION OF A SYSTEM FOR
MANAGING A CHAIN OF A CLOTHING STORES

Виконав: студент(ка) 2 курсу, групи КНТ-122м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ШЕВЕРДІНКІН І.А.

(ПРИЗВИЩЕ та ініціали)

Керівник КАПЛІЄНКО Т.І.

(ПРИЗВИЩЕ та ініціали)

Рецензент ШИТІКОВА О.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ

Кафедра програмних засобів

Ступінь вищої освіти магістр

Спеціальність 121 Інженерія програмного забезпечення

(код і найменування)

Освітня програма (спеціалізація) Інженерія програмного забезпечення

(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.

Сергій СУББОТІН

“ ” 2023 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ШЕВЕРДІНКИНА Іллі Андрійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проекту (роботи) Дослідження та програмна реалізація системи для управління мережею магазинів одягу. Research And Software Implementation Of A System For Managing A Chain Of A Clothing Stores

керівник проекту (роботи) к.т.н., доцент, КАПЛІЄНКО Тетяна Ігорівна

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “14” листопада 2023 року №437

2. Строк подання студентом проекту (роботи) 01 грудня 2023 року

3. Вихідні дані до проекту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Опис програми. 3. Розробка програми. 4. Керівництво програміста. 5. Керівництво оператора. Додаток А. Текст програми. Додаток Б. Слайди презентації.

5. Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	КАПЛІЄНКО Т. І., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання « 08 » вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	3 тиждень	Розділ 2
4	Розробка архітектури програми.	4-5 тиждень	Розділ 3,4
5	Розробка програми.	6-8 тижні	Розділ 3,4
6	Оформлення пояснювальної записки та документів до неї.	9-10 тиждень	Додатки
7	Нормоконтроль та рецензування.	11 тиждень	
8	Захист роботи.	12 тиждень	

Студент(ка)

_____ Ілля ШЕВЕРДІНКІН
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Тетяна КАПЛІЄНКО
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
157 с., 9 табл., 90 рис., 2 дод., 23 джерел.

ВЕБСАЙТ, JAVASCRIPT, NESTJS, REACT, REDUX, MATERIAL UI,
CRM, ОДЯГ.

Об'єкт дослідження – управління взаємовідносин з клієнтами з використанням спеціалізованих систем (Customer Relationship Management або CRM).

Предмет дослідження – розробка вебсайтів CRM.

Мета роботи – створення вебсайту CRM для мережі магазинів одягу.

Матеріали, методи та технічні засоби: функціональне програмування, фреймворк React, мови програмування JS, платформа NestJs.

Результати. Створено вебсайт CRM, що дозволяє переглядати та керувати бізнесом мережі магазинів одягу, а саме робітниками, постачальниками, магазинами, товарами, складом, акціями тощо.

Висновки. Розроблено вебсайт CRM, що дозволяє керувати мережею магазинів одягу.

Галузь використання – управління мережею магазинів одягу використовуючи вебсайт.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 157 pages, 9 tables, 90 figures, 2 appendixes, 23 sources.

WEBSITE, JAVASCRIPT, NESTJS, REACT, REDUX, MATERIAL UI, CRM, CLOTHING.

Object of study is customer relationship management using Customer Relationship Management (CRM) systems.

Subject of study is development of websites for CRM systems.

The purpose of the work is to create a website for a CRM system for a chain of clothing stores.

Materials, methods and tools: functional programming, React framework, JS programming languages, NestJs platform.

Results. A website CRM system has been created that allows you to view and manage the business of a clothing store chain, namely employees, suppliers, stores, goods, warehouse, promotions, etc.

Conclusions. A website CRM system has been developed that allows you to manage a network of clothing stores.

The field of use is management of a network of clothing stores using a website.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ	9
1 Аналіз предметної області	10
1.1 Огляд предметної області	10
1.2 Опис структури та складників вебсайтів	10
1.3 Наявні рішення.....	11
1.3.1 ClickUp.....	11
1.3.2 Zendesk.....	12
1.3.3 Microsoft Dynamics	12
1.4 Висновки.....	15
2 Опис програми	16
2.1 Вибір та обґрунтування мови програмування	16
2.1.1 Порівняння мов програмування.....	16
2.1.2 Порівняння фреймворків/бібліотек/платформ	17
2.2.1 Бібліотека React	19
2.2.2 Платформа Nest.js.....	20
2.3 Обґрунтування вибору середовища розробки	21
2.4 Опис складників програми	22
2.5 Використані бібліотеки.....	22
2.6 Обґрунтування вибору системи керування базами даних	27
2.7 Висновки розділу	29
3 Розробка програми	30
3.1 Схема БД.....	30
3.2 Розробка інтерфейсу.....	33
3.3 Структура проєкту.....	36
3.4 Використані алгоритми.....	40
3.5 Використання бібліотек та фреймворків.....	45

3.5.1 Бібліотека React	45
3.5.2 Платформа Nest.js	48
3.5.3 Бібліотека class-validator	51
3.5.4 Бібліотека Joi	53
3.5.4 Бібліотека Mongoose	54
3.5.5 Бібліотека Redux	56
3.5.6 Бібліотека MUI	60
3.6 Використання сервісів контролю версій	65
4 Керівництво програміста	69
4.1 Опис застосування	69
4.2 Інструкція по експлуатації програми	69
4.3 Взаємодія з БД	70
4.4 Хостинг	72
4.5 Помилки	73
5 Керівництво оператора	77
5.1 Призначення та умови виконання програми	77
5.2 Виконання програми	77
5.3 Повідомлення оператору	78
5.4 Користувачі	80
Висновки	86
Перелік джерел посилань	87
Додаток А Текст програми	89
Додаток Б Слайди презентації	148

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
Cookie	– інформація, отримана від вебсайту, яка зберігається у клієнта, тобто браузера, а потім відправлена на той самий сайт, якщо його буде повторно відвідано;
CRM	– Customer Relationship Management;
DTO	– data transfer object. Об'єкт який містить у собі дані, структура яких визначена заздалегідь;
IDE	– integrated development environment;
Npm	– node package manager. Менеджер пакетів, за допомогою якого інсталюються бібліотеки у проєкт, також через нього запускають скрипти;
RESTfull	– це архітектурний стиль API, який використовує HTTP-запити для доступу та використання даних;
БД	– база даних;
Валідація	– перевірка значень на відповідність зазначеним правилам;
Декоратор	– інструмент за допомогою якого до класу додають метаданні;
Ендпоінт	– шлях, за яким можна звернутися до back-end серверу для отримання або передачі інформації;
Мікросервіс	– архітектурний стиль, за яким єдиний застосунок будується як сукупність малих сервісів;
ООП	– об'єктно орієнтоване програмування;
ПК	– персональний комп'ютер;
Редьюсер	– файл, у якому зберігається стан застосунку та методи для його зміни;
СКБД	– система керування базами даних.

ВСТУП

З прогресивним розвитком сучасного світу з'являється все більше і більше можливостей для вдосконалення діяльності компанії. Інформаційні ресурси є важливою складовою інформаційно-виробничих процесів, якій приділяється багато уваги в публікаціях останніх років, але відсутність методології дослідження та оцінки ефективності їх використання потребує подальших розробок у цьому напрямку. Існує велика кількість різноманітних систем, моделей, програм, технологій, що викликає труднощі у виборі варіанту для ефективної роботи підприємства. Кожне підприємство прагне виробляти якісну продукцію, яка буде користуватися попитом у споживачів. Практика показує, що кінцевий результат роботи залежить як від організації виробничого процесу, так і від організації збуту продукції, а саме від співпраці відділу збуту та маркетингу з потенційними клієнтами, покупцями та споживачами. Саме співпраця між продавцем і покупцем визначає майбутній обсяг замовлень на продукцію, а отже, безпосередньо впливає на замовлення продукції, і відповідно безпосередньо впливає на прибутковість підприємства.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд предметної області

Управління взаємовідносинами з клієнтами - це концепція, яка охоплює концепції, що використовуються компаніями для управління своїми відносинами з клієнтами, включаючи збір, зберігання та аналіз інформації про клієнтів, постачальників, партнерів та інформацію про взаємовідносини з ними. Сучасна концепція CRM спрямована на вивчення ринку та специфічних потреб клієнтів. У цій роботі розглядається CRM система, основною задачею та метою якого є спростити управління та структурування мережею магазинів одягу.

1.2 Опис структури та складників вебсайтів

Зазвичай вебсайт складається з 3 частин: верхньої частини, яка англійською називається "хедер", основної частини з інформацією та нижньої частини, яка англійською називається "футер". Верхня частина часто містить навігацію, яка дозволяє переміщатися по сторінках вебсайту. Нижня частина зазвичай містить коротку інформацію про власника, його пошту, номер телефону, адресу, а іноді список посилань на сторінки, які вважаються найбільш популярними або важливими для інформування користувача. Останньою, але не менш важливою є основна частина, яка містить інформацію з різних сторінок, будь то зображення, відео або просто текст.

Окрім front-end, сайт має внутрішню частину, яка відповідає за тимчасове зберігання інформації, наприклад, записуючи деякі дані в файли cookie або в зовнішню базу даних. Ця частина називається back-end або API. Завдяки цій частині сайт може зберігати зміни, внесені користувачем під час роботи з сайтом.

Back-end частина майже завжди є єдиною частиною сайту, яка має доступ до бази даних, через що є невід'ємною частиною при розробці та проєктуванні вебзастосунків з можливістю створювати та змінювати дані.

З таких елементів складаються і вебсайти, які мають схожу мету та тематику з проєктом цієї роботи. Приклади таких сайтів буде розглянуто у наступному розділі

1.3 Наявні рішення

1.3.1 ClickUp

ClickUp [1] дозволяє легко організувати та розвивати відносини з клієнтами за допомогою візуалізації даних, автоматизації маркетингу, комунікацій та багато іншого. Функції та інструменти повністю налаштовуються, тому мультифункціональні команди, такі як маркетологи, інженери з підтримки та фінансисти, можуть використовувати ClickUp для управління своїми завданнями і допомагати швидше укладати угоди.

Проте у цієї системи є і мінуси, найголовніший з них це те, що ця система в основному застосовується як менеджмент завдань для різних команд та як місце зберігання кінцевої інформації. Використовуючи цю систему неможливо вести облік працівників, клієнтів або товару.

Побачити сайт можна на рисунку 1.1.

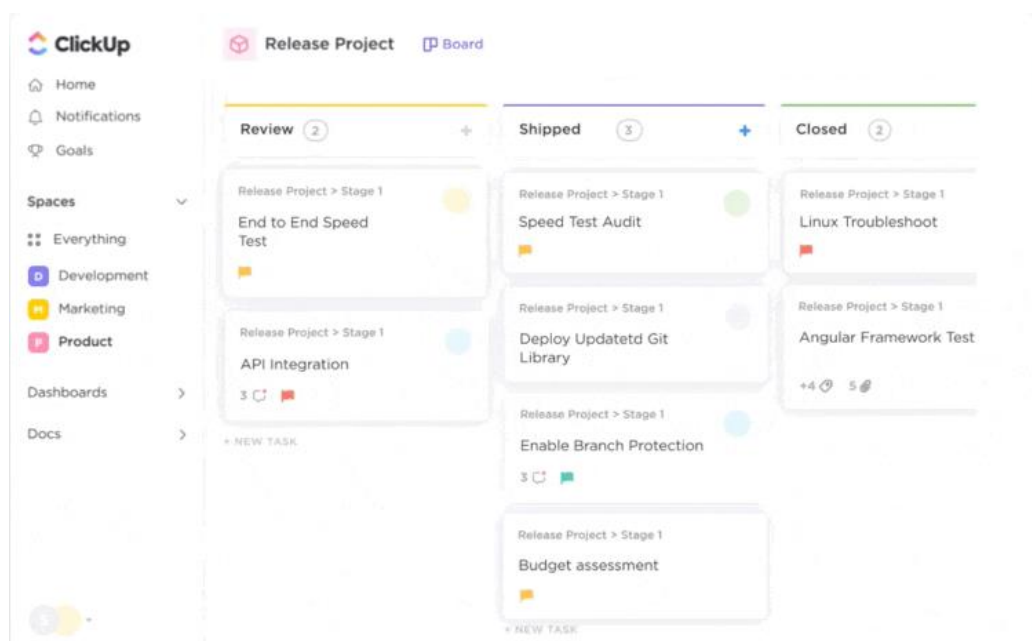


Рисунок 1.1 – Сайт ClickUp [1]

1.3.2 Zendesk

Zendesk [2] - це платформа для обслуговування клієнтів, яка допомагає компаніям надавати підтримку своїм клієнтам. Вона пропонує набір інструментів, які допомагають компаніям керувати взаємодією з клієнтами, включаючи систему продажу квитків, функціонал чату та базу знань. Zendesk інтегрується з різноманітними іншими програмними платформами, що дозволяє компаніям легко надавати підтримку по всіх каналах.

Проте як і попередня система, ця система не надає функціоналу для ведення обліку клієнтів, працівників товарів тощо, що накладає обмеження на сфери або цілі, для яких цю систему можна використати без допомоги інших систем.

Сайт можна побачити на рисунку 1.2.

Deal Name	Pipeline Stage	Value	Ownership	Last Activity Date	Days Since Last Communication
Alpha Recruiter	Abatz	\$4,000.00			
BD Insurance	Asaaron Asaronson	\$250,000.00			
Bridge Motors	Abba Vitali	\$125,000.00			
Landscape Designs	Abbie Merck	\$6,000.00			
ModTech	Asaaron Asaronson	\$25,000.00			
Moxy Ad Agency	Adah Starton	\$77,000.00			
Truckee Automotive	Ahmad Loffhead	\$150,000.00			

Summary: TOTAL DEALS 7 / 7, TOTAL VALUE \$637K / \$637K, AVERAGE VALUE \$91K / \$91K

Рисунок 1.2 – Сайт Zendesk [2]

1.3.3 Microsoft Dynamics

Microsoft Dynamics [3] - це лінійка програм для бізнесу, яку пропонує компанія Microsoft. Вона охоплює різні аспекти бізнесу, від фінансів і операцій

до продажів і маркетингу. Dynamics покликаний допомогти організаціям керувати фінансами, операціями, продажами, маркетингом та обслуговуванням клієнтів.

Ця система має більш обширний функціонал ніж попередні дві, проте вона також більше розрахована на бізнеси, які вже мають свою систему обліку товару, робітників, замовлень та не дозволяє використовувати лише її для виконання усіх необхідних робіт.

Сайт зображено на рисунку 1.3.

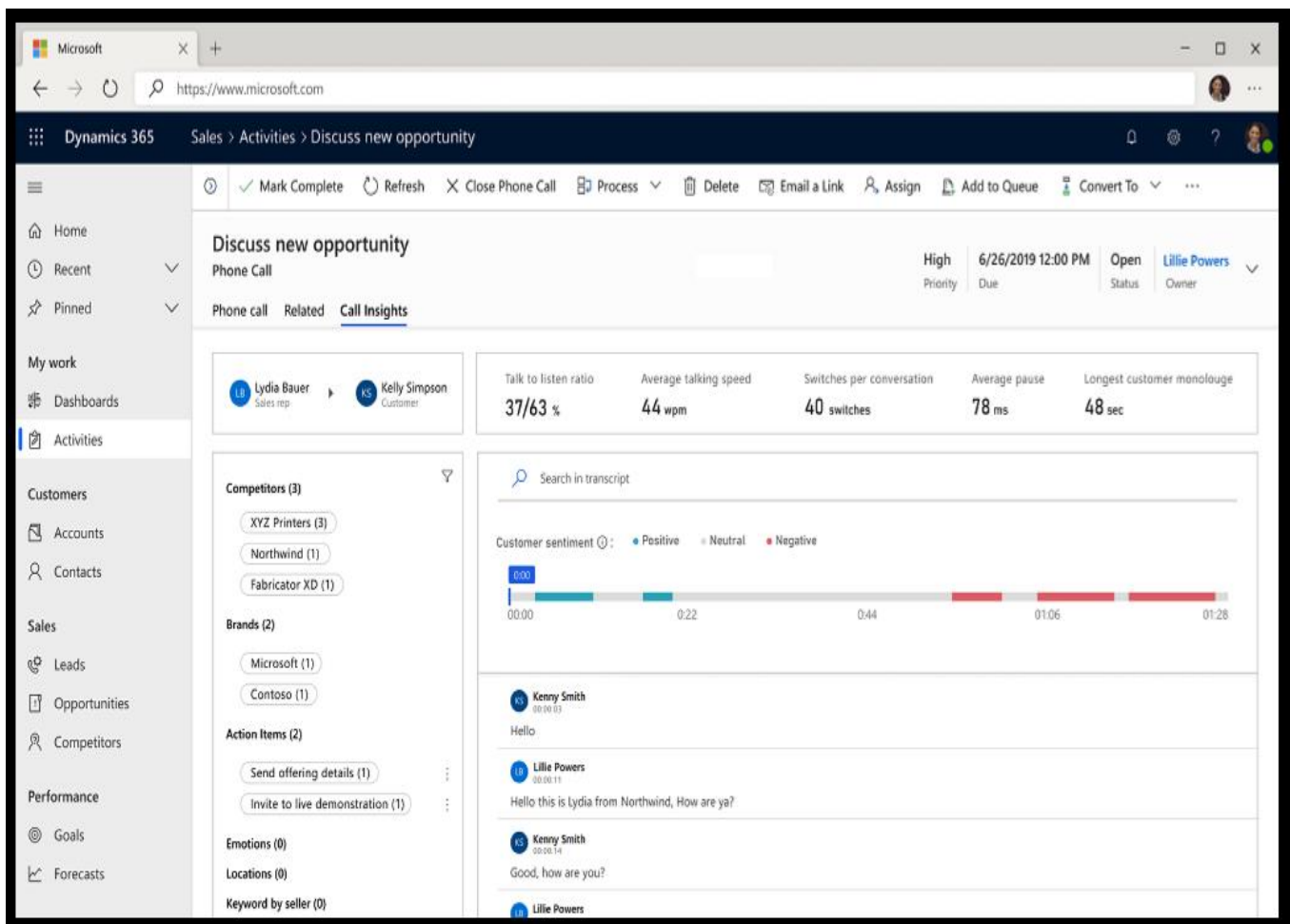


Рисунок 1.3 – Сайт Microsoft Dynamics [3]

Порівняння сайтів наведено на таблиці 1.1.

Таблиця 1.1 – Порівняння сайтів аналогів

Назва	Переваги	Недоліки
Сайт Clickup	Дозволяє: 1. Створювати, призначати та відстежувати завдання для окремих осіб або команд; 2. Контролювати час, витрачений на завдання та проєкти; 3. Сприяти командній роботі та комунікації всередині платформи; 4. Ставити завдання та відстежувати прогрес у досягненні конкретних цілей.	Немає можливості управління персоналом, клієнтами, замовленнями, продажами та складом магазинів.
Сайт Zendesk	Дозволяє: 1. Візуалізувати та управляти можливостями продажів на різних етапах; 2. Організовувати та відстежувати дії клієнтів; 3. Отримувати інформацію про ефективність та тенденції продажів; 4. Підключається до електронної пошти для відстеження комунікацій; 5. Має можливість отримати доступ до функцій CRM на ходу за допомогою мобільного застосунку.	Немає можливості управління персоналом, клієнтами, замовленнями та складом магазинів.
Сайт Microsoft dynamics	1. Комплексний пакет, що включає CRM, планування ресурсів підприємства та інші бізнес-інструменти; 2. Інсайти на основі штучного інтелекту для підвищення ефективності продажів; 3. Створення та керування цільовими маркетинговими кампаніями; 4. Безперешкодне підключення до Office 365, Teams та інших продуктів Microsoft.	Немає можливості управління персоналом, клієнтами, замовленнями та складом магазинів.

1.4 Висновки

У цьому розділі було проаналізовано предметну область, а саме вебсайти, які надають CRM-послуги, описали компоненти вебсайтів та більш детальний опис компонентів з їх можливим наповненням. Було знайдено декілька існуючих рішень та порівняно їх між собою. Однак, незважаючи на те, що було знайдено 3 приклади подібних сайтів, вони все одно не задовольняють головному завданню, а саме можливості повного контролю та управління всіма основними компонентами мережі магазинів, такими як співробітники, клієнти, товари, замовлення тощо.

2 ОПИС ПРОГРАМИ

Ціль проєкту – розробка вебсайту, призначеного для аналізу управління мережею магазинів одягу. Вебсайт матиме декілька таблиць для зберігання даних про працівників, покупців, товари тощо. Окремо буде реалізовано функціонал у вигляді вкладки зі звітами, які можуть бути корисними для менеджера.

2.1 Вибір та обґрунтування мови програмування

2.1.1 Порівняння мов програмування

Наразі найчастіше використовуваними мовами програмування для розробки вебсайтів та API є JavaScript (JS)[4], TypeScript (TS)[5] і PHP[6]. PHP найстарша з них і з кожним роком втрачає популярність, в той час JS та TS все більше стають популярнішими. Особливою популярністю користується TS, яка є послідовником JS, але вже з типізацією. Однак попри те, що TS вважається окремою мовою, свого компілятора вона не має, тому для роботи коду з TypeScript, код перетворюється на JS код та після цього виконується.

Переваги та недоліки обраних мов програмування для розробки вебсайтів можна побачити на таблиці 2.1.

Таблиця 2.1 – Порівняння мов програмування

Назва	Переваги	Недоліки
JS	- Через те що немає типізації прискорюється розробка; - Наявність великої кількості допоміжних бібліотек.	Через брак типізації можна пропустити помилки пов'язані з даними, які повертає функція.

Продовження таблиці 2.1

Назва	Переваги	Недоліки
TS	1. Наявність типів допомагає зручніше розробляти великі проєкти зручнішою; 2. Завдяки прт є доступ до великої кількості бібліотек, а сама мова має чимало вбудованих функцій.	1. Наявність явної типізації ускладнює і робить довшим початковий етап розробки через необхідність створення інтерфейсів.
PHP	1. Мова не конфліктує з html-розміткою, що дозволяє використовувати PHP-частину безпосередньо під час розмітки; 2. Мова достатньо проста, щоб нею міг користуватися будь-хто.	1. Мова стає менш популярною через появу зручніших конкурентів; 2. Вебсайти, написані на PHP, менш безпечні, ніж застосунки, написані іншими мовами.

В якості мови програмування для проєкту було обрано TS. Цей вибір був зроблений тому, що PHP вже застаріла і потребує переробки, JS є недостатньо описовою через відсутність явної типізації, що може створити зайвих проблем при розробці середніх та великих проєктів, особливо на back-end частині, яка у цьому проєкті відіграватиме значну роль. Окрім того TS має набагато зручнішу платформу для розробки, яка також зробить процес розроблення простішим та більш безпечнішим від можливих вразливостей.

2.1.2 Порівняння фреймворків/бібліотек/платформ

Фреймворк – це основа для розробки front-end інтерфейсу програми. Існує кілька популярних і розвинених фреймворків і бібліотек: React[7], Vue[8] та Angular[9].

Щоб візуалізувати різницю, можна побачити їх порівняння в таблиці 2.2.

Таблиця 2.2 – Порівняння front-end фреймворків/бібліотек

Назва фреймворку / бібліотеки	Переваги	Недоліки
React	1. Завдяки технології ReactDOM, має високу швидкість рендеру компонентів; 2. Використовує функціональне програмування; 3. Підтримка розробки за допомогою TS та JS.	1. Через прагнення розробників переходити на функціональне програмування, об'єктно орієнтоване програмування перестає бути актуальним; 2. Перемішані розмітка та скрипти можуть заплутати.
Vue	1. Документація описує всі вбудовані засоби фреймворку; 2. Дозволяє створювати проекти різної величини. 3. Займає невеликий обсяг пам'яті	1. Не має значної кількості додаткових ресурсів та бібліотек; 2. Буде проблемою переробити проект який робився з використанням іншого фреймворка.
Angular	1. Має значну підтримку TypeScript; 2. Завдяки детальній документації не буде проблемою знайти опис будь-якого елементу системи.	1. Слабша продуктивність у порівнянні з конкурентами 2. Містить велику кількість структур для використання, через що ускладнюється вивчення фреймворку.

Окрім вибору фреймворку для front-end необхідно ще обрати бібліотеку/платформу для back-end. Для порівняння було обрано Express.js[10] та Nest.js[11]. Їх порівняння можна побачити на таблиці 2.3.

Таблиця 2.3 – Порівняння back-end платформ/бібліотек

Назва фреймворку / бібліотеки	Переваги	Недоліки
Express.js	1. Має досить швидкий старт. Для початку функціонування API майже не потрібно нічого налаштовувати;	1. Не підходить для великих проектів та середніх, що планують розширятися;

Продовження таблиці 2.3

Назва фреймворку / бібліотеки	Переваги	Недоліки
Express.js	2. Проста структура, що не потребує багатьох допоміжних елементів; 3. Має багато бібліотек що розширюють функціонал.	2. Оскільки має досить просту структуру та не має можливості підтримувати роботу з TS складно відбувається оновлення та подальше розширення.
Nest.js	1. Має обширну структуру проєкту, що спрощує пошук та зміну окремих елементів у великих проєктах; 2. Має великий обсяг бібліотек, що допомагають у розробці та розширюють функціонал; 3. З самого початку працює з TS; 4. Має вбудовані методи для роботи з запитами, у тому числі метод захисту від сторонніх.	1. Через велику структуру має повільний старт бо з'являється необхідність створення цієї структури; 2. Не підходить для маленьких API, через великий обсяг роботи необхідний для старту роботи API.

Фреймворком для front-end було обрано React через можливість поєднання розмітки з мовою JS/TS та наявність великої кількості допоміжних бібліотек.

Для back-end було обрано Nest.js через значну підтримку TS та простоту подальшої підтримки та розширення проєкту. Також важливим фактором вибору стала наявність значної кількості вбудованих методів для роботи з запитами.

2.1.3 Бібліотека React

Це одна з найпопулярніших бібліотек з багатьма додатковими функціями, які, звісно, мають бути попередньо встановлені в проєкті. Це бібліотека на основі HTML, яка була розширена за допомогою мови програмування JavaScript (JS). Він називається JSX, розширення React для синтаксису JS. Це виглядає як

звичайний скрипт, написаний на JS, який повертає структуру, дуже схожу на HTML, у функції.

Такий спосіб написання структури сторінки дозволяє спростити динаміку інформації завдяки наявності JS-коду всередині елементів.

Крім того, React має революційну систему оновлення компонентів сторінки. До появи цієї інновації оновлення сторінки запускалися вручну, після чого всі компоненти сторінки створювалися заново. Розроблена система самостійно оновлює тільки змінені компоненти, у цьому допомагає бібліотека Redux, з використанням якої керується стан застосунку.

2.1.4 Платформа Nest.js

Nest – це фреймворк для створення ефективних, масштабованих серверних застосунків на Node.js. Він використовує сучасний JavaScript, побудований на TypeScript (зберігає сумісність з чистим JavaScript) і поєднує в собі елементи ООП (об’єктно-орієнтованого програмування), ФП (функціонального програмування) і ФРП (функціонально-реактивного програмування).

Всередині Nest.js реалізована з використанням технології Express. Також платформа сумісна з багатьма плагінами, що дозволяє використовувати безліч сторонніх плагінів, які доступні.

Протягом останніх років, завдяки Node.js, JS став обов’язковою частиною Інтернету як для front-end, так і для back-end застосунків, що привело до створення бібліотек та фреймворків Angular, React та Vue, що значно підвищують швидкість та зручність розробки та допомагають створювати швидкі застосунки, які легко тестувати та розширювати. На back-end частині існує багато різних бібліотек та інструментів для Node.js, але вони не вирішують важливу проблему – архітектуру.

Nest має на меті надати архітектуру застосунків «з коробки», яка дозволяє без особливих зусиль створювати застосунки, що легко тестуються, масштабуються та підтримуються.

2.2 Обґрунтування вибору середовища розробки

Середовище розробки не впливає напряду на проєкт, але воно впливає на розробника та на швидкість розробки. Адже завдяки додаткам до середовища розробки може пришвидшитись оформлення коду, наприклад існують додатки які підказують місця у яких можна написати код красивіше, або зрозуміліше ніж зараз.

Обрано було перевірене середовище розробки, а саме Visual Studio Code (VSCode). Це IDE розробляється компанією Microsoft, і її головною особливістю є наявність великої кількості розширень, що доповнюють середовище розробки та роблять з нього щось більше, дозволяючи керувати контейнерами Docker, налаштовувати лінтери, такі як ESLint. VSCode має розширення, пов'язані з Git'ом, і багато іншого. Але, звичайно, головна перевага цієї IDE перед багатьма іншими – це те, що вона безкоштовна.

Порівняння середовищ розробки можна побачити в таблиці 2.4.

Таблиця 2.4 – Порівняння середовищ розробки

Назва	Переваги	Недоліки
VSCode	1. Продукт повністю відкритий та безкоштовний; 2. Має багато розширень. 3. Не сповільнює розробку під час сильних навантажень.	1. Майже немає вбудованих функцій, які б прискорювали написання коду; 2. Перевантаження розширеннями може погано вплинути на продуктивність.

Продовження таблиці 2.4

Назва	Переваги	Недоліки
WebStorm	1. Досить швидкий; 2. Має велику кількість вбудованих функцій, включаючи перевірку коду, що значно спрощує написання коду. Має можливість встановлювати розширення, які збільшують функціональність; 3. Має можливість встановлення розширень, які розширюють функціональність.	1. Потрібна платна підписка; 2. Користувач може не бачити багато доступних функцій, бо не потребує їх.

2.3 Опис складників програми

Вебсайт складається з 2 основних частин: перша частина - це front-end, сторінки, анімація, переходи тощо, а друга частина - це back-end, який взаємодіє з базою даних. Обидві частини відіграють важливу роль у функціонуванні вебсайту, але перша частина є більшою через необхідність опису розмітки вебсайту.

2.4 Використані бібліотеки

У інтерфейсній частині багато бібліотек використовуються для керування внутрішнім станом, тобто для тимчасового зберігання поточної інформації та даних. Було використано такі технології:

- Redux[12];
- Redux-thunk;
- js-cookie.

Redux — найпопулярніша бібліотека для React. Ця бібліотека надає можливість створювати та керувати станом програми. Стан додатка схожий на невелику тимчасову базу даних, яку використовують для зберігання інформації, яка зараз використовується. Його основна мета — зменшити кількість запитів до

бази даних, але цей стан ніколи не може замінити базу даних. Крім того, Redux утворює зв'язок фреймворка з бібліотекою. Завдяки цьому зв'язку React відстежує стан програми та оновлює елементи зі зміненням станом.

Redux-thunk — це додаток до бібліотеки Redux, який розширяє функціонал, дозволяючи використовувати асинхронні операції при зміні стану. Це спрощує отримання та використання даних з бази даних, оскільки отримання інформації -це асинхронна операція.

Js-cookie — це бібліотека, яка надає додатковий функціонал для обробки файлів cookie браузера. У файли cookie вебсайт може записувати інформацію, необхідну для подальшого використання під час роботи. Файли cookie стали необхідними на сучасних вебсайтах, усуваючи необхідність запису цієї інформації в базу даних для кожного окремого користувача. На розробленому прикладі сайту можна побачити один із варіантів використання файлів cookie, тобто запис інформації авторизованих користувачів. Ім'я та результат авторизації записуються у файл cookie, і наступного разу, коли користувач відвідує сайт, користувачу буде дозволено або заборонено доступ до сайту.

Окрім технологій керування станом були задіяні такі бібліотеки:

- React-dom;
- Axios;
- MUI[13];
- React-material-ui-form-validator;
- Notistack[14];
- Eslint.

React-dom – бібліотека, яка дозволяє переходити між сторінками без видимих підвантажень нової сторінки. Також ця бібліотека допомагає розробляти навігацію по сайту, маючи власні інструменти навігації, які дозволяють просто записати існуючі кінцеві точки, що містять інформацію, і, в разі введення неправильного маршруту, перенаправляти на обрану сторінку.

Axios – використовується для надсилання запитів до API. Вона має розширений функціонал, що дозволяє створювати шаблони, які

використовуються для створення записів, керувати заголовком і тілом запитів, а також вказувати відповідь на запит. Також вона має кілька типів запитів, крім звичайних GET, POST, PUT, DELETE, бібліотека відправляє такі запити як OPTIONS, PURGE, LOCK тощо.

MUI – бібліотека, яку використовують для більш швидкої розмітки. Бібліотека містить велику кількість різноманітних компонентів, які дуже зручні для створення сторінок. При необхідності можна модифікувати елементи, змінити їх колір або навіть написати власний компонент на основі одного з існуючих. Компонент Grid використовувався найчастіше. Цей компонент дозволяє розділити простір на екрані на частини і розподілити різним компонентам ці частини. Це робить дуже зручним створення адаптивного інтерфейсу. Бібліотека надає ще багато інших інструментів, які були використані при розробці, оскільки бібліотека пропонує безліч безкоштовних інструментів, включаючи звичні всім іконки, які не потребують додаткових пояснень.

React-material-ui-form-validator – це бібліотека, завдяки якій створення форм та їх валідація спрощується на скільки це можливо. Бібліотека надає текстове поле з готовою валідацією, критерії якої описуються для кожного поля окремо, також можна створювати власні правила. Помилки валідації зручно відображаються знизу поля.

Notistack – бібліотека, за допомогою якої було реалізовано відображення помилок від back-end. Ця бібліотека реалізує впливаючі повідомлення, які з'являються у верхній правій частині екрану, коли back-end повертає помилку при запиті.

Eslint – бібліотека, яку використовують для написання гарного та зручного для читання коду. У спеціальному файлі створюються правила за якими перевіряються файли з кодом та виділяються місця, які можна написати красивіше.

Вищезазначені технології були використані для розробки front-end, але проєкт також має back-end частину, яка була реалізована за допомогою мови

програмування TS. Для розробки back-end були використані наступні інструменти:

- Nest.js;
- Joi[15];
- Mongoose[16];
- Class-validator[17];
- Class-transformer.

Nest.js — це популярний фреймворк з відкритим вихідним кодом для створення масштабованих і підтримуваних серверних застосунків за допомогою JavaScript або TypeScript. Він побудований на основі Node.js і використовує концепції та шаблони як традиційних серверних фреймворків, таких як Express.js, так і сучасних інтерфейсних фреймворків, таких як Angular. Nest.js має на меті забезпечити надійну, організовану та ефективну структуру для розробки серверних застосунків, що робить його добре придатним для створення API, мікросервісів та вебзастосунків. Поєднання сучасних функцій, архітектурних шаблонів та зростаючої екосистеми робить його привабливим вибором для розробників, які прагнуть створювати масштабовані та підтримувані застосунки, незалежно від того, чи створюють вони RESTful API, мікросервіси або інші серверні рішення.

Joi — це бібліотека для перевірки об'єктів за конкретними правилами. У проєкті вона використовувалась для валідації файлів з налаштуваннями. Для цього в кожному файлі створювалися налаштування з правилами за якими потрібно перевіряти, а під час запуску програми відбувалася перевірка.

Mongoose — це бібліотека взаємодії з базою даних. Після вибору MongoDB, Mongoose стала необхідною для роботи з таблицями. Ця бібліотека дозволяє створювати схеми таблиць, відповідно до яких під час підключення до бази буде створено таблицю, якщо вона ще не існує.

Class-validator — це бібліотека, яка реалізує валідацію даних з запитів. Ця бібліотека дуже зручно комбінується та використовується разом з Nest.js. Для того, щоб валідація працювала, достатньо створити клас DTO та за допомогою

декораторів визначити кожному полю правила, за якими потрібно їх перевіряти. Бібліотека має великий обсяг готових правил та надає можливість створити власні.

Class-transformer – це бібліотека часто використовується як допоміжна для class-validator. У цьому проєкті вона використовувалась для правильної валідації об'єктів, що знаходяться всередині інших об'єктів. Для такої валідації необхідно перетворити звичайний об'єкт у екземпляр класу, який матиме в собі правила для валідації.

Це були основні бібліотеки, які мали значний вплив на роботу застосунку, але були використані ще бібліотеки з меншою важливістю, а саме dotenv, md5, reflect-metadata, cookie-parser та eslint.

Dotenv допомагає з конфігурацією застосунку, а точніше додає змінні з файлу .env до змінних оточення застосунку, завдяки чому ці змінні можна використовувати під час роботи застосунку.

Md5 – бібліотека для створення md5 хешу. Цей хеш використовувався для збереження паролю. Для того щоб не зберігати пароль у його початковому вигляді з нього робиться md5 хеш та зберігається до бази, а при спробі авторизації перевіряється хеш у базі з тим, який вийшов з паролем який ввів користувач. Особливість формування md5 хешу така, що він не може буде розшифрований назад у звичайний текст, що робить такий метод зберігання паролів набагато безпечнішим ніж зберігання паролю у чистому вигляді.

Reflect-metadata – бібліотека, яка дозволяє переглядати метадані запиту. У проєкті вона використовувалася для того щоб уникнути перевірки авторизації користувача для деяких запитів, де вона не потрібна, наприклад для запиту авторизації.

Cookie-parser – бібліотека, яка дозволяє back-end працювати з cookie. Окрім того бібліотека дозволяє шифрувати значення cookie, що і було використано у проєкті для збереження accessToken. Щоб не можливо було його дешифрувати та отримати інформацію про користувача, яка міститься у токени.

Eslint – так само, як і у front-end бібліотека використовується для збільшення читабельності та зовнішнього вигляду коду.

2.5 Обґрунтування вибору системи керування базами даних

MongoDB[18] було обрано як систему керування базами даних (СКБД) для проєкту. Це нереляційна база даних, тобто база даних, яка не вимагає чіткої структури і може зберігати різноманітну інформацію. Подібний спосіб зберігання інформації є менш безпечним для використання у великих проєктах, бо відсутність контролю структури даних у таблицях може призвести до помилок через дуже різні дані у одній таблиці.

Основними конкурентами при виборі СКБД є MySQL[19] і PostgreSQL[20]. Ці бази даних реляційні, тому мають чітку структури таблиць. Вони також підходять для цього проєкту, але головною перевагою MongoDB є те, що MongoDB може зберігатися на віддаленому сервері, а не на локальній машині чи хостингу.

Порівняння СКБД можна побачити на таблиці 2.5.

Таблиця 2.5 – Порівняння СКБД

Назва	Переваги	Недоліки
MongoDB	1. NoSQL-база даних, яка використовує гнучку, безсхемну модель даних, що робить її добре придатною для застосунків з динамічними структурами даних, що розвиваються;	1. MongoDB жертвує певним рівнем узгодженості даних на користь масштабованості та продуктивності, оскільки не повністю відповідає властивостям ACID; 2. Бази даних MongoDB можуть швидко зростати, і управління сховищем може стати проблемою при великомасштабному розгортанні.

Продовження таблиці 2.5

Назва	Переваги	Недоліки
	2. Дані зберігаються у форматі BSON (Binary JSON), з яким легко працювати розробникам, особливо у вебзастосунках; 3. Має хмарні бази даних, що дозволяє безкоштовно зберігати на них данні маючи обмеження на кількість запитів.	
PostgreSql	1. Система реляційних баз даних, яка відповідає властивостям ACID, що робить її придатною для застосунків, які вимагають високої узгодженості та надійності даних; 2. Підтримує потужну мову запитів SQL, яка дозволяє отримувати складні та структуровані дані та маніпулювати ними; 3. Має зрілу та активну спільноту з відкритим вихідним кодом, яка надає обширну документацію, плагіни та інструменти.	1. Хоча PostgreSQL може обробляти великі набори даних, вона може не так легко масштабуватися по горизонталі, як деякі NoSQL бази даних, такі як MongoDB; 2. Реляційна модель даних може бути складнішою в налаштуванні та підтримці, особливо для застосунків зі схемами даних, що швидко змінюються.

Обрано було саме MongoDB, здебільшого через можливість створити хмарну базу даних, що у комбінації з Vercel[21] дозволило абсолютно безкоштовно викласти проєкт на сервер.

2.6 Висновки розділу

У цьому розділі було обґрунтовано вибір основних складників проєкту. Було описано використані бібліотеки, фреймворк тощо.

У результаті було обрано мову програмування TypeScript, фреймворк React для front-end та платформу Nest.js для back-end та Visual Studio Code як середовище розробки.

3 РОЗРОБКА ПРОГРАМИ

3.1 Схема БД

Для керування структурою таблиць бази даних бібліотека надає можливість створювати схеми таблиць.

Було створено 7 схем сутностей, з них 6 основних, та 1 допоміжна. Допоміжна схема це warehouse схема, вона використовується для збереження інформації про наявність різних товарів та їх кількість у різних магазинах.

Схеми та їх опис наведено у таблиці 3.1.

Таблиця 3.1 – Схеми сутностей БД

Сутність	Поле	Обов'яз- кове поле	Опис поля
Client	createdAt	+	Дата створення
	createdBy	-	Користувач, який створив
	updatedAt	-	Дата оновлення
	updatedBy	-	Користувач, який оновив
	deletedAt	-	Дата видалення
	id	+	Ідентифікатор
	name	+	Ім'я
	email	+	Електронна пошта
	phoneNumber	+	Номер телефону
	size	-	Розмір одягу
Cloth	createdAt	+	Дата створення
	createdBy	-	Користувач, який створив
	updatedAt	-	Дата оновлення
	updatedBy	-	Користувач, який оновив
	deletedAt	-	Дата видалення
	id	+	Ідентифікатор
	name	+	Назва одягу
	price		Ціна
	desc	+	Опис

Продовження таблиці 3.1.

Сутність	Поле	Обов'яз- кове поле	Опис поля
Cloth	availableSizes	+	Розміри, які можуть бути у цього одягу
Delivery	createdAt	+	Дата створення
	createdBy	-	Користувач, який створив
	updatedAt	-	Дата оновлення
	updatedBy	-	Користувач, який оновив
	deletedAt	-	Дата видалення
	id	+	Ідентифікатор
	deliveredTo	+	Ід магазину, у який була поставка
	deliveredFrom	-	Ід магазину, з якого забрали товар(при внутрішній)
	totalAmountDelivered	+	Загальна кількість доставлених товарів
	typeOfDelivery	+	Тип доставки(внутрішня, зовнішня)
	price	-	Ціна, за якою купили товари з поставки(при зовнішній)
	clothDelivered	+	Одяг, який було доставлено
Order	createdAt	+	Дата створення
	createdBy	-	Користувач який створив
	updatedAt	-	Дата оновлення
	updatedBy	-	Користувач, який оновив
	deletedAt	-	Дата видалення
	id	+	Ідентифікатор
	clientId	+	Ід клієнта замовлення
	clothIdList	+	Інформація про товари, які було замовлено
	status	+	Статус замовлення

Продовження таблиці 3.1.

Сутність	Поле	Обов'яз- кове поле	Опис поля
Order	price	+	Ціна замовлення
Staff	createdAt	+	Дата створення
	createdBy	-	Користувач, який створив
	updatedAt	-	Дата оновлення
	updatedBy	-	Користувач, який оновив
	deletedAt	-	Дата видалення
	id	+	Ідентифікатор
	name	+	Ім'я працівника
	email	+	Електронна пошта
	isAdmin	+	Чи є працівник адміністратором
	password	-	Пароль адміністратора
	position	+	Позиція
	storeId	+	Номер магазину, у якому працює
	isHidden	+	Чи є працівник прихованим (використовується для виділення першого адміна, який буде створюватися до початку роботи системи)
Store	createdAt	+	Дата створення
	createdBy	-	Користувач, який створив
	updatedAt	-	Дата оновлення
	updatedBy	-	Користувач, який оновив
	deletedAt	-	Дата видалення
	id	+	Ідентифікатор
	address	+	Адреса магазину
	isActive	+	Чи є магазин активним(чи працює)

Кінець таблиці 3.1.

Сутність	Поле	Обов'яз- кове поле	Опис поля
Warehouse	createdAt	+	Дата створення
	createdBy	-	Користувач, який створив
	updatedAt	-	Дата оновлення
	updatedBy	-	Користувач, який оновив
	deletedAt	-	Дата видалення
	id	+	Ідентифікатор
	clothId	+	Ід одягу
	storeId	+	Ід магазину
	size	+	Розмір одягу
	amount	+	Кількість одягу

3.2 Розробка інтерфейсу

Розробка інтерфейсу проводилась за більш розповсюдженим варіантом для CRM систем, у якому ліворуч знаходиться навігаційна панель, а праворуч від неї основний контент. Окрім основних вікон з даними, які беруться зі своїх таблиць бази було створено “Звіти”, для проведення аналітики продажів та менеджменту працівників.

На рисунку 3.1 можна побачити загальний дизайн.



Рисунок 3.1 – Загальний дизайн

Дизайн має визначені кольори які були використані, це були: Police Blue (#395C6B), Beau Blue (#BCD3F2) та Dark Sea Green (#78C091) кольори. Кольору Police Blue було зроблено header, Beau Blue було зроблено панель навігації, а обраний елемент навігації має трохи темніший колір Aero (#82b0eb), кольором Dark Sea Green було виконано більшість елементів керування, тобто кнопки, текстові поля, поля з вибором, поля-відмітки тощо.

Для відображення інформації з основних таблиць бази даних було використано елемент таблицю. Окрім назви стовбців та самих даних, таблиця має кнопки для створення запису та для фільтрації, також деякі назви стовбців можуть бути натиснуті для зміни сортування. А кожний елемент таблиці має кнопку з трьома точками, при натисканні на яку буде відкрито меню вибору, у якому будуть знаходитися додаткові дії з записом, найчастіше це редагування та видалення, але у деякій таблицях, як наприклад у таблиці працівників є кнопки надання та прибирання прав адміністратора.

Таку таблицю можна побачити на рисунку 3.2, а меню вибору на рисунку 3.3.

Обліковий номер ↑	Електронна пошта	ПІБ	Посада	Чи є адміністратором	ID магазину	Час останнього оновлення	Хто останній раз оновив	СТВОРИТИ	▼
1	string@s.com	test	string	✓	1	2023-09-21T08:06:08.027Z	string@s.com	...	
2	sheverdinkin@gmail.com	qwewqewqewq	qwewqewqe	✓	1	2023-09-20T11:16:19.804Z	string@s.com	...	
4	test@test.com	test	test	✗	123	2023-09-21T08:10:36.612Z	string@s.com	...	
6	tes@q.com	rtet	asdf	✓	12	2023-09-21T08:10:48.840Z	string@s.com	...	
7	qweqwe@qwe.qwe	qwe	qwe	✗	12	2023-09-20T11:44:11.130Z	string@s.com	...	
8	asf@qwr.asf	qwrqwr	2134rwq	✗	12	2023-09-20T11:44:20.722Z	string@s.com	...	
9	sheverdinkin1@gmail.com	qwe	qwe	✗	12	2023-09-20T12:11:14.462Z	string@s.com	...	
10	sheverdinkin2@gmail.com	qwe	qwe	✗	1	2023-09-28T09:06:06.541Z	string@s.com	...	
11	sheverdinkinw@gmail.com	qwe	qwe	✗	2	2023-09-20T12:11:33.676Z	string@s.com	...	
12	sheverdinkin3@gmail.com	atest	123	✗	11	2023-09-28T09:05:53.500Z	string@s.com	...	

Рисунок 3.2 – Таблиця відображення даних з бази даних

Обліковий номер ↑	Електронна пошта	ПІБ	Посада	Чи є адміністратором	ID магазину	Час останнього оновлення	Хто останній раз оновив	СТВОРИТИ	▼
1	string@s.com	test	string	✓	1	2023-09-21T08:06:08.027Z	string@s.com	...	
2	sheverdinkin@gmail.com	qwewqewqewq	qwewqewqe	✓	1	2023-09-20T11:16:19.804Z	string@s.com	...	
4	test@test.com	test	test	✗	123	2023-09-21T08:10:36.612Z	string@s.com	...	

Рисунок 3.3 – Меню вибору у таблиці

Також схоже меню вибору з'являється при натисканні на кружечок з буквою імені користувача – аватар, воно дозволяє користувачу вийти з аккаунту. Меню вибору з аватару користувача можна побачити на рисунку 3.4.

Company name								Т	Вийти
Посада	Чи є адміністратором	ID магазину	Час останнього оновлення	Хто останній раз оновив	СТВОРИТИ		▼		

Рисунок 3.4 – Меню вибору з аватару

3.3 Структура проєкту

Структура проєкту досить велика, оскільки як у front-end так і в back-end файли розподілено за окремими папками, а кількість файлів перевищує 100.

Структуру front-end частини проєкту можна побачити на рисунку 3.5.

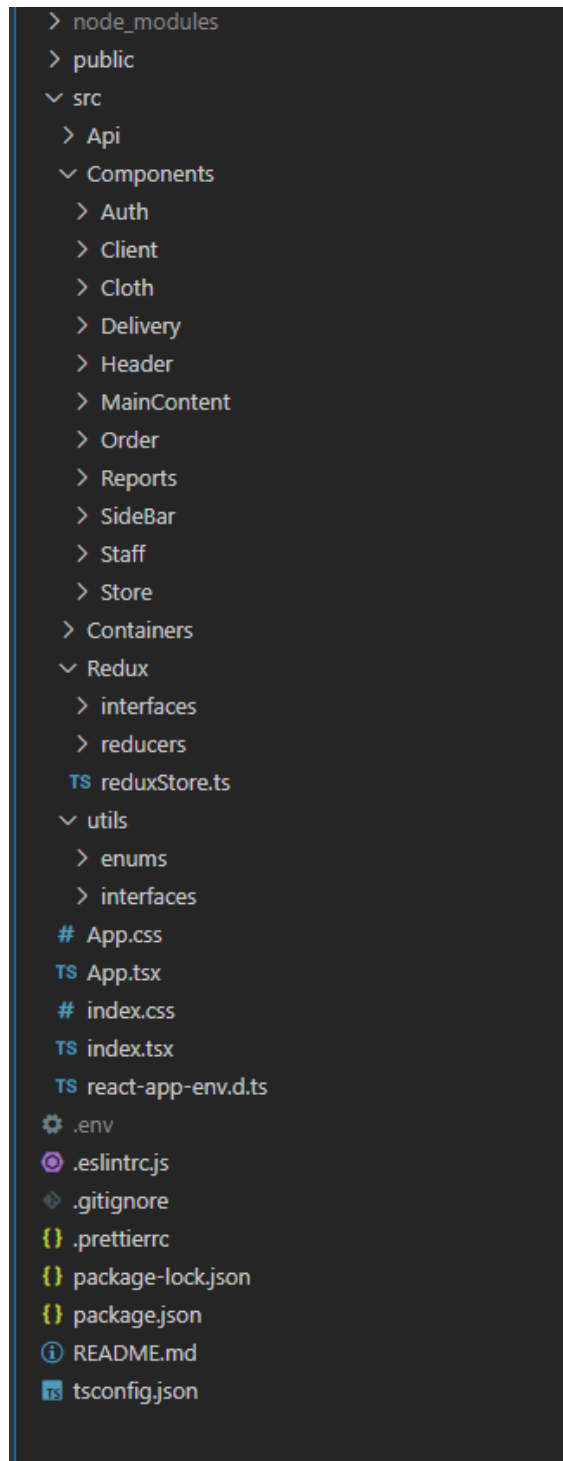


Рисунок 3.5 – Структура проєкту front-end

У корені проєкту front-end є тека src, в середині якої знаходяться усі файли проєкту, окрім налаштувань. Налаштування знаходяться на одному рівні з текою src, а саме це файли .env, .eslintrc.js, .gitignore, .prettierrc, package.json, package-log.json, tsconfig.json.

У файлі .env зберігаються локальні зміни проєкту, такий файл необхідний лише під час запуску проєкту з локальної машини. Файли .eslintrc.js та prettierrc містять у собі правила та налаштування для бібліотека eslint та prettier. Файл .gitignore містить перелік файлів, які не потрібно записувати у гіт. Файли package.json та package-log.json містять перелік бібліотек та скриптів для їх інсталювання та запуску через npm. Файл tsconfig.json містить налаштування для TypeScript, використовуючи які він перетворює TS у JS.

Тека Api містить у собі файли з методами доступу до back-end для кожного редьюсера та загальний файл який містить екземпляр axios з налаштованою адресою до back-end api. Тека Components містить усі компоненти з розміткою та CSS стилями. Containers містить контейнери компонент, які використовуються для того щоб передавати у компоненти дані та методи з редьюсерів. Тека Redux складається з двох тек та файлу, тека interfaces містить інтерфейси кожного редьюсера, а reducers – самі редьюсери. Файл reduxStore об'єднує всі редьюсери у спільний store. Тека utils містить загальні файли, які можуть бути використані будь-де. У теці enums знаходяться файли з переліками, тобто константні значення, наприклад як статуси замовлень. Тека interfaces містить інтерфейси.

Файли App.tsx/css та index.tsx/css містять стилі та розмітку ініціалізаційних файлів.

Структуру back-end частини проєкту можна побачити на рисунку 3.6.

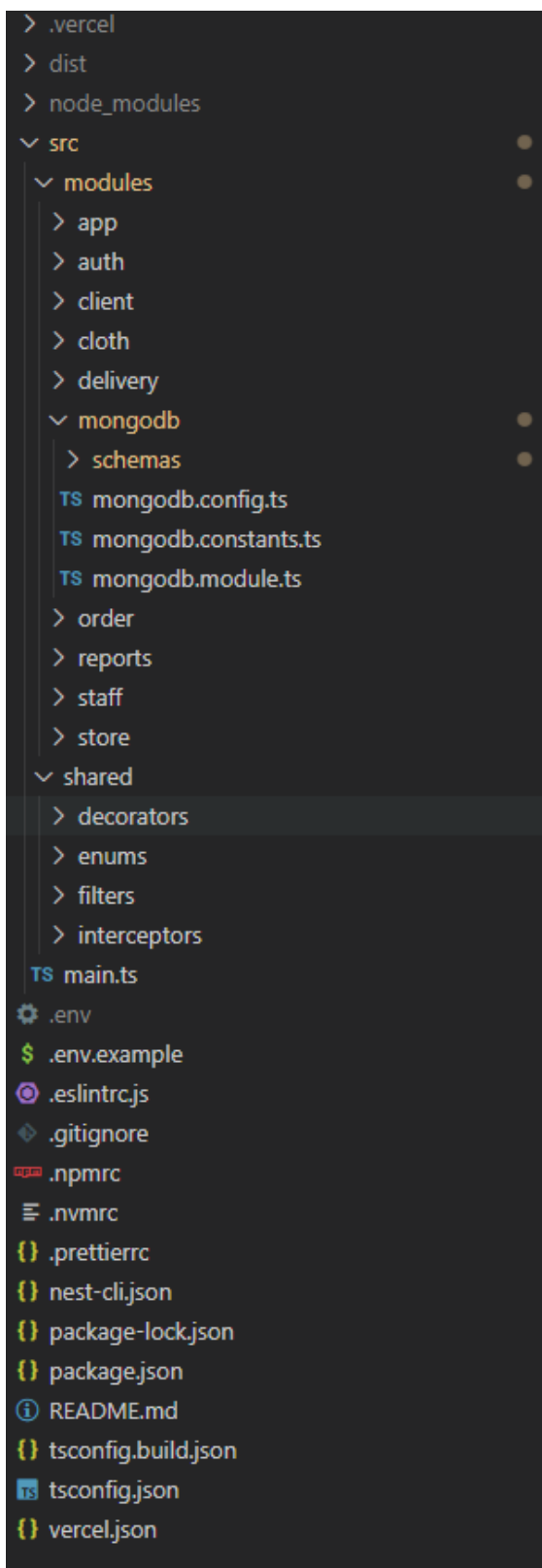


Рисунок 3.6 – Структура проєкту back-end

Проект містить 4 теки, серед них 3 згенеровані бібліотеками, це `.vercel`, `dist` та `node_modules`. Тека `.vercel` створена бібліотекою `vercel`, яка допомагає

зберігати застосунок на хостингу, `dist` містить у собі код проєкту у форматі JS, а `node_modules` – це тека, яка містить код усіх сторонніх бібліотек, які були використані.

Тека `models` містить модулі проєкту, це App модуль, який є головним та містить у середині усі інші, `mongodb` модуль містить підключення до `mongodb` бази даних, а також схеми сутностей, інші модулі мають всередині `controller`, у якому визначається які ендпоінти матиме модуль, `service`, у якому знаходиться логіка модуля, `module`, у якому визначаються залежності модуля, тобто які інші модулі він в себе імпортує, `config`, де знаходяться налаштування модуля, наприклад максимальні значення для валідації та тека `dto`, яка містить у собі файли з класами, які визначають формат відповіді кожного ендпоінта та валідацію.

`Shared` складається з `decorators`, `enums`, `filters` та `iterceptors`. `Decorators` зберігає декоратори, які використовуються у проєкті, `enums` містить переліки констант, `filters` – обробник помилок, а `iterceptors` – функції, які `Nest.js` викликає до та після виконання ендпоінта, у проєкті `interceptor` використовується для логування запитів та відповідей.

У файлі `.env` зберігаються локальні зміни проєкту, такий файл необхідний лише під час запуску проєкту з локальної машини, а у файлі `.env-example` міститься структура `.env` файлу, тобто перелік змінних які є у проєкті. Файли `.eslintrc.js` та `prettierrc` містять у собі правила та налаштування для бібліотека `eslint` та `prettier`. Файл `.gitignore` містить перелік файлів, які не потрібно записувати у гіт. `.npmrc` – містить налаштування `npm`, а `.nvmrc` визначає версію `Node.js`, яку необхідно використовувати при запуску проєкту. Файли `package.json` та `package-log.json` містять перелік бібліотек та скриптів для їх інсталиювання та запуску через `npm`. Файл `tsconfig.json` містить налаштування для `TypeScript`, використовуючи які він перетворює TS у JS. `Nestcli.js` містить налаштування `Nest.js`. Файл `.vercel` містить налаштування `Vercel` бібліотеки, за допомогою якої `back-end` та `front-end` має хост на сервері.

3.4 Використані алгоритми

У процесі розробки проєкту було написано різні алгоритми як у back-end частині так і у front-end, деякі з них були необхідні для реалізації конкретної поведінки, деякі спрощували та пришвидшували написання розмітки.

Приклад одного з алгоритмів на front-end – це алгоритм для динамічної розмітки елементів таблиці. Оскільки з back-end може прийти будь-яка кількість записів, було необхідно мати можливість відображати саме стільки скільки буде потрібно, для цього використовуючи метод “map”, який є вбудованим методом масивів у JS, створювався масив елементів таблиці для кожного запису, який прийшов з back-end. Приклад цього алгоритму на рисунку 3.7.

```

<TableBody>
  {delivers?.map((delivery) => {
    const menu = IsolatedMenu(delivery);
    return (
      <React.Fragment key={'tab' + delivery.id}>
        <TableRow>
          <TableCell align="center">
            <IconButton
              size="small"
              onClick={handleCollapseToggle(delivery.id)}
            >
              {collapseOpen.includes(delivery.id) ? (
                <KeyboardArrowUpIcon />
              ) : (
                <KeyboardArrowDownIcon />
              )}
            </IconButton>
          </TableCell>
          <TableCell align="center">{delivery.id}</TableCell>
          <TableCell align="center">
            {delivery.deliveredFrom}
          </TableCell>
          <TableCell align="center">{delivery.deliveredTo}</TableCell>
          <TableCell align="center">
            {delivery.totalAmountDelivered} шт.
          </TableCell>
          <TableCell align="center">
            {!Number.isNaN(delivery.price) ? delivery.price : ''}
          </TableCell>
          <TableCell align="center">
            {delivery.updatedAt || delivery.createdAt}
          </TableCell>
          <TableCell align="center">
            {delivery.deletedBy ||
              delivery.updatedBy ||
              delivery.createdBy}
          </TableCell>
          <TableCell align="center">
            <IconButton onClick={menu.handleMenuClick(delivery.id)}>
              <MoreHorizIcon />
            </IconButton>
            {menu.jsx}
          </TableCell>
        </TableRow>
      </TableRow>
    )
  })}

```

Рисунок 3.7 – Приклад динамічної розмітки для таблиці

Ще одним алгоритмом було розроблено динамічне відображення кнопки додавання елементу при оформленні доставки (рисунки 3.8). Така кнопка мала відображатися лише для останнього елементу. Для цього під час формування масиву елементів відстежувався порядковий номер елементу, що формується, та

у випадку, якщо це останній елемент, додавалася кнопка. Алгоритм можна побачити на рисунку 3.9.

Створити поставку

Доставка в

Тип поставки ▼

ID одягу
0

Розміри

Новий розмір ▼

ВИДАЛИТИ

ID одягу
0

Розміри

Новий розмір ▼

ДОДАТИ ТОВАР
ВИДАЛИТИ

ВІДМІНИТИ

СТВОРИТИ

Рисунок 3.8 – Елемент оформлення доставки

```

<Grid container justifyContent="space-evenly">
  {clothDeliverValidation.length === i + 1 ? (
    <Button
      color="button"
      onClick={handleAddClothDeliver(i)}
      disabled={disabled}
    >
      Додати товар
    </Button>
  ) : (
    ..
  )}

```

Рисунок 3.9 – Алгоритм додавання кнопки до елемента доставки

Також було реалізовано декілька алгоритмів для back-end. Наприклад під час створення поставки було створено алгоритм для коректного оновлення таблиці warehouse. Оскільки під час поставки кількість товару та наявність товару у конкретних магазинах змінюється, може бути необхідність створити

одночасно декілька записів у базу, для пришвидшення цього процесу та для коректного задання ID запису і був розроблений цей алгоритм. Його суть полягає у тому, що під час проходження по масиву з товарами створювався масив товарів, для яких необхідно буде створити запис у базі і створення записів трохи переносилося. Записи які вже були створені оновлювалися одразу. Після проходження всіх товарів, робилося же одне проходження вже за масивом з товарами для яких потрібно створювати запис та робилося масивне створювання, що є набагато швидшим ніж окреме створення кожного запису. Алгоритм можна побачити на рисунку 3.10.

```

80     const warehouseCreateArrays = await Promise.all(
81         dto.clothDelivered.map(async (cloth) => {
82             const warehouseClothes = await this.warehouseSchema.find({
83                 clothId: cloth.clothId,
84                 storeId: dto.deliveredTo,
85             });
86             return Promise.all(
87                 cloth.sizes.map(async (size) => {
88                     const warehouseCloth = warehouseClothes.find(
89                         (warehouse) => warehouse.size === size.size,
90                     );
91
92                     totalAmountDelivered += size.count;
93
94                     if (warehouseCloth) {
95                         await warehouseCloth.updateOne({
96                             id: warehouseCloth.id,
97                             amount: warehouseCloth.amount + size.count,
98                             updatedAt: new Date().toISOString(),
99                             updatedBy: userEmail,
100                         });
101                     } else {
102                         return {
103                             clothId: cloth.clothId,
104                             storeId: dto.deliveredTo,
105                             size: size.size,
106                             amount: size.count,
107                             createdAt: new Date().toISOString(),
108                             createdBy: userEmail,
109                         };
110                     }
111                 }),
112             );
113         }),
114     );
115
116     const warehouseCreateArray = [].concat(...warehouseCreateArrays);
117
118     const lastWarehouse = await this.warehouseSchema
119         .find(null, null, { sort: { id: -1 } })
120         .limit(1);
121
122     warehouseCreateArray.forEach((warehouseCreate, i) => {
123         if (!warehouseCreate) {
124             return;
125         }
126         this.warehouseSchema.create({
127             ...warehouseCreate,
128             id: (lastWarehouse[0]?.id || 0) + 1 + i,

```

Рисунок 3.10 – Алгоритм створення записів під час створення поставки

3.5 Використання бібліотек та фреймворків

У попередньому розділі було описано бібліотеки та фреймворки використані під час розробки. У цьому розділі буде детальніше описано яка бібліотека для чого використовувалася та як вона була використана.

3.5.1 Бібліотека React

React - це бібліотека, яка використовується для розробки всього front-end. Почати роботу з бібліотекою можна одразу після запуску команди `prx-create-react-app`. Команда ініціалізує проєкт та створює початкові файли, необхідні для запуску, наприклад створює перший компонент.

Компоненти у бібліотеці можуть бути функцією та класом. Розробники цієї бібліотеки рекомендують використовувати функціональні компоненти – всі компоненти у проєкті є функціями (рис. 3.11).

```

2  const App = (props: IAppProps) => {
3    useEffect(() => {
4      const isAuthorized = Cookies.get('isAuthorized');
5      const userName = Cookies.get('userName');
6      props.setAuthorized(isAuthorized === 'true', userName);
7    }, []);
8
9    useEffect(() => {
10     props.errors.forEach((error) => {
11       enqueueSnackbar(error, {
12         variant: 'error',
13         autoHideDuration: 5000,
14         anchorOrigin: { vertical: 'top', horizontal: 'right' },
15       });
16     });
17     props.clearError();
18   }, [props.errors]);
19
20   return (
21     <Grid container className="app" alignContent="baseline">
22       <HeaderContainer />
23       <Grid container className="mainElem" justifyContent={'center'}>
24         {props.auth?.isAuthorized === false ? (
25           <Navigate to="/auth" />
26         ) : (
27           <Grid item xs={4} sm={2.5} lg={1.3}>
28             <SideBarContainer />
29           </Grid>
30         )}
31         <Grid item xs={8} sm={9.5} lg={10.7}>
32           <MainContentContainer />
33         </Grid>
34       </Grid>
35       <SnackbarProvider maxSnack={10}/>
36     </Grid>
37   );
38 };
39
40 export default App;

```

Рисунок 3.11 – Функціональна компонента React

Бібліотека React надає методи, які надають можливість виконувати різні дії під час різних етапів створення сторінки або під час її зміни. З використанням цих методів було реалізовано перевірку на те авторизований користувач чи ні (рис. 3.12), відображення помилок (рис. 3.13), яке виконується коли до

компоненти потрапляють нові дані про помилки та завантаження даних з бази даних (рис. 3.14).

```
useEffect(() => {
  const isAuthorized = Cookies.get('isAuthorized');
  const userName = Cookies.get('userName');
  props.setAuthorized(isAuthorized === 'true', userName);
}, []);
```

Рисунок 3.12 – Перевірка користувача на авторизованість

```
useEffect(() => {
  props.errors.forEach((error) => {
    enqueueSnackbar(error, {
      variant: 'error',
      autoHideDuration: 5000,
      anchorOrigin: { vertical: 'top', horizontal: 'right' },
    });
  });
  props.clearError();
}, [props.errors]);
```

Рисунок 3.13 – Відображення помилок у сповіщеннях

```
useEffect(() => {
  props.getDeliveryThunk({
    page: pagination.page,
    limit: pagination.rows,
    sort: {
      order,
      orderBy,
    },
  });
}, []);
```

Рисунок 3.14 – Завантаження даних з бази даних

Крім того, React має зовнішню бібліотеку `react-dom`, яка дозволяє здійснювати адресацію між компонентами на основі шляху (рис. 3.15). Компонент `NavLink` дозволяє переходити між посиланнями сайту без необхідності перезавантаження сторінки, роблячи переходи між елементами

сайту дуже плавними і зручними. Також є компонент Navigate, який дозволяє перенаправляти користувача на конкретну адресу.

```
const MainContent = () =>
  <Box style={{ padding: '20px' }}>
    <Routes>
      <Route path={"/staff"} element={<StaffContainer />} />
      <Route path={"/clients"} element={<ClientContainer />} />
      <Route path={"/cloth"} element={<ClothContainer />} />
      <Route path={"/orders"} element={<OrderContainer />} />
      <Route path={"/stores"} element={<StoreContainer />} />
      <Route path={"/delivery"} element={<DeliveryContainer />} />
      <Route path={"/reports"} element={<ReportsContainer />} />
      <Route path={"/auth"} element={<AuthContainer />} />
      <Route path="*" element={<Navigate to="/staff" />} />
    </Routes>
  </Box>;
export default MainContent;
```

Рисунок 3.15 – Компонент маршрутизації

3.5.2 Платформа Nest.js

Основою back-end частини застосунку стала платформа Nest.js. Ця платформа є дуже популярною та постійно розвивається, має велике ком'юніті, що дозволяє вирішувати проблеми та знаходити рішення швидко та результативно. Платформа має великий функціонал та співпрацює з багатьма популярними бібліотеками, які розширюють можливості для розробки. Розробка на цій платформі виконується через так звані модулі, це класи, які поєднують у собі розподілення ендпоінтів, логіку для них та дозволяє використовувати інші модулі у собі. Завдяки такому підходу стає зручно розділяти логіку за секторами відповідальності, та пришвидшує пошук окремої ділянки коду, при необхідності. Приклад модуля можна побачити на рисунку 3.16.

```

@Module({
  imports: [StoreModule, ClothModule],
  controllers: [DeliveryController],
  providers: [
    {
      provide: DELIVERY_MODEL_SCHEMA,
      useFactory: (conn: Connection) => conn.model('delivery', DeliverySchema),
      inject: [MONGODB_CONNECTION],
    },
    {
      provide: WAREHOUSE_MODEL_SCHEMA,
      useFactory: (conn: Connection) =>
        conn.model('warehouse', WarehouseSchema),
      inject: [MONGODB_CONNECTION],
    },
    DeliveryService,
  ],
})
export class DeliveryModule {}

```

Рисунок 3.16 – Модуль платформи Nest.js

Окрім звичайних модулів існує головний модуль, який об'єднує у собі всі інші та використовуючи який створюється та запускається застосунок. Головний модуль можна побачити на рисунку 3.17, а код, яким запускається застосунок на рисунку 3.18.

```

@Module({
  imports: [
    ConfigModule.forRoot(config),
    MongooseModule,
    AuthModule,
    StaffModule,
    ClientModule,
    ClothModule,
    ReportModule,
    OrderModule,
    StoreModule,
    DeliveryModule,
  ],
  controllers: [AppController],
})
export class AppModule {}

```

Рисунок 3.17 – Головний модуль

```
async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  const configService = app.get<ConfigService>(ConfigService);

  app.use(cookieParser(configService.getOrThrow('module.app.cookieSecret')));

  app.useGlobalInterceptors(new RequestLoggerInterceptor());

  app.enableCors({
    origin: true,
    credentials: true,
  });

  app.useGlobalFilters(new HttpExceptionsFilter());

  app.useGlobalPipes(
    new ValidationPipe({
      disableErrorMessage: false,
      dismissDefaultMessages: false,
      transform: true,
    }),
  );

  await app.listen(3001);
}
bootstrap();
```

Рисунок 3.18 – Запуск застосунку, використовуючи головний модуль

Платформа має декоратор, яким виділяється клас, який містить ендпоінти та у якому можна зазначати валідацію кожного ендпоінта. Приклад такого класу можна побачити на рисунку 3.19.

```

@Controller('delivery')
export class DeliveryController {
  constructor(private readonly deliveryService: DeliveryService) {}

  @Post()
  async createDelivery(
    @Body() dto: DeliveryCreateRequestDto,
    @UserEmail() userEmail: string,
  ) {
    return this.deliveryService.createDelivery(dto, userEmail);
  }

  @Get()
  async getDelivery(@Query() dto: GetDeliveryRequest) {
    return this.deliveryService.getDelivery(dto);
  }

  @Put()
  async updateDelivery(
    @Body() dto: DeliveryUpdateRequestDto,
    @UserEmail() userEmail: string,
  ): Promise<IDeliverySchema> {
    return this.deliveryService.updateDelivery(dto, userEmail);
  }

  @Delete()
  async deleteDelivery(
    @Query('id') id: number,
    @UserEmail() userEmail: string,
  ): Promise<number> {
    return this.deliveryService.softDeleteDelivery(id, userEmail);
  }
}

```

Рисунок 3.19 – Клас визначення ендпоінтів

3.5.3 Бібліотека class-validator

Бібліотека class-validator використовується для валідації даних з запитів. Ця бібліотека гарно інтегрується з Nest.js та використовується у офіційній документації Nest.js. Для початку інтеграції бібліотеки її необхідно додати до застосунку, як на рисунку 3.20.

```
app.useGlobalPipes(
  new ValidationPipe({
    disableErrorMessage: false,
    dismissDefaultMessages: false,
    transform: true,
  }),
);
```

Рисунок 3.20 – Додавання бібліотеки до застосунку

Після цього бібліотека зможе обробляти данні запитів згідно з правил описаних у кожному класі. Приклад dto класу з правилами можна побачити на рисунку 3.21.

```
export class GetDeliveryRequest {
  @IsNumber()
  @Transform(({ value }) => Number(value) || 10)
  limit: number;

  @IsNumber()
  @Transform(({ value }) => Number(value) || 0)
  page: number;

  @IsOptional()
  @IsEnum(EnumSort)
  order?: EnumSort;

  @IsOptional()
  @IsString()
  orderBy?: string;

  @IsOptional()
  @IsNumber()
  @Transform(({ value }) => Number(value))
  id?: number;

  @IsOptional()
  @IsNumber()
  @Transform(({ value }) => Number(value))
  deliveredTo?: number;

  @IsOptional()
  @IsNumber()
  @Transform(({ value }) => Number(value))
  deliveredFrom?: number;
}
```

Рисунок 3.21 – Приклад класу з правилами

Ця бібліотека часто використовується з бібліотекою class-transformer. Вона використовується для трансформації значення до початку валідації. На прикладі рисунку 3.21 class-transformer використовується для перетворення значення до числового. Також вона використовується для реалізації поглибленої валідації, тобто коли поле є типом іншого класу, який теж містить правила. Приклад такої валідації зображено на рисунку 3.22. На цьому рисунку декоратор Type використовується для визначення типу поля. А ValidateNested означає що бібліотека class-validator має зайти вглиб цього поля та перевіряти його елементи. Параметр each true означає, що елемент є масивом та потрібно робити перевірку для кожного елементу масиву.

```
export class DeliverySizeCount implements IDeliverySizeCount {
  @IsEnum(ClothSizes)
  size: ClothSizes;

  @IsNumber()
  @Transform(({ value }) => Number(value))
  count: number;
}

export class ClothDelivered implements IClothDelivered {
  @IsArray()
  @ArrayMinSize(1)
  @ValidateNested({ each: true })
  @Type(() => DeliverySizeCount)
  sizes: DeliverySizeCount[];

  @IsNumber()
  @Transform(({ value }) => Number(value))
  clothId: number;
}
```

Рисунок 3.22 – Приклад поглибленої валідації

3.5.4 Бібліотека Joi

Бібліотека Joi – це бібліотека для валідації даних. У проєкті ця бібліотека використовується для валідації config файлів. У них валідуються дані, які

вписуються руками у кодї, та данї, які беруться з змінних оточення. Приклад такого файлу на рисунку 3.23.

```
const cfgRaw = {
  port: parseInt(process.env.PORT),
  jwtSecret: process.env.JWT_SECRET,
  cookieSecret: process.env.COOKIE_SECRET,
} as const;

Logger.verbose('Validating ' + path.basename(__filename));
Joi.attempt(
  cfgRaw,
  Joi.object({
    port: Joi.number().required(),
    jwtSecret: Joi.string().required(),
    cookieSecret: Joi.string().required(),
  }).required(),
);

export const config = registerAs('module.app', () => cfgRaw);
```

Рисунок 3.23 – Валідація у файлі конфігурації

3.5.4 Бібліотека Mongoose

У проєкті використовується бібліотека mongoose для забезпечення взаємодії між back-end та MongoDB. Основними функціями бібліотеки є підключення до бази даних, створення схем і спілкування з базою даних для отримання або відправки інформації.

Для підключення до бази даних потрібен лише рядок url у якому вже буде база до якої потрібно підключитися, ім'я користувача та пароль. Підключення бази у проєкт можна побачити на рисунку 3.24.

```

const providers = [
  {
    provide: MONGODB_CONNECTION,
    imports: [ConfigModule],
    inject: [ConfigService],
    useFactory: (cfg: ConfigService): Promise<typeof mongoose> =>
      mongoose.connect(cfg.getOrThrow('module.mongodb.url'), {
        connectTimeoutMS: cfg.getOrThrow<number>(
          'module.mongodb.connectTimeoutMS',
        ),
        dbName: cfg.getOrThrow<string>('module.mongodb.dbName'),
      }),
  },
];

@Global()
@Module({
  imports: [ConfigModule.forFeature(config)],
  providers: [...providers],
  exports: [...providers],
})
export class MongodbModule implements OnModuleDestroy {
  async onModuleDestroy(): Promise<void> {
    Logger.log('Closing MongoDB connection...', MongodbModule.name);
    await mongoose.disconnect();
    Logger.verbose('OK', MongodbModule.name);
  }
}

```

Рисунок 3.24 – Підключення до бази даних

Окрім підключення бази, необхідно вказати яку саме таблицю потрібно використати, у проєкті це виконано у вигляді окремого імпорту у модулі. Підключення виконується за спеціальним константним значенням, яке створюється для кожної схеми окремо. Приклад підключення таблиці на рисунку 3.25.

```

providers: [
  {
    provide: CLIENT_MODEL_SCHEMA,
    useFactory: (conn: Connection) => conn.model('client', ClientSchema),
    inject: [MONGODB_CONNECTION],
  },
  ClientService
]

```

Рисунок 3.25 – Підключення до таблиці

Щоб взаємодіяти з окремими таблицями, потрібно створити схеми для цих таблиць. Хоча база даних не є реляційною базою даних і не має явної структури таблиць, все одно потрібно створити структуру для зв'язку з таблицями. Приклад структури показано на рисунку 3.26.

```
export const CLIENT_MODEL_SCHEMA = Symbol('CLIENT_MODEL_SCHEMA');

export interface IClientSchema extends Document {
  id: number;
  name: string;
  email: string;
  phoneNumber: string;
  size: ClothSizes;
  createdBy: string;
  createdAt: string;
  updatedBy?: string;
  updatedAt?: string;
  deletedBy?: string;
}

export const ClientSchema = new mongoose.Schema<IClientSchema>(
  {
    id: { type: Number, required: true, unique: true },
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    phoneNumber: { type: String, required: true },
    size: { type: String, required: false },
    createdBy: { type: String, required: true },
    createdAt: { type: String, required: true },
    updatedBy: { type: String, required: false },
    updatedAt: { type: String, required: false },
    deletedBy: { type: String, required: false },
  },
  {
    timestamps: false,
  },
);
```

Рисунок 3.26 – Схема таблиці Mongoose

3.5.5 Бібліотека Redux

Бібліотека використовується front-end частиною для створення, збереження та зміни стану застосунку.

Перш ніж почати працювати з бібліотекою, її необхідно підключити та додати усі необхідні редьюсери та доповнення(у цьому випадку `redux-thunk`). Підключення бібліотеки можна побачити на рисунках 3.27 та компонент `Provider` на рисунку 3.28.

```
const reducers = combineReducers({
  staff: staffReducer,
  client: clientReducer,
  cloth: clothReducer,
  order: orderReducer,
  store: storeReducer,
  delivery: deliveryReducer,
  reports: reportsReducer,
  auth: authReducer,
});

const store = createStore(reducers, applyMiddleware(ReduxThunk));

export default store;
```

Рисунок 3.27 – Об'єднання редьюсерів та доповнень

```
);
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <Provider store={store}>
        <AppContainer />
      </Provider>
    </BrowserRouter>
  </React.StrictMode>
);
```

Рисунок 3.28 – Підключення об'єднаних редьюсерів до застосунку

Для керування станом використовується редьюсер, який зберігає поточний стан застосунку та має методи для зміни стану. Приклад редьюсера зображено на рисунку 3.27.

```

const clothReducer = (state = initialState, action: ActionTypes): IClothState => {
  switch (action.type) {
    case GET_CLOTH: {
      return action.data;
    }

    case GET_CLOTH_SIZES: {
      const clothIndex = state.sizesByShop?.findIndex((sizeByShop) => sizeByShop.clothId === action.data.clothId);
      let newSizesByShop = state.sizesByShop || [];

      if (clothIndex && clothIndex !== -1) {
        newSizesByShop = newSizesByShop?.map((sizeByShop) => {
          if (sizeByShop.clothId === action.data.clothId) {
            return action.data;
          }
          return sizeByShop;
        });
      } else {
        newSizesByShop.push(action.data);
      }

      return { ...state, sizesByShop: newSizesByShop };
    }

    case CLEAR_ERROR: {
      if (state.clothError === undefined) {
        return state;
      }
      return { ...state, clothError: undefined };
    }

    case SET_CLOTH_ERROR: {
      return { ...state, clothError: action.data };
    }

    default:
      return state;
  }
};

```

Рисунок 3.27 – Редьюсер

Окрім самого редьюсера у файлі знаходяться типи для дій у редьюсері, їх можна побачити на рисунку 3.28.

```

type GetClothActionType = {
  type: typeof GET_CLOTH;
  data: IClothState
}

type GetClothSizesActionType = {
  type: typeof GET_CLOTH_SIZES;
  data: IClothSizeInShops
}

type SetErrorActionType = {
  type: typeof SET_CLOTH_ERROR;
  data: string;
}

type ClearErrorActionType = {
  type: typeof CLEAR_ERROR;
  data: undefined;
}

```

Рисунок 3.28 – Типи дій у редьюсері

Для того, щоб компонент отримав дані з редьюсера, було створено контейнери, які використовуються як компоненти. Ці контейнери отримують дані та передають їх основним компонентам разом з необхідними методами.

Приклад контейнера можна побачити на рисунку 3.29.

```

const ClothContainer = (props: any) => {
  useEffect(() => {}, []);
  return <Cloth {...props} />;
};

const mapStateToProps = (state: IState): Partial<IClothProps> => {
  return {
    cloth: state.cloth,
  };
};

export default connect(mapStateToProps, {
  getClothThunk,
  updateClothThunk,
  createClothThunk,
  deleteClothThunk,
  getClothSizesThunk,
})(ClothContainer);

```

Рисунок 3.29 – Контейнер компоненти

Проблема бібліотеки Redux полягає в тому, що вона не може обробляти асинхронні операції, тобто всі операції, які не виконуються миттєво, наприклад, запити до бази даних. Щоб отримати цю інформацію в зчитувач і записати її в стан програми, потрібна додаткова бібліотека Redux-thunk. Ця бібліотека дозволяє використовувати асинхронні операції в методах зміни стану програми. Метод дещо змінює зовнішній вигляд, але має той самий функціонал, а саме зміну стану інформації.

Приклад такого методу можна побачити на рисунку 3.30.

```
// thunks
export const getClothThunk = (clothData: IClothGet) => async (dispatch: Dispatch<ActionTypes>) => {
  const clothResp = await ClothApi.getAllCloth(clothData);

  if ('error' in clothResp) {
    dispatch({ type: SET_CLOTH_ERROR, data: clothResp.error });
    return;
  }

  dispatch({ type: GET_CLOTH, data: clothResp });
};

export const getClothSizesThunk = (shopId: number) => async (dispatch: Dispatch<ActionTypes>) => {
  const clothResp = await ClothApi.getClothSizeCountByShop(shopId);

  if ('error' in clothResp) {
    dispatch({ type: SET_CLOTH_ERROR, data: clothResp.error });
    return;
  }

  dispatch({ type: GET_CLOTH_SIZES, data: clothResp });
};
```

Рисунок 3.30 – Приклад методу з Redux-thunk

3.5.6 Бібліотека MUI

Бібліотека MUI надає великий інструментар з готових компонентів у проєкті. Зручні та легкі у використанні компоненти бібліотеки використовувались по всьому проєкту. Перевагою цих компонентів є те, що вони можуть бути персоналізовані, тобто, при необхідності можна створити свій власний, використовуючи один із існуючих компонентів як основу, або змінити один з тих що вже є.

Головні компоненти, які використовувались та їх опис написано у таблиці 3.2.

Таблиця 3.2 – Використані компоненти бібліотеки MUI

Назва	Опис
Paper	Звичайний блок-контейнер, що характеризується тінню, яка змінюється залежно від параметра висоти і візуально піднімає або опускає елемент.
Typography	Отриманий елемент відображення тексту має менший розмір, його можна вирівняти по горизонталі та вертикалі, а також змінити елемент, від якого він походить (h1, h2, span і т.д.).
Grid	Контейнер, який може змінювати свою ширину відповідно до розміру екрану. Екран розділено на 12 блоків, і в деталях компонента ви можете вказати, скільки блоків він займає.
IconButton	Кругла кнопка відображення іконки
Dialog	Компонент, що відображається над іншими для відображення важливої інформації або потреби користувача в дії (рис. 3.31).
Button	Кнопка в різних стилях, в тому числі з заблокованим або неактивним.
TextField	Поле для вставки тексту має різні стилі, може мати кілька рядків, а також існує функція, коли поле займає всю можливу ширину.
Divider	Полоса для візуального відділення одного блока від іншого
Table	Таблиця для відображення різної інформації, може мати сортування(рис. 3.32)
Select	Впливаючий список з опціями, які налаштовуються(рис. 3.33)
Menu	Меню, яке з'являється після натискання на кнопку, має вибір з опцій, які будуть прописані(рис. 3.34)
ThemeProvider	Застосовується для розширення стандартних кольорів.
Drawer	Блок, що з'являється з краю екрана та затемнює інші блоки(рисунок 3.35).
Tooltip	Підказка, що з'являється при наведенні на елемент(рис. 3.36).

Створити працівника

Електронна пошта

Прізвище Ім'я Побатькові

Посада

ID магазину

Зробити адміністратором

ВІДМІНИТИ

СТВОРИТИ

Рисунок 3.31 – Елемент Dialog

Обліковий номер ↑	Електронна пошта	ПІБ	Посада	Чи є адміністратором	ID магазину	Час останнього оновлення	Хто останній раз оновив	СТВОРИТИ	
1	string@s.com	test	string	✓	1	2023-09-21T08:06:08.027Z	string@s.com	...	
2	sheverdinkin@gmail.com	qwewqwewqw	qwewqwewqe	✓	1	2023-09-20T11:16:19.804Z	string@s.com	...	
4	test@test.comn	test	test	✗	123	2023-09-21T08:10:36.612Z	string@s.com	...	
6	tes@q.com	rtet	adsf	✓	12	2023-09-21T08:10:48.840Z	string@s.com	...	
7	qweqwe@qwe.qwe	qwe	qwe	✗	12	2023-09-20T11:44:11.130Z	string@s.com	...	
8	asf@qwr.asf	qwrqwr	2134rwq	✗	12	2023-09-20T11:44:20.722Z	string@s.com	...	
9	sheverdinkin1@gmail.com	qwe	qwe	✗	12	2023-09-20T12:11:14.462Z	string@s.com	...	
10	sheverdinkin2@gmail.com	qwe	qwe	✗	1	2023-09-28T09:06:06.541Z	string@s.com	...	
11	sheverdinkinw@gmail.com	qwe	qwe	✗	2	2023-09-20T12:11:33.676Z	string@s.com	...	
12	sheverdinkin3@gmail.com	atest	123	✗	11	2023-09-28T09:05:53.500Z	string@s.com	...	

Рисунок 3.32 – Елемент Table

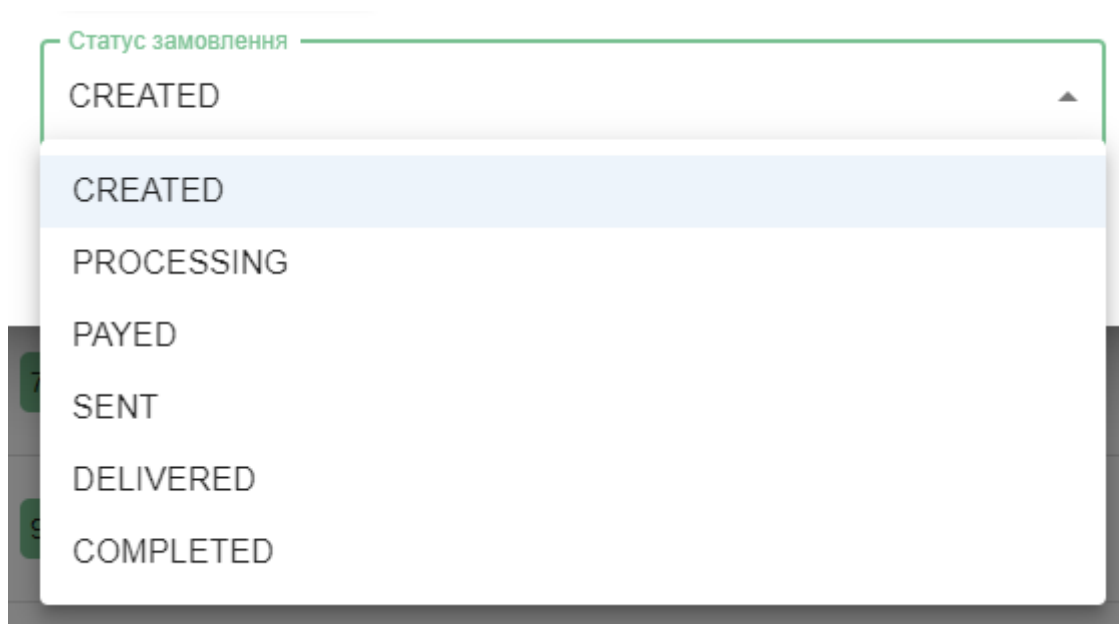


Рисунок 3.33 – Элемент Select

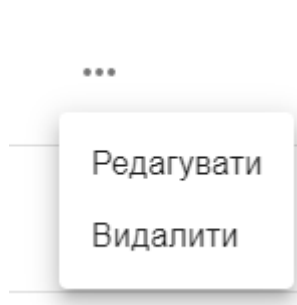


Рисунок 3.34 – Элемент Menu

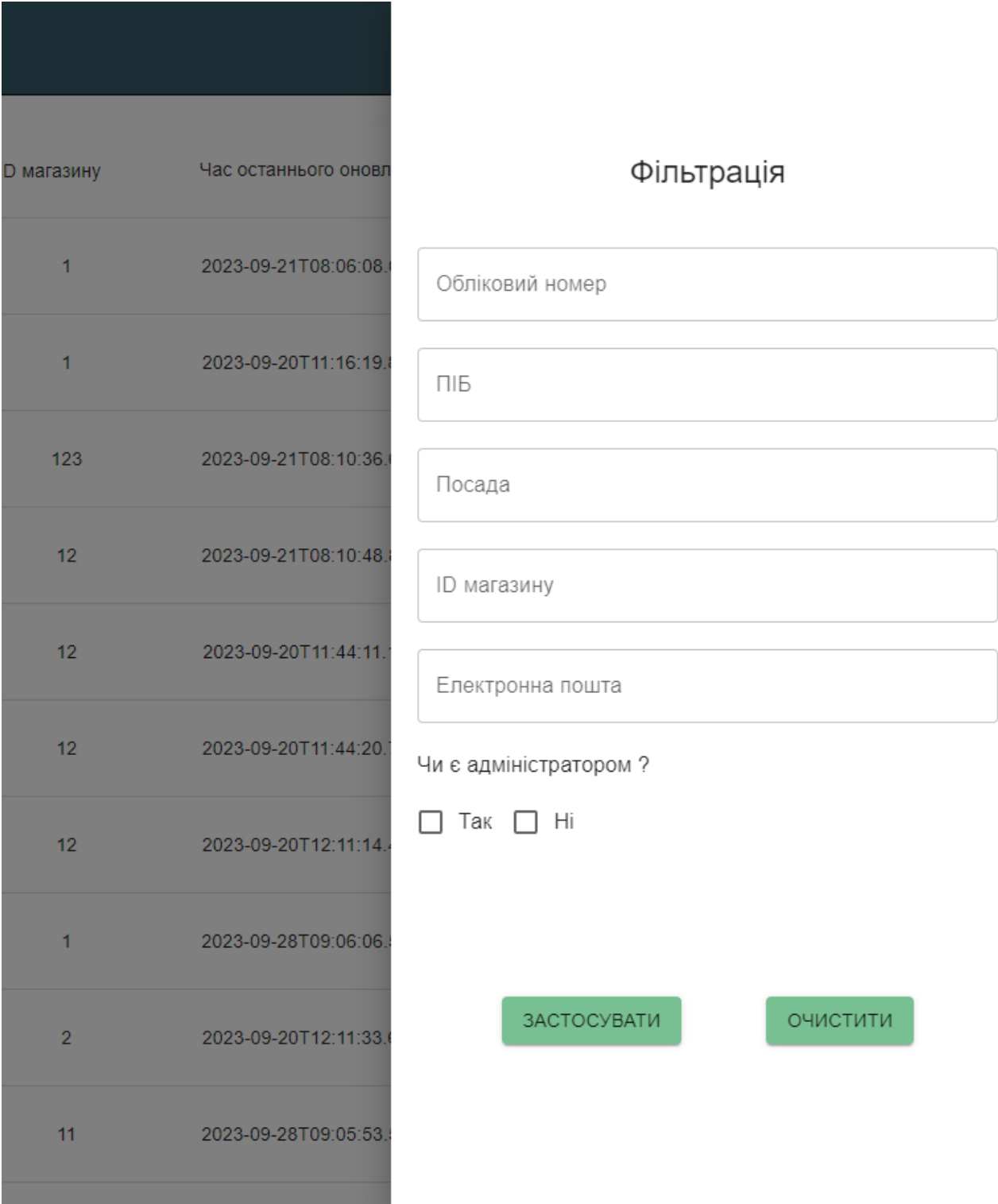


Рисунок 3.35 – Елемент Drawer

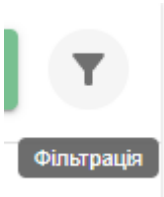


Рисунок 3.36 – Елемент Tooltip

Також окрім компонентів описаних зверху було також використано іншу бібліотеку з її компонентами. Ці компоненти використовуються для перевірки даних під час написання тексту. Окремою перевагою є те, що цей компонент базується на компоненті TextField з бібліотеки MUI, тому він добре вписується в загальний дизайн. Компонент перевіряє поле на основі заданих правил і відображає помилку, якщо валідація не пройдена (рис. 3.37).

Створити працівника

Електронна пошта

Це поле обов'язкове

Прізвище Ім'я Побатькові

Мінімальна дозволена довжина - 2 символи

Посада

Мінімальна дозволена довжина - 2 символи

ID магазину

Це поле обов'язкове

☐ Зробити адміністратором

Рисунок 3.37 – Компонент строки з валідацією

3.6 Використання сервісів контролю версій

У сучасних проєктах, як великих так і не дуже, вже стало необхідністю та загальною звичкою використовувати сервіси контролю версій, такі як GitHub,

GitLab та інші. Такі сервіси мають велику кількість переваг, основні з них це мінімізування ризику втрати коду, у випадку поламки диску, на якому він знаходився, бо після початку використання сервісів контролю версій, користувач переносить свій код у хмарне сховище. Окрім того, такі сервіси зберігають окремо кожну завантажену у них версію коду, тому можна завжди повернутися до попередніх версій, або переглянути зміни які було зроблено під час кожної нової версії. Найпопулярнішими сервісами зараз є GitLab та GitHub. Вони мають деякі відмінності, які не матимуть вплив на розробку проєкту, оскільки впливають на більш продвинуту та масштабну розробку. Тому обрано було той сервіс, який використовувався мною у особистих цілях, а саме GitHub. Було створено два проєкти, які можна побачити на рисунку 3.26.

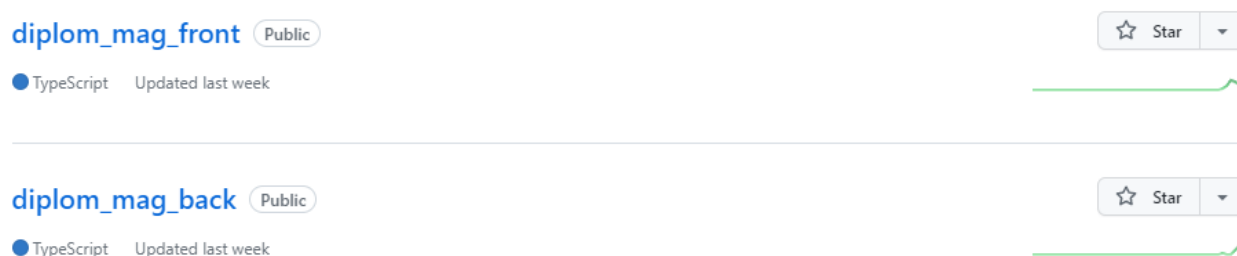


Рисунок 3.26 – Проєкти на GitHub

Впродовж усієї розробки робилися записи змін у ці проєкти. Ці записи можна побачити на рисунках 3.27 та 3.28 для front-end та back-end відповідно. Загалом було зроблено 33 записи для front-end та 21 запис для back-end.

add comments for cookies sheverdinkin committed last week ✓	 a4bf502 <>
change order edit check, fix lint sheverdinkin committed last week ✓	 e03bbf8 <>
Commits on Sep 28, 2023	
add api to store and delivery, add loaders on table and buttons sheverdinkin committed last week	 90564d0 <>
add new order statuses sheverdinkin committed last week	 35461a8 <>
Commits on Sep 26, 2023	
connect order with api sheverdinkin committed last week	 c7d9008 <>
Commits on Sep 23, 2023	
add cloth module, change reducer clear_error actions sheverdinkin committed 2 weeks ago	 8664b14 <>
Commits on Sep 22, 2023	
connect client with api sheverdinkin committed 2 weeks ago	 00860a6 <>
Commits on Sep 21, 2023	
finish adding api to staff, change staff reducer, change filter tab sheverdinkin committed 2 weeks ago	 e264d74 <>
Commits on Sep 20, 2023	
connect with backend api, add auth and partial staff api sheverdinkin committed 2 weeks ago	 7d8ce9f <>
Commits on Sep 19, 2023	
fix lint problems sheverdinkin committed 2 weeks ago	 9ade9b <>
finish auth sheverdinkin committed 2 weeks ago	 1a73300 <>
Commits on Sep 18, 2023	
finish reports, start adding login, add cookie lib sheverdinkin committed 2 weeks ago	 31992ab <>
Commits on Sep 15, 2023	
finish delivery, start reports sheverdinkin committed 3 weeks ago	 62d67d9 <>

Рисунок 3.27 – Записи змін з GitHub для front-end

fixes with auth cookies sheverdinkin committed last week	9f34178	<>
change imports for vercel sheverdinkin committed last week	18a195c	<>
update ts and vercel configs sheverdinkin committed last week ✓	5dcee8c	<>
change imports for vercel sheverdinkin committed last week ✓	b32f884	<>
change imports for vercel sheverdinkin committed last week ✓	c3232ae	<>
change imports for vercel integration sheverdinkin committed last week ✓	4d8a1df	<>
add vercel config sheverdinkin committed last week ✓	861e7cf	<>
fix lint problems sheverdinkin committed last week ✗	bf4313f	<>
add reports to reports module sheverdinkin committed last week	d7d83e5	<>
Commits on Sep 28, 2023		
expand delivery size checking sheverdinkin committed last week	a1f5c7f	<>
expand status changing in order, start add and remove cloth amount on... sheverdinkin committed last week	1ab6889	<>
Commits on Sep 27, 2023		
add delivery module sheverdinkin committed last week	1c02814	<>
Commits on Sep 26, 2023		
add order module, start reports module sheverdinkin committed last week	7415327	<>
Commits on Sep 24, 2023		
add store module sheverdinkin committed 2 weeks ago	cad4133	<>
Commits on Sep 23, 2023		
add cloth module sheverdinkin committed 2 weeks ago	d9f6e11	<>
Commits on Sep 22, 2023		
add client module sheverdinkin committed 2 weeks ago	6b6139f	<>

Рисунок 3.27 – Записи змін з GitHub для back-end

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Опис застосування

Розроблене програмне забезпечення, призначене для керування та відстеження ресурсів, персоналу, продажів та аналізу успішності та популярності різних магазинів та товарів для мережі магазинів одягу.

Сайт виконує декілька задач:

- зберігання та зміна інформації про наявність товару, магазини, персонал, клієнтів та замовлення товарів;
- аналіз популярних товарів та магазинів;

Область використання сайту – це менеджмент мережі магазинів одягу.

Єдиним обмеженням програми є необхідність підключення до Інтернету, оскільки вона є вебсайтом і розміщується на хостингу, для доступу до неї потрібен доступ до Інтернету.

4.2 Інструкція по експлуатації програми

Першим кроком у використанні вебсайту є його налаштування та запуск. Для конфігурації сайту існує файл `.env`, який містить налаштування проєкту у `front-end` та `back-end` частинах. Змінні конфігурації та їхній опис наведено в таблиці 4.1 та 4.2

Таблиця 4.1 – Налаштування `front-end` проєкту та опис змінних

Назва змінної	Опис	Приклад значення
REACT_APP_API_URL	Зберігає адресу, за якою <code>front-end</code> буде звертатися до <code>back-end</code> за даними.	<code>http://localhost:3001/</code>

Таблиця 4.2 – Налаштування back-end проєкту та опис змінних

Назва змінної	Опис	Приклад значення
MONGO_URL	Зберігає url адресу для підключення до бази даних.	http://localhost:3001/
MONGO_DB_NAME	Ім'я бази даних.	cloth_crm
MONGO_CONNECT_TIMEOUT_MS	Кількість мілісекунд які back-end намагатиметься підключитися до бази даних у випадку коли це не вийде зробити одразу.	2000
PORT	Порт, на якому буде запускатися back-end.	3001
JWT_SECRET	Секрет, використовуючи який створюються JWT токени.	PGUAsq7MB9pW
JWT_EXPIRES_IN	Час, за який JWT токен стає протермінованим.	30 days
COOKIE_SECRET	Секрет, яким шифрується та дешифрується куки.	RMgAUqdHVBck

Коли встановлення і заповнення всіх необхідні змінних завершено, треба запустити проєкт. Для цього необхідно запустити команду `npm install` в обох частинах проєкту, а після її завершення запустити `npm run start:dev` на back-end і `npm start` на front-end. Команду `npm install` потрібно виконати лише під час першого запуску, щоб встановити всі необхідні модулі та бібліотеки. У цьому може не бути необхідності, якщо проєкт запускається на хостингу, оскільки хостинг зазвичай встановлює і запускає проєкт сам, але може знадобитися запуск команд для введення їх у відповідні поля на хостингу.

4.3 Взаємодія з БД

Для початку роботи з віддаленою базою MongoDB, потрібно перейти на сайт, де можна отримати доступ до бази даних MongoDB, зареєструватися або увійти до свого облікового запису, а потім вибрати або створити проєкт (рис. 4.1). Далі потрібно обрати кластер для зберігання БД. Від цього залежить ємність і трафік бази даних. У різних регіонах існують платні та безкоштовні

кластери (рисунок 4.2). Після цього можна створювати таблиці в базі даних або переглядати існуючі таблиці.

Після цього взаємодія з базою даних може здійснюватися використовуючи застосунки для перегляду баз даних або back-end.

Create a Project

Name Your Project
Add Members

Name Your Project

Project names have to be unique within the organization (and other restrictions).

test

Cancel
Next

Рисунок 4.1 – Створення проєкту у MongoDB

Create a Shared Cluster

Welcome to MongoDB Atlas! We've recommended some of our most popular options, but feel free to customize your cluster to your needs. For more information, check our [documentation](#).

PREVIEW Serverless
Dedicated
FREE Shared

For learning and exploring MongoDB in a sandbox environment. Basic configuration controls.
No credit card required to start. Upgrade to dedicated clusters for full functionality.
Explore with sample datasets. Limit of one free cluster per project.

Cloud Provider & Region

AWS, Stockholm (eu-north-1)

aws
Google Cloud
Azure

★ Recommended region ⓘ ⓘ Dedicated tier region ⓘ

NORTH AMERICA	EUROPE	AUSTRALIA
Oregon (us-west-2) ★	Stockholm (eu-north-1) ★	Sydney (ap-southeast-2) ★
N. Virginia (us-east-1) ★	Frankfurt (eu-central-1) ★	ASIA
Ohio (us-east-2) ★ ⓘ	Paris (eu-west-3) ★	Singapore (ap-southeast-1) ★

Рисунок 4.2 – Створення кластера у MongoDB

4.4 Хостинг

Викладення сайту на хостинг робить набагато простішим його експлуатацію та перевірку, тому що пропадає необхідність запускати його локально кожного разу. Для викладення на хостинг було обрано сервіс Vercel. Це безкоштовний сервіс, який підтримує більшість популярних фреймворків та досить просто інтегрується у проєкт. Для викладення front-end на хостинг необхідно було лише надати сервісу можливість перегляду git репозиторію з проєктом та вказати, що проєкт розроблено з використанням React, після чого сервіс сам запустив проєкт, та залишилось лише налаштувати змінні оточення. Для викладення на хостинг back-end довелося докласти трохи більше зусиль, бо з якихось незрозумілих причин сервіс не вміє сам запускати back-end застосунки. Для вирішення цієї проблеми довелося додавати бібліотеку vercel у back-end проєкт та викладати його, використовуючи бібліотеку та файл конфігурації. Файл конфігурації можна побачити на рисунку 4.3.

```

{} vercel.json > ...
1  {
2    "version": 2,
3    "builds": [
4      {
5        "src": "src/main.ts",
6        "use": "@vercel/node"
7      }
8    ],
9    "routes": [
10     {
11       "src": "/(.*)",
12       "dest": "src/main.ts",
13       "methods": [
14         "GET",
15         "POST",
16         "PUT",
17         "PATCH",
18         "OPTIONS",
19         "DELETE",
20         "HEAD",
21         "CONNECT",
22         "TRACE"
23       ]
24     }
25   ]
26 }

```

Рисунок 4.3 – Файл конфігурації Vercel

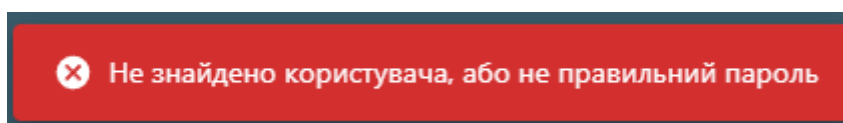
4.5 Помилки

Проект має обробку помилок, які можуть виникнути під час експлуатації, а саме помилки валідації та помилки з back-end. Помилки з back-end відображаються у правому верхньому куті екрану у вигляді сповіщень, а помилки валідації з front-end напряму під полем де ці помилки виникли.

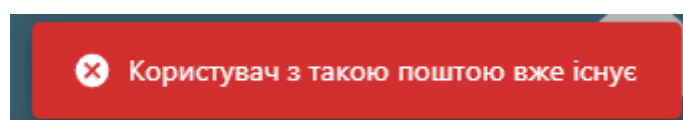
Back-end може повернути помилки під час наступних дій:

- не вдала спроба авторизації (рисунок 4.4);
- спроба створити клієнта або працівника з поштою, яка вже зареєстрована (рисунок 4.5);
- спроба створити одяг з назвою яка вже існує (рисунок 4.6);

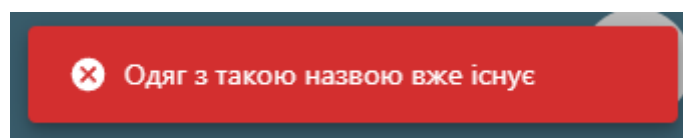
- спроба пошуку сутності з ід, який не існує, або було видалено (рисунки 4.7);
- спроба поставки розміру одягу, який не може бути у конкретного товару (рисунки 4.8);
- спроба видалення поставки, після якої кількість якогось з товарів стане менше 0 (рисунки 4.9);
- спроба створити замовлення з статусом SENT та з кількістю товару більшою ніж є на складі одного магазину (рисунки 4.10);
- спроба зміни замовлення, яке вже відправлене(статус SENT та інші після нього) (рисунки 4.11);
- спроба змінити статус замовлення на RETURNED, якщо замовлення ще не відправлено (рисунки 4.12);
- спроба створити магазин з адресою, яка вже існує (рисунки 4.13).



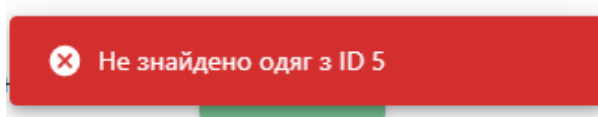
Рисунки 4.4 – Помилка невдалої авторизації



Рисунки 4.5 – Помилка створення користувача з зайнятою поштою



Рисунки 4.6 – Помилка створення одягу з зайнятою назвою



Рисунки 4.7 – Помилка пошуку одягу за ID

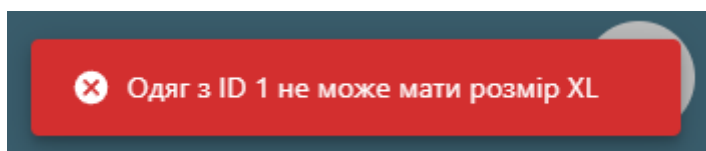


Рисунок 4.8 – Помилка спроби поставки одягу з неіснуючим для нього розміром

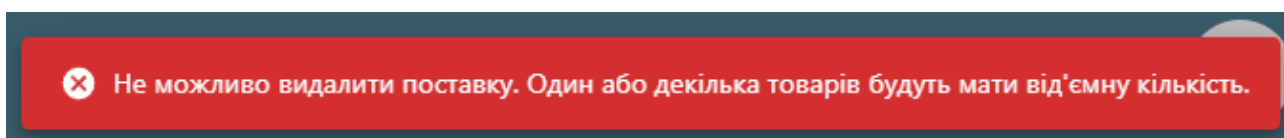


Рисунок 4.9 – Помилка видалення поставки, коли кількість на складі може стати від'ємною

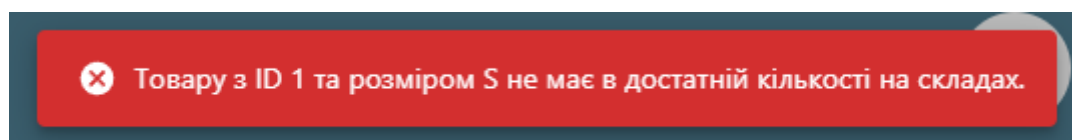


Рисунок 4.10 – Помилка при створенні замовлення з товаром та конкретним розміром, яких на складі менше ніж замовлено

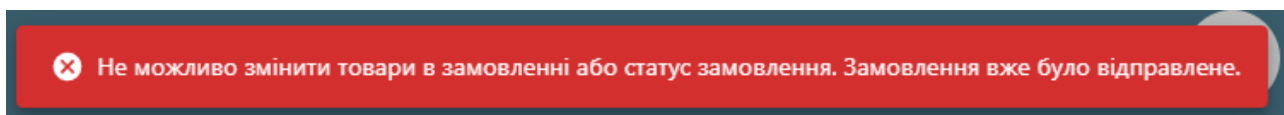


Рисунок 4.11 – Спроба змінити замовлення, яке було відправлене



Рисунок 4.12 – Спроба змінити замовлення на RETURNED, коли воно ще не було відправлене

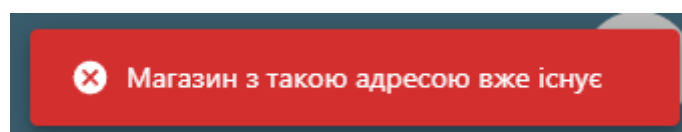


Рисунок 4.13 – Спроба створити магазин за вже існуючою адресою

Помилки описані вище можна побачити під час взаємодії з front-end частиною сайту. Однак існують помилки, які неможливо отримати через інтерфейс, оскільки сам інтерфейс має захист від деяких ситуацій. Одною з таких ситуацій є спроба створити замовлення зі статусом “RETURNED”. Таку дію виконати не можливо ніяк. Однак через інтерфейс користувач навіть не зможе спробувати, але якщо надіслати відповідний запит вручну до back-end частини, то можна отримати помилки зображену на рисунку 4.14.

```
{
  "response": {
    "statusCode": 400,
    "message": "Не можливо створити замовлення з статусом RETURNED",
    "error": "Bad Request"
  },
  "status": 400,
  "options": {},
  "message": "Не можливо створити замовлення з статусом RETURNED",
  "name": "BadRequestException"
}
```

Рисунок 4.14 – Спроба створити замовлення зі статусом RETURNED

5 КЕРІВНИЦТВО ОПЕРАТОРА

5.1 Призначення та умови виконання програми

Розроблене програмне забезпечення, призначене для надання можливості керування та аналізу продажів та персоналу для мережі магазинів одягу.

Для коректного функціонування вебсайту необхідно:

- якщо сайт завантажується на хостинг, знадобиться будь-який пристрій з доступом до Інтернету та один з нових та популярних браузерів;
- для належної роботи вебсайту на персональному комп'ютері (ПК) необхідно мати один з найновіших популярних браузерів, стабільне з'єднання з Інтернетом і комп'ютер, здатний впоратися з навантаженням браузера.

5.2 Виконання програми

Для початку роботи з сайтом треба перейти за посиланням, визначеним на хостингу.

Для прикладу хостинг `vercel` використаний у проєкті автоматично створює ім'я для сайту, використовуючи назву проєкту. Приклад імені для сайту можна побачити на рисунку 5.1.

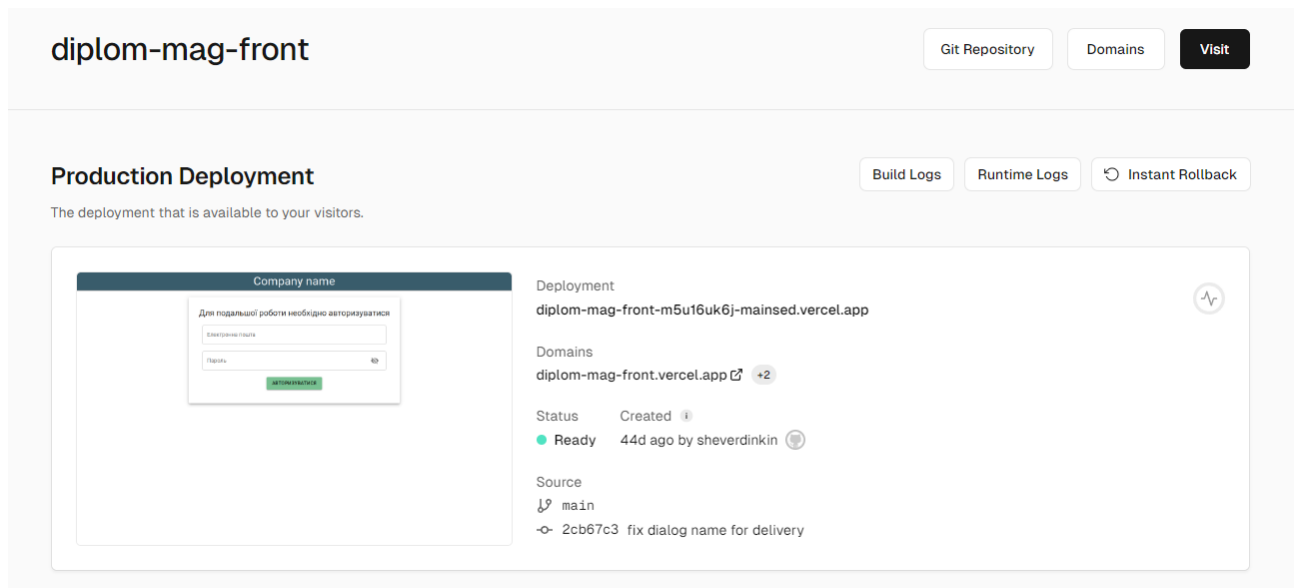


Рисунок 5.1 – Ім'я сайту на хостингу vercel

5.3 Повідомлення оператору

Повідомлення оператору реалізовані у вигляді помилок валідації, сповіщень при помилках з back-end, які було показано у попередньому розділі та повідомлення, які потребують дій користувача, наприклад під час видалення запису, надання або видалення користувачу прав адміністратора тощо (рисунки 5.2. – 5.5).

Створити працівника

Електронна пошта

Це поле обов'язкове

Прізвище Ім'я Побатькові

Мінімальна дозволена довжина - 2 символи

Посада

Мінімальна дозволена довжина - 2 символи

ID магазину

Це поле обов'язкове

☒ Зробити адміністратором

Пароль

Мінімальна довжина паролю - 8 символів, має містити хоча б 1 букву та 1 цифру

ВІДМІНИТИ

СТВОРИТИ

Рисунок 5.2 – Помилка валідації на front-end частині

Видалити працівника "test" ?

ВІДМІНИТИ

ВИДАЛИТИ

Рисунок 5.3 – Повідомлення про видалення

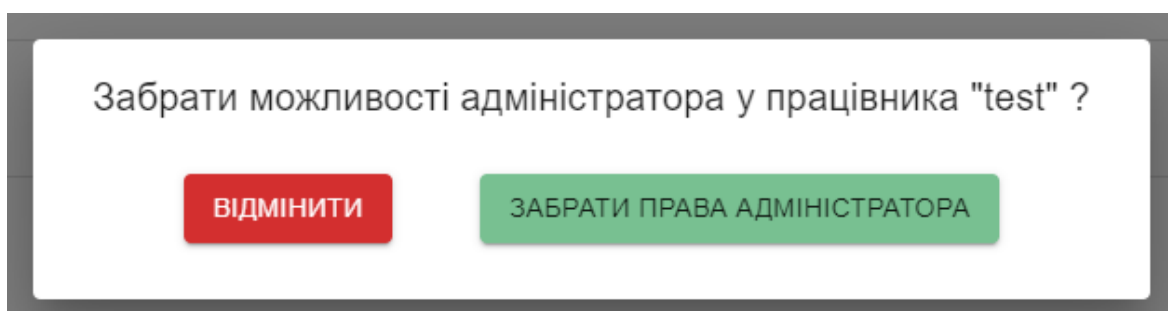


Рисунок 5.4 – Повідомлення про видалення прав адміністратора

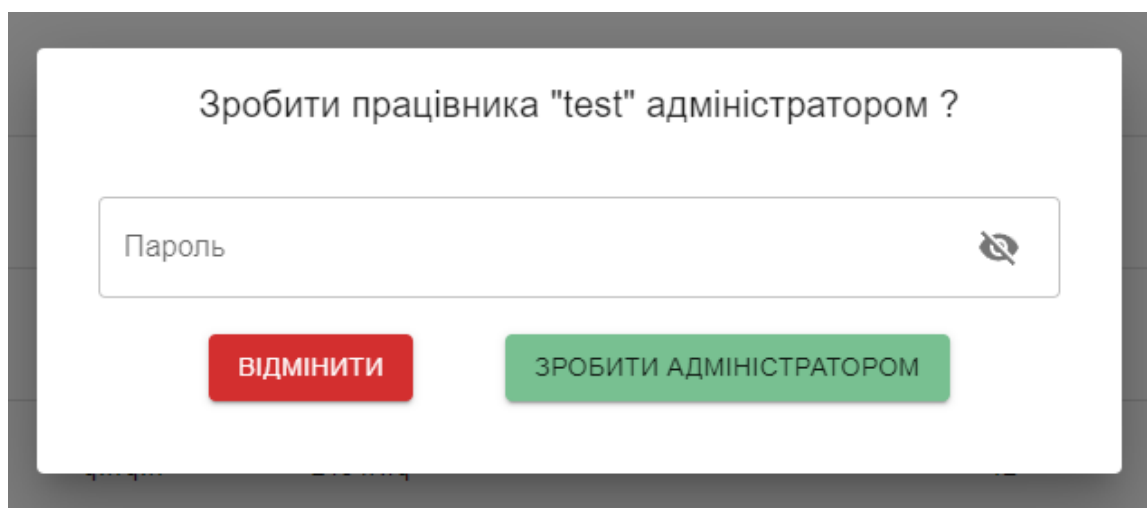


Рисунок 5.5 – Повідомлення про надання прав адміністратора

5.4 Користувачі

На сайті є єдиний тип користувачів, це працівники, які мають можливості адміністратора, у таких користувачів є доступ до наступних сторінок:

- авторизація(рисунок 5.6);
- працівники(рисунок 5.7);
- клієнти(рисунок 5.8);
- товари(рисунок 5.9);
- замовлення(рисунок 5.10);
- магазини(рисунок 5.11);
- поставки(рисунок 5.12);
- звіти(рисунок 5.13).



Company name

Для подальшої роботи необхідно авторизуватися

Електронна пошта

Пароль

АВТОРИЗУВАТИСЯ

Рисунок 5.6 – Сторінка авторизації



Company name

T

Працівники

Клієнти

Товари

Замовлення

Магазини

Поставки

Звіти

Обліковий номер ↑	Електронна пошта	ПІБ	Посада	Чи є адміністратором	ID магазину	Час останнього оновлення	Хто останній раз оновив	СТВОРИТИ
1	string@s.com	test	string	✓	1	2023-09-21T08:06:08.027Z	string@s.com	...
2	sheverdinkin@gmail.com	qwewqewqewq	qwewqewqe	✓	1	2023-09-20T11:16:19.804Z	string@s.com	...
4	test@test.com	test	test	✓	123	2023-10-19T07:40:24.931Z	string@s.com	...
6	tes@q.com	rtet	adsf	✓	12	2023-09-21T08:10:48.840Z	string@s.com	...
7	qewqew@qwe.qwe	qwe	qwe	✗	12	2023-09-20T11:44:11.130Z	string@s.com	...
8	asf@qwr.asf	qwrqwr	2134rvq	✗	12	2023-09-20T11:44:20.722Z	string@s.com	...
9	sheverdinkin1@gmail.com	qwe	qwe	✗	12	2023-09-20T12:11:14.462Z	string@s.com	...
10	sheverdinkin2@gmail.com	qwe	qwe	✗	1	2023-09-28T09:06:06.541Z	string@s.com	...
11	sheverdinkinw@gmail.com	qwe	qwe	✗	2	2023-09-20T12:11:33.676Z	string@s.com	...
12	sheverdinkin3@gmail.com	atest	123	✗	11	2023-09-28T09:05:53.500Z	string@s.com	...

Рисунок 5.7 – Сторінка з працівниками

Company name

T

Працівники

Клієнти

Товари

Замовлення

Магазини

Поставки

Звіти

Обліковий номер ↑	Електронна пошта	ПІБ	Номер телефону	Розмір одягу	Час останнього оновлення	Хто останній раз оновив	СТВОРИТИ
1	sheverdinkin@gmail.com	Ілья TEST	0999843820	S	2023-09-22T07:17:28.798Z	string@s.com	...
3	sheverdinkin12@gmail.com	Ілья	0999843820	XS	2023-09-22T07:17:43.560Z	string@s.com	...
4	sheverdinkin14@gmail.com	Ілья	0999843820	S	2023-09-22T07:17:52.666Z	string@s.com	...
5	sheverdinka@gmail.coma	Ілля Шевердінкін	0966260384	M	2023-09-22T07:18:03.600Z	string@s.com	...
6	sheverdinkin@gmail.comc	Ілля Шевердінкін	0966260384	L	2023-09-22T07:18:12.732Z	string@s.com	...
7	shev11erdinkin@gmail.com	Ілья	0999843820	M	2023-09-22T07:18:20.956Z	string@s.com	...
8	sheverdinkin@gmail.com	Ілля Шевердінкін	0966260384	XS	2023-09-22T07:18:34.170Z	string@s.com	...
9	sheverqwedin@gmail.com	Ілля Шевердінкін	0966260384	L	2023-09-22T07:18:42.085Z	string@s.com	...
10	sheverdizvinkin@gmail.com	Ілья	0999843820	XL	2023-09-22T07:18:48.672Z	string@s.com	...
11	sheverdqrfsainkin@gmail.com	Ілля Шевердінкін	0966260384	L	2023-09-22T07:18:56.926Z	string@s.com	...

Рисунок 5.8 – Сторінка з клієнтами

Company name

T

Працівники

Клієнти

Товари

Замовлення

Магазини

Поставки

Звіти

Обліковий номер ↑	Назва	Опис	Ціна	Доступні розміри	Час останнього оновлення	Хто останній раз оновив	СТВОРИТИ
1	name	desc	1	XS S L	2023-09-22T09:14:18.627Z	string@s.com	...
2	name1	desc	100	XS S L	2023-09-29T09:21:43.758Z	string@s.com	...
3	name3	desc	1	XS S L	2023-09-23T18:48:15.835Z	string@s.com	...
6	name5	desc	1	XS S L	2023-09-22T09:14:31.032Z	string@s.com	...
7	name6	desc	1	XS S L	2023-09-22T09:14:33.397Z	string@s.com	...
9	name8	desc	1	XS S L	2023-09-22T09:14:37.839Z	string@s.com	...
10	name9	desc	1	XS S L	2023-09-22T09:14:40.434Z	string@s.com	...
11	name10	desc	1	XS S L	2023-09-22T09:14:42.761Z	string@s.com	...

Rows per page: 10 1-8 of 8

Рисунок 5.9 – Сторінка з товарами

Company name

T

Працівники

Клієнти

Товари

Замовлення

Магазини

Поставки

Звіти

Обліковий номер ↑	ID клієнта	ID замовлених товарів	Статус замовлення	Ціна замовлення	Час останнього оновлення	Хто останній раз оновив	СТВОРИТИ	
1	1	1	COMPLETED	1	2023-09-28T07:29:40.853Z	string@s.com		...
2	1	2	COMPLETED	2	2023-09-28T07:52:33.577Z	string@s.com		...
3	1	2	SENT	1	2023-09-28T07:23:55.231Z	string@s.com		...
4	1	1	RETURNED	1	2023-09-28T07:41:38.182Z	string@s.com		...
5	1	2	COMPLETED	2	2023-02-28T07:47:05.307Z	string@s.com		...
6	1	2	RETURNED	2	2023-09-28T07:56:18.333Z	string@s.com		...
7	1	3	COMPLETED	2	2023-09-29T09:17:40.713Z	string@s.com		...
8	1	6	COMPLETED	2	2023-09-29T09:18:01.985Z	string@s.com		...
9	1	7	COMPLETED	1	2023-09-29T09:18:25.074Z	string@s.com		...
10	1	9	COMPLETED	1	2023-09-29T09:18:33.128Z	string@s.com		...

Рисунок 5.10 – Сторінка з замовленнями

Company name

T

Працівники

Клієнти

Товари

Замовлення

Магазини

Поставки

Звіти

Обліковий номер ↑	Адреса	Чи працює	Час останнього оновлення	Хто останній раз оновив	СТВОРИТИ	
1	test	✓	2023-09-28T09:02:42.291Z	string@s.com		...
2	test2	✗	2023-09-28T09:02:23.041Z	string@s.com		...
5	teststest	✓	2023-09-28T09:02:48.773Z	string@s.com		...
6	eyueeyue	✗	2023-09-29T09:11:10.160Z	string@s.com		...
7	test1243	✗	2023-09-29T09:15:26.974Z	string@s.com		...
8	qwewqewqe	✗	2023-09-29T09:15:33.845Z	string@s.com		...

Rows per page: 10 1-6 of 6 < >

Рисунок 5.11 – Сторінка з магазинами

Company name

T

Працівники

Клієнти

Товари

Замовлення

Магазини

Поставки

Звіти

	Обліковий номер ↑	Доставка з	Доставка в	Всього товару доставлено	Ціна	Час останнього оновлення	Хто останній раз оновив	СТВОРИТИ	
▼	1		1	15 шт.	100	2023-09-27T09:55:19.375Z	string@s.com		...
▼	2		1	15 шт.	100	2023-09-27T09:56:31.150Z	string@s.com		...
▼	3		2	15 шт.	100	2023-09-27T09:56:44.865Z	string@s.com		...
▼	4		2	15 шт.	100	2023-09-27T09:58:31.671Z	string@s.com		...
▼	5		2	15 шт.	100	2023-09-27T09:59:15.138Z	string@s.com		...
▼	6		2	15 шт.	100	2023-09-27T09:59:52.178Z	string@s.com		...
▼	7		2	15 шт.	100	2023-09-27T10:00:32.702Z	string@s.com		...
▼	8		2	15 шт.	100	2023-09-27T10:02:11.027Z	string@s.com		...
▼	9		2	15 шт.	100	2023-09-27T10:02:51.046Z	string@s.com		...
▼	10		2	15 шт.	100	2023-09-27T10:03:34.217Z	string@s.com		...

Рисунок 5.12 – Сторінка з поставками

Company name

T

Працівники

Клієнти

Товари

Замовлення

Магазини

Поставки

Звіти

Топ 5 магазинів за продажами за пів року

Номер магазину	Кількість продажів	Зміна продажів
2	11 шт.	▲ 9 шт.
5	10 шт.	▼ 0 шт.
6	7 шт.	▼ 0 шт.
1	6 шт.	▼ 0 шт.
7	2 шт.	▼ 0 шт.

Топ 5 товарів за продажами за пів року

Номер товару	Кількість продажів	Зміна продажів
3	10 шт.	▼ 0 шт.
2	8 шт.	▲ 6 шт.
6	7 шт.	▼ 0 шт.
7	6 шт.	▼ 0 шт.
1	3 шт.	▼ 0 шт.

Топ 5 розмірів за популярністю за пів року

Розмір	Кількість продажів	Зміна продажів
S	19 шт.	▲ 17 шт.
XS	19 шт.	▼ 0 шт.

Кількість нових та звільнених працівників за пів року

Місяць	Найнято	Звільнено
Травень	0 чол.	0 чол.
Червень	0 чол.	0 чол.
Липень	0 чол.	0 чол.
Серпень	0 чол.	0 чол.
Вересень	13 чол.	3 чол.
Жовтень	0 чол.	0 чол.

Дохід за останні пів року

Місяць	Дохід
Травень	0 грн.
Червень	0 грн.
Липень	0 грн.
Серпень	2 грн.
Вересень	528 грн.
Жовтень	0 грн.

Рисунок 5.13 – Сторінка з звітами

На усіх сторінках окрім звітів та працівників кожний запис в таблиці має додаткові дії, а саме видалення та редагування (рисунок 5.14). Працівники мають додаткові дії, а саме надання чи видалення прав адміністратора у користувача (рисунок 5.15).

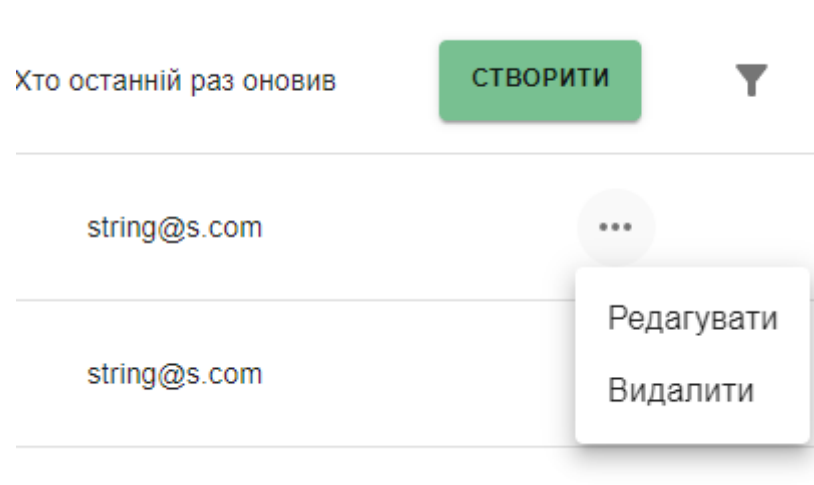


Рисунок 5.14 – Додаткові дії у таблицях

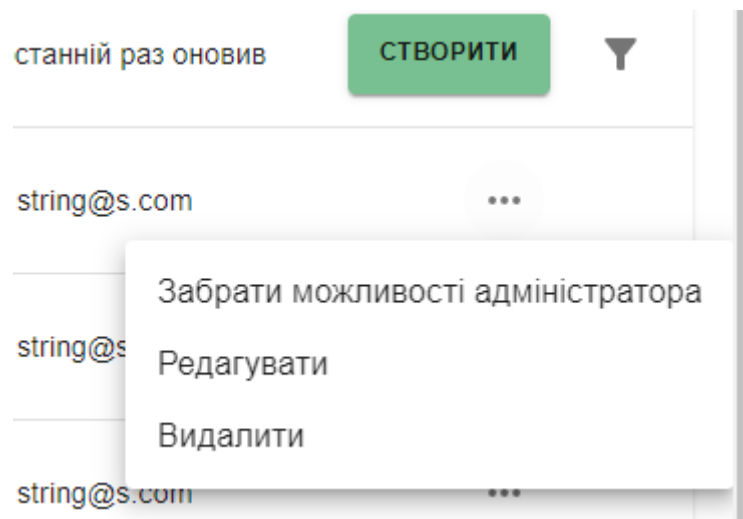


Рисунок 5.15 – Додаткові дії у таблиця працівників

ВИСНОВКИ

У дипломній роботі було проаналізовано предмет і тему дослідження, а саме системи управління мережевими магазинами та розробку такої системи. У ході виконання роботи було проаналізовано існуючі рішення, такі як ClickUp, Zendesk та Microsoft Dynamics, описано їх переваги та недоліки, а також показано, чого їм не вистачає для виконання задачі проєкту. Було проаналізовано поширені мови програмування для веброзробки та обрано TypeScript як мову розробки проєкту. Також було проаналізовано різні системи управління базами даних і обрано MongoDB, як найбільш підходящу. Також було описано бібліотеки, платформи та фреймворки, що використовувалися при розробці, їх застосування в проєкті та приклади їх використання. Були надані схеми бази даних з назвами полів, описами полів та необхідністю їх заповнення. Описано структуру проєкту з поясненням елементів структури, її змісту та призначення, а також алгоритми, використані при розробці. Також було описано спосіб взаємодії програміста з базою даних, тобто створення та використання бази даних. Крім того, було названо користувача та визначено сторінки, до яких він має доступ. У кінці, були відображені помилки, повідомлення та зовнішній вигляд сторінок проєкту.

Кінцевим результатом проєкту стало створення вебсайту для управління мережі магазинів одягу з можливістю отримання звітів діяльності.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. WebSite ClickUp [Electronic resource]. – Access mode :
<https://clickup.com/>
2. WebSite Zendesk [Electronic resource]. – Access mode:
<https://www.zendesk.com/>
3. WebSite Microsoft Dynamics [Electronic resource]. – Access mode:
<https://dynamics.microsoft.com/>
4. JS [Electronic resource]. – Access mode:
<https://developer.mozilla.org/en-US/docs/Web/JavaScript>
5. TS [Electronic resource]. – Access mode: <https://www.typescriptlang.org/>
6. PHP [Electronic resource]. – Access mode: <https://www.php.net/>
7. React [Electronic resource]. – Access mode: <https://reactjs.org/>
8. Vue [Electronic resource]. – Access mode: <https://vuejs.org/>
9. Angular [Electronic resource]. – Access mode: <https://angular.io/>
10. Express [Electronic resource]. – Access mode: <https://expressjs.com/>
11. Nest.js [Electronic resource]. – Access mode: <https://expressjs.com/>
12. Redux [Electronic resource]. – Access mode: <https://redux.js.org/>
13. MUI [Electronic resource]. – Access mode: <https://mui.com/>
14. Notistack [Electronic resource]. – Access mode: <https://notistack.com/>
15. Joi [Electronic resource]. – Access mode: <https://joi.dev/>
16. Mongoose [Electronic resource]. – Access mode:
<https://mongoosejs.com/>
17. Class validator [Electronic resource]. – Access mode:
<https://www.npmjs.com/package/@nestjs/class-validator/v/0.13.1>
18. MongoDB [Electronic resource]. – Access mode:
<https://cloud.mongodb.com/>
19. MySql [Electronic resource]. – Access mode: <https://www.mysql.com/>
20. PostgreSQL [Electronic resource]. – Access mode:
<https://www.postgresql.org/>

21. Vercel [Electronic resource]. – Access mode: <https://vercel.com/>
22. 10 Best CRM [Electronic resource]. – Access mode: <https://clickup.com/blog/crm-software-examples/#10-5-microsoft-dynamics>
23. WebStorm [Electronic resource]. – Access mode: <https://www.jetbrains.com/webstorm/>

ДОДАТОК А
Текст програми

Front-end

auth.api.ts

```
import { AxiosError } from 'axios';
import {
  IAuth,
  IAuthState,
  IAuthorize,
  ILogout,
} from '../Redux/interfaces';
import { instance } from './axios.instance';
import Cookies from 'js-cookie';

export const AuthApi = {
  authorize: async (authData: IAuthorize): Promise<IAuthState> => {
    try {
      const auth = (await instance.post<IAuth>('auth/login', authData)).data;
      if (auth.isAuthenticated && auth.name && auth.accessToken) {
        Cookies.set('isAuthorized', auth.isAuthenticated.toString());
        Cookies.set('userName', auth.name);
      }

      return {
        auth,
      };
    } catch (e) {
      const error = e as AxiosError<any>;
      return {
        auth: {
          isAuthenticated: false,
          error: error.response?.data?.message,
        },
      };
    }
  },
  logout: async (): Promise<ILogout> => {
    try {
      await instance.post<ILogout>('auth/logout');
      Cookies.remove('isAuthorized');
      Cookies.remove('userName');
      return {};
    } catch (e) {
      const error = e as AxiosError<any>;
      return {
        error: error.response?.data?.message,
      };
    }
  },
};
```

axios.instance.ts

```
import axios from 'axios';

export const instance = axios.create({
  withCredentials: true,
  baseURL: process.env.REACT_APP_API_URL,
});
```

client.api.ts

```
import { AxiosError } from 'axios';
import {
  IClient,
  IClientCreate,
  IClientDelete,
  IClientError,
  IClientGet,
  IClientState,
  IClientUpdate,
} from '../Redux/interfaces';
import { instance } from './axios.instance';

export const ClientApi = {
  getAllClient: async (clientData: IClientGet): Promise<IClientState | IClientError> => {
```

```

    if (clientData === undefined) {
      return [] as any;
    }

    try {
      const { filter, limit, page, sort } = clientData;
      return (
        await instance.get<IClientState>(
          'client', {
            params: {
              limit,
              page,
              order: sort?.order,
              orderBy: sort?.orderBy,
              ...filter,
            },
          }
        )
      ).data;
    } catch (e) {
      const error = e as AxiosError<any>;
      return { error: error.response?.data?.message };
    }
  },

  createClient: async (clientData: IClientCreate): Promise<IClient | IClientError> => {
    try {
      return (await instance.post('client', clientData)).data;
    } catch (e) {
      const error = e as AxiosError<any>;
      return { error: error.response?.data?.response?.message?.toString() || error.response?.data?.message };
    }
  },

  updateClient: async (clientData: IClientUpdate): Promise<IClient | IClientError> => {
    try {
      return (await instance.put('client', clientData)).data;
    } catch (e) {
      const error = e as AxiosError<any>;
      return { error: error.response?.data?.response?.message?.toString() || error.response?.data?.message };
    }
  },

  deleteClient: async (clientData: IClientDelete): Promise<number | IClientError> => {
    try {
      return (await instance.delete('client', { params: clientData })).data;
    } catch (e) {
      const error = e as AxiosError<any>;
      return { error: error.response?.data?.response?.message?.toString() || error.response?.data?.message };
    }
  },
};

```

cloth.api.ts

```

import { AxiosError } from 'axios';
import {
  ICloth,
  IClothCreate,
  IClothDelete,
  IClothError,
  IClothGet,
  IClothSizeInShops,
  IClothState,
  IClothUpdate,
} from '../Redux/interfaces';
import { instance } from './axios.instance';

export const ClothApi = {
  getAllCloth: async (clothData: IClothGet): Promise<IClothState | IClothError> => {
    if (clothData === undefined) {
      return [] as any;
    }

    try {
      const { filter, limit, page, sort } = clothData;

```

```

    return (
      await instance.get<IClothState>(
        'cloth', {
          params: {
            limit,
            page,
            order: sort?.order,
            orderBy: sort?.orderBy,
            ...filter,
          },
        }
      )
    ).data;
  } catch (e) {
    const error = e as AxiosError<any>;
    return { error: error.response?.data?.message };
  }
},

getClothSizeCountByShop: async (clothId: number): Promise<IClothSizeInShops | IClothError> => {
  try {
    return (await instance.get(`cloth/${clothId}`)).data;
  } catch (e) {
    const error = e as AxiosError<any>;
    return { error: error.response?.data?.response?.message?.toString() || error.response?.data?.message };
  }
},

createCloth: async (clothData: IClothCreate): Promise<ICloth | IClothError> => {
  try {
    return (await instance.post('cloth', clothData)).data;
  } catch (e) {
    const error = e as AxiosError<any>;
    return { error: error.response?.data?.response?.message?.toString() || error.response?.data?.message };
  }
},

updateCloth: async (clothData: IClothUpdate): Promise<ICloth | IClothError> => {
  try {
    return (await instance.put('cloth', clothData)).data;
  } catch (e) {
    const error = e as AxiosError<any>;
    return { error: error.response?.data?.response?.message?.toString() || error.response?.data?.message };
  }
},

deleteCloth: async (clothData: IClothDelete): Promise<number | IClothError> => {
  try {
    return (await instance.delete('cloth', { params: clothData })).data;
  } catch (e) {
    const error = e as AxiosError<any>;
    return { error: error.response?.data?.response?.message?.toString() || error.response?.data?.message };
  }
},
};

```

delivery.api.ts

```

import { AxiosError } from 'axios';
import {
  IDelivery,
  IDeliveryCreate,
  IDeliveryDelete,
  IDeliveryError,
  IDeliveryGet,
  IDeliveryState,
  IDeliveryUpdate,
} from '../Redux/interfaces';
import { instance } from './axios.instance';

export const DeliveryApi = {
  getAllDelivery: async (deliveryData: IDeliveryGet): Promise<IDeliveryState | IDeliveryError> => {
    if (deliveryData === undefined) {
      return [] as any;
    }
  }
}

```

```

    try {
      const { filter, limit, page, sort } = deliveryData;
      return (
        await instance.get<IDeliveryState>(
          'delivery', {
            params: {
              limit,
              page,
              order: sort?.order,
              orderBy: sort?.orderBy,
              ...filter,
            },
          }
        )
      ).data;
    } catch (e) {
      const error = e as AxiosError<any>;
      return { error: error.response?.data?.message };
    }
  },

  createDelivery: async (deliveryData: IDeliveryCreate): Promise<IDelivery | IDeliveryError> => {
    try {
      return (await instance.post('delivery', deliveryData)).data;
    } catch (e) {
      const error = e as AxiosError<any>;
      return { error: error.response?.data?.response?.message?.toString() || error.response?.data?.message };
    }
  },

  updateDelivery: async (deliveryData: IDeliveryUpdate): Promise<IDelivery | IDeliveryError> => {
    try {
      return (await instance.put('delivery', deliveryData)).data;
    } catch (e) {
      const error = e as AxiosError<any>;
      return { error: error.response?.data?.response?.message?.toString() || error.response?.data?.message };
    }
  },

  deleteDelivery: async (deliveryData: IDeliveryDelete): Promise<number | IDeliveryError> => {
    try {
      return (await instance.delete('delivery', { params: deliveryData })).data;
    } catch (e) {
      const error = e as AxiosError<any>;
      return { error: error.response?.data?.response?.message?.toString() || error.response?.data?.message };
    }
  },
};

```

order.api.ts

```

import { AxiosError } from 'axios';
import {
  IOrder,
  IOrderCreate,
  IOrderDelete,
  IOrderError,
  IOrderGet,
  IOrderState,
  IOrderUpdate,
} from '../Redux/interfaces';
import { instance } from './axios.instance';

export const OrderApi = {
  getAllOrder: async (orderData: IOrderGet): Promise<IOrderState | IOrderError> => {
    if (orderData === undefined) {
      return [] as any;
    }

    try {
      const { filter, limit, page, sort } = orderData;
      return (
        await instance.get<IOrderState>(
          'order', {
            params: {
              limit: limit,

```

```

        page: page,
        order: sort?.order,
        orderBy: sort?.orderBy,
        ...filter,
      },
    )
  ).data;
} catch (e) {
  const error = e as AxiosError<any>;
  return { error: error.response?.data?.message };
}
},

createOrder: async (orderData: IOrderCreate): Promise<IOrder | IOrderError> => {
  try {
    return (await instance.post('order', orderData)).data;
  } catch (e) {
    const error = e as AxiosError<any>;
    return { error: error.response?.data?.response?.message?.toString() || error.response?.data?.message };
  }
},

updateOrder: async (orderData: IOrderUpdate): Promise<IOrder | IOrderError> => {
  try {
    return (await instance.put('order', orderData)).data;
  } catch (e) {
    const error = e as AxiosError<any>;
    return { error: error.response?.data?.response?.message?.toString() || error.response?.data?.message };
  }
},

deleteOrder: async (orderData: IOrderDelete): Promise<number | IOrderError> => {
  try {
    return (await instance.delete('order', { params: orderData })).data;
  } catch (e) {
    const error = e as AxiosError<any>;
    return { error: error.response?.data?.response?.message?.toString() || error.response?.data?.message };
  }
},
};

```

reports.api.ts

```

import { AxiosError } from 'axios';
import {
  IReportsError,
  IReportsState,
} from '../Redux/interfaces';
import { instance } from './axios.instance';

export const ReportsApi = {
  getReports: async (): Promise<IReportsState | IReportsError> => {
    try {
      return { reports: (await instance.get('report')).data };
    } catch (e) {
      const error = e as AxiosError<any>;
      return { error: error.response?.data?.response?.message?.toString() || error.response?.data?.message };
    }
  },
};

```

staff.api.ts

```

import { instance } from './axios.instance';
import {
  IStaff,
  IStaffCreate,
  IStaffDelete,
  IStaffError,
  IStaffGet,
  IStaffState,
  IStaffUpdate,
} from '../Redux/interfaces';
import { AxiosError } from 'axios';

```

```

export const StaffApi = {
  getAllStaff: async (staffData: IStaffGet): Promise<IStaffState | IStaffError> => {
    if (staffData === undefined) {
      return [] as any;
    }

    try {
      const { filter, limit, page, sort } = staffData;
      return (
        await instance.get<IStaffState>(
          'staff', {
            params: {
              limit: limit,
              page: page,
              order: sort?.order,
              orderBy: sort?.orderBy,
              ...filter,
            },
          }
        )
      ).data;
    } catch (e) {
      const error = e as AxiosError<any>;
      return { error: error.response?.data?.message };
    }
  },

  createStaff: async (staffData: IStaffCreate): Promise<IStaff | IStaffError> => {
    try {
      return (await instance.post('staff', staffData)).data;
    } catch (e) {
      const error = e as AxiosError<any>;
      return { error: error.response?.data?.response?.message?.toString() || error.response?.data?.message };
    }
  },

  updateStaff: async (staffData: IStaffUpdate): Promise<IStaff | IStaffError> => {
    try {
      return (await instance.put('staff', staffData)).data;
    } catch (e) {
      const error = e as AxiosError<any>;
      return { error: error.response?.data?.response?.message?.toString() || error.response?.data?.message };
    }
  },

  deleteStaff: async (staffData: IStaffDelete): Promise<number | IStaffError> => {
    try {
      return (await instance.delete('staff', { params: staffData })).data;
    } catch (e) {
      const error = e as AxiosError<any>;
      return { error: error.response?.data?.response?.message?.toString() || error.response?.data?.message };
    }
  },
};

```

store.api.ts

```

import { instance } from './axios.instance';
import {
  IStore,
  IStoreCreate,
  IStoreDelete,
  IStoreError,
  IStoreGet,
  IStoreState,
  IStoreUpdate,
} from '../Redux/interfaces';
import { AxiosError } from 'axios';

export const StoreApi = {
  getAllStore: async (storeData: IStoreGet): Promise<IStoreState | IStoreError> => {
    if (storeData === undefined) {
      return [] as any;
    }
  }
}

```

```

    try {
      const { filter, limit, page, sort } = storeData;
      return (
        await instance.get<IStoreState>(
          'store', {
            params: {
              limit: limit,
              page: page,
              order: sort?.order,
              orderBy: sort?.orderBy,
              ...filter,
            },
          }
        )
      ).data;
    } catch (e) {
      const error = e as AxiosError<any>;
      return { error: error.response?.data?.message };
    }
  },

  createStore: async (storeData: IStoreCreate): Promise<IStore | IStoreError> => {
    try {
      return (await instance.post('store', storeData)).data;
    } catch (e) {
      const error = e as AxiosError<any>;
      return { error: error.response?.data?.response?.message?.toString() || error.response?.data?.message };
    }
  },

  updateStore: async (storeData: IStoreUpdate): Promise<IStore | IStoreError> => {
    try {
      return (await instance.put('store', storeData)).data;
    } catch (e) {
      const error = e as AxiosError<any>;
      return { error: error.response?.data?.response?.message?.toString() || error.response?.data?.message };
    }
  },

  deleteStore: async (storeData: IStoreDelete): Promise<number | IStoreError> => {
    try {
      return (await instance.delete('store', { params: storeData })).data;
    } catch (e) {
      const error = e as AxiosError<any>;
      return { error: error.response?.data?.response?.message?.toString() || error.response?.data?.message };
    }
  },
};

```

app.interface.ts

```

import { IAuth } from './auth.interface';

export interface IAppProps {
  auth: IAuth;
  errors: string[];
  setAuthorized(isAuthorized: boolean, userName?: string): void;
  clearError(): void;
}

```

auth.interface.ts

```

export interface IAuth {
  isAuthorized: boolean;
  name?: string;
  error?: string;
  accessToken?: string;
}

export interface ILogout {
  error?: string;
}

export interface IAuthorize {
  email: string;
  password: string;
}

```

```

}

export interface IAuthState {
  auth: IAuth;
}

export interface IAuthProps {
  auth: IAuthState;
  authorizeThunk(authData: IAuthorize): void;
}

```

client.interface.ts

```

import { IGetSort } from '../utils/interfaces/get.sort.interface';

export enum ClothSizes {
  XS = 'XS',
  S = 'S',
  M = 'M',
  L = 'L',
  XL = 'XL',
  XXL = 'XXL',
}

export interface IClient {
  id: number;
  name: string;
  email: string;
  phoneNumber: string;
  size: ClothSizes;
  createdBy: string;
  createdAt: string;
  updatedBy: string;
  updatedAt: string;
  deletedBy: string;
}

export interface IClientState {
  client: IClient[];
  clientCount: number;
  clientError?: string;
}

export interface IClientError {
  error: string;
}

export interface IClientProps {
  client: IClientState;
  getClientThunk(clientGetData?: IClientGet): void;
  updateClientThunk(clientToUpdate: IClientUpdate): void;
  createClientThunk(clientToCreate: IClientCreate): void;
  deleteClientThunk(clientToDelete: IClientDelete): void;
}

export interface IClientCreate {
  name: string;
  email: string;
  phoneNumber: string;
  size: ClothSizes;
}

export interface IClientUpdate {
  id: number;
  name?: string;
  email?: string;
  phoneNumber?: string;
  size?: ClothSizes;
}

export interface IClientDelete {
  id: number;
}

export interface IClientGetFilter {
  id?: number;
}

```

```

    name?: string;
    email?: string;
    phoneNumber?: string;
    size?: ClothSizes;
  }

  export interface IClientGet {
    limit?: number;
    page?: number;
    filter?: IClientGetFilter;
    sort?: IGetSort
  }

```

cloth.interface.ts

```

import { IGetSort } from '../utils/interfaces/get.sort.interface';
import { ClothSizes } from './client.interface';

interface IClothSizeCount {
  size: ClothSizes;
  count: number;
}

export interface IClothSizeInShop {
  sizes: IClothSizeCount[];
  shopId: number;
}

export interface IClothSizeInShops {
  shops: IClothSizeInShop[];
  clothId: number;
}

export interface IClothError {
  error: string;
}

export interface ICloth {
  id: number;
  name: string;
  desc: string;
  price: number;
  availableSizes: ClothSizes[];
  createdBy: string;
  createdAt: string;
  updatedBy: string;
  updatedAt: string;
  deletedBy: string;
}

export interface IClothState {
  cloth: ICloth[];
  clothCount: number;
  sizesByShop?: IClothSizeInShops[];
  clothError?: string;
}

export interface IClothProps {
  cloth: IClothState;
  getClothThunk(clothGetData?: IClothGet): void;
  updateClothThunk(clothToUpdate: IClothUpdate): void;
  createClothThunk(clothToCreate: IClothCreate): void;
  deleteClothThunk(clothToDelete: IClothDelete): void;
  getClothSizesThunk(clothId: number): void
}

export interface IClothCreate {
  desc: string;
  name: string;
  price: number;
  availableSizes: ClothSizes[];
}

export interface IClothUpdate {
  id: number;
  name?: string;

```

```

    desc?: string;
    price?: number;
    availableSizes?: ClothSizes[];
}

```

```

export interface IClothDelete {
  id: number;
}

```

```

export interface IClothGetFilter {
  id?: number;
  name?: string;
  desc?: string;
  price?: number;
  availableSizes?: ClothSizes[];
}

```

```

export interface IClothGet {
  limit?: number;
  page?: number;
  filter?: IClothGetFilter;
  sort?: IGetSort;
}

```

delivery.interface.ts

```

import { IGetSort } from '../utils/interfaces/get.sort.interface';
import { ClothSizes } from './client.interface';

```

```

export enum DeliveryType {
  INTERNAL = 'INTERNAL',
  EXTERNAL = 'EXTERNAL'
}

```

```

export interface IDeliverySizeCount {
  size: ClothSizes;
  count: number;
}

```

```

export interface IClothDelivered {
  sizes: IDeliverySizeCount[];
  clothId: number;
}

```

```

export interface IDelivery {
  id: number;
  deliveredTo: number;
  deliveredFrom?: number;
  totalAmountDelivered: number;
  typeOfDelivery: DeliveryType;
  price?: number;
  clothDelivered: IClothDelivered[];
  createdBy: string;
  createdAt: string;
  updatedBy: string;
  updatedAt: string;
  deletedBy: string;
}

```

```

export interface IDeliveryState {
  delivery: IDelivery[];
  deliveryCount: number;
  deliveryError?: string;
}

```

```

export interface IDeliveryError {
  error: string;
}

```

```

export interface IDeliveryProps {
  delivery: IDeliveryState;
  getDeliveryThunk(deliveryGetData?: IDeliveryGet): void;
  updateDeliveryThunk(deliveryToUpdate: IDeliveryUpdate): void;
  createDeliveryThunk(deliveryToCreate: IDeliveryCreate): void;
  deleteDeliveryThunk(deliveryToDelete: IDeliveryDelete): void;
  getDeliverySizesThunk(deliveryId: number): void
}

```

```

}

export interface IDeliveryCreate {
  deliveredTo: number;
  deliveredFrom?: number;
  typeOfDelivery: DeliveryType;
  price?: number;
  clothDelivered: IClothDelivered[];
}

export interface IDeliveryUpdate {
  id: number;
  deliveredTo?: number;
  deliveredFrom?: number;
  price?: number;
  clothDelivered?: IClothDelivered[];
}

export interface IDeliveryDelete {
  id: number;
}

export interface IDeliveryGetFilter {
  id?: number;
  deliveredTo?: number;
  deliveredFrom?: number;
}

export interface IDeliveryGet {
  limit?: number;
  page?: number;
  filter?: IDeliveryGetFilter;
  sort?: IGetSort;
}

```

header.interface.ts

```

export interface IHeaderProps {
  isAuthorized: boolean;
  name?: string;
  logoutThunk(): void;
}

```

order.interface.ts

```

import { IGetSort } from '../../utils/interfaces/get.sort.interface';
import { ClothSizes } from './client.interface';

export enum OrderStatuses {
  CREATED = 'CREATED',
  PROCESSING = 'PROCESSING',
  PAYED = 'PAYED',
  SENT = 'SENT',
  DELIVERED = 'DELIVERED',
  COMPLETED = 'COMPLETED',
  RETURNED = 'RETURNED',
  CANCELED = 'CANCELED',
}

export interface IClothId {
  clothId: number;
  amount: number;
  size: ClothSizes;
}

export interface IOrder {
  id: number;
  clientId: string;
  clothIdList: IClothId[];
  status: OrderStatuses;
  price: number;
  createdBy: string;
  createdAt: string;
  updatedBy: string;
  updatedAt: string;
  deletedBy: string;
}

```

```

}

export interface IOrderState {
  order: IOrder[];
  orderCount: number;
  orderError?: string;
}

export interface IOrderError {
  error: string;
}

export interface IOrderProps {
  order: IOrderState;
  getOrderThunk(orderGetData?: IOrderGet): void;
  updateOrderThunk(orderToUpdate: IOrderUpdate): void;
  createOrderThunk(orderToCreate: IOrderCreate): void;
  deleteOrderThunk(orderToDelete: IOrderDelete): void;
}

export interface IOrderCreate {
  clientId: string;
  clothIdList: IClothId[];
  status?: OrderStatuses;
}

export interface IOrderUpdate {
  id: number;
  clientId?: string;
  clothIdList?: IClothId[];
  status?: OrderStatuses;
}

export interface IOrderDelete {
  id: number;
}

export interface IOrderGetFilter {
  id?: number;
  clientId?: string;
  status?: OrderStatuses;
}

export interface IOrderGet {
  limit?: number;
  page?: number;
  filter?: IOrderGetFilter;
  sort?: IGetSort
}

```

reports.interface.ts

```

import { ClothSizes } from './client.interface';

export interface IStoreBySales {
  storeId: number;
  numberOfSales: number;
  change: number;
}

export interface IClothBySales {
  clothId: number;
  numberOfSales: number;
  change: number;
}

export interface IIncomeByMonth {
  income: number;
  monthNumber: number;
}

export interface IHalfYearIncome {
  totalIncome: number;
  incomeByMonth: IIncomeByMonth[];
}

```

```

export interface IStaffChangesByMonth {
  firedStaffCount: number;
  hiredStaffCount: number;
  monthNumber: number;
}

export interface IStaffChanges {
  staffChangesByMonth: IStaffChangesByMonth[];
  totalStaffNumber: number;
}

export interface ISizesPopularityByMonth {
  size: ClothSizes;
  numberOfSales: number;
  change: number;
}

export interface IReports {
  storesBySales: IStoreBySales[];
  clothesBySales: IClothBySales[];
  halfYearIncome: IHalfYearIncome;
  staffChanges: IStaffChanges;
  sizePopularity: ISizesPopularityByMonth[];
}

export interface IReportsState {
  reports: IReports;
  reportsError?: string;
}

export interface IReportsError {
  error: string;
}

export interface IReportsProps {
  reports: IReportsState;
  getReportsThunk(): void;
}

```

staff.interface.ts

```

import { IGetSort } from '../utils/interfaces/get.sort.interface';

export interface IStaff {
  id: number;
  name: string;
  email: string;
  isAdmin: boolean;
  password: string;
  position: string;
  storeId: number;
  createdBy: string;
  createdAt: string;
  updatedBy: string;
  updatedAt: string;
  deletedBy: string;
}

export interface IStaffState {
  staff: IStaff[];
  staffCount: number;
  staffError?: string;
}

export interface IStaffError {
  error: string;
}

export interface IStaffProps {
  staff: IStaffState;
  getStaffThunk(staffGetData: IStaffGet): void;
  updateStaffThunk(staffToUpdate: IStaffUpdate): void;
  createStaffThunk(staffToCreate: IStaffCreate): void;
  deleteStaffThunk(staffToDelete: IStaffDelete): void;
}

```

```

export interface IStaffCreate {
  name: string;
  email: string;
  position: string;
  isAdmin: boolean;
  password?: string;
  storeId: number;
}

export interface IStaffUpdate {
  id: number;
  name?: string;
  email?: string;
  position?: string;
  isAdmin?: boolean;
  password?: string;
  storeId?: number;
}

export interface IStaffDelete {
  id: number;
}

export interface IStaffGetFilter {
  id?: number;
  name?: string;
  email?: string;
  position?: string;
  isAdmin?: boolean;
  storeId?: number;
}

export interface IStaffGet {
  limit: number;
  page: number;
  filter?: IStaffGetFilter;
  sort?: IGetSort;
}

```

state.interface.ts

```

import { IAuthState } from './auth.interface';
import { IClientState } from './client.interface';
import { IClothState } from './cloth.interface';
import { IDeliveryState } from './delivery.interface';
import { IOrderState } from './order.interface';
import { IReportsState } from './reports.interface';
import { IStaffState } from './staff.interface';
import { IStoreState } from './store.interface';

export interface IState {
  staff: IStaffState;
  client: IClientState;
  cloth: IClothState;
  order: IOrderState;
  store: IStoreState;
  delivery: IDeliveryState;
  reports: IReportsState;
  auth: IAuthState;
}

```

store.interface.ts

```

import { IGetSort } from '../utils/interfaces/get.sort.interface';

export interface IStore {
  id: number;
  address: string;
  isActive: boolean;
  createdBy: string;
  createdAt: string;
  updatedBy: string;
  updatedAt: string;
  deletedBy: string;
}

```

```

export interface IStoreState {
  store: IStore[];
  storeCount: number;
  storeError?: string;
}

export interface IStoreError {
  error: string;
}

export interface IStoreProps {
  store: IStoreState;
  getStoreThunk(storeGetData?: IStoreGet): void;
  updateStoreThunk(storeToUpdate: IStoreUpdate): void;
  createStoreThunk(storeToCreate: IStoreCreate): void;
  deleteStoreThunk(storeToDelete: IStoreDelete): void;
}

export interface IStoreCreate {
  address: string;
  isActive: boolean;
}

export interface IStoreUpdate {
  id: number;
  address?: string;
  isActive?: boolean;
}

export interface IStoreDelete {
  id: number;
}

export interface IStoreGetFilter {
  id?: number;
  address?: string;
  isActive?: boolean;
}

export interface IStoreGet {
  limit?: number;
  page?: number;
  filter?: IStoreGetFilter;
  sort?: IGetSort
}

```

authReducer.ts

```

import { Dispatch } from 'redux';
import { IAuthState, IAuthorize, ILogout } from '../interfaces';
import { AuthApi } from '../../Api/auth.api';

const initialState = {} as IAuthState;

const AUTHORIZE = 'AUTHORIZE',
  CLEAR_ERROR = 'CLEAR_ERROR',
  LOGOUT = 'LOGOUT',
  SET_AUTHORIZE = 'SET_AUTHORIZE';

export type ActionTypes = AuthorizeActionType | SetAuthorizeActionType | ClearErrorActionType | LogoutActionType;

const authReducer = (state = initialState, action: ActionTypes): IAuthState => {
  switch (action.type) {
    case AUTHORIZE: {
      return action.data;
    }

    case SET_AUTHORIZE: {
      const newAuth = { ...state.auth };
      newAuth.isAuthorized = action.data.isAuthorized;
      newAuth.name = action.data.userName;
      return { ...state, auth: newAuth };
    }

    case CLEAR_ERROR: {

```

```

    if (state.auth.error === undefined) {
      return state;
    }
    return { ...state, auth: { ...state.auth, error: undefined } };
  }

  case LOGOUT: {
    if (action.data.error) {
      return { ...state, auth: { ...state.auth, error: action.data.error } };
    }
    return { ...state, auth: { isAuthorized: false, name: undefined } };
  }

  default:
    return state;
  }
};

type AuthorizeActionType = {
  type: typeof AUTHORIZE;
  data: IAuthState;
}

type LogoutActionType = {
  type: typeof LOGOUT;
  data: ILogout;
}

type SetAuthorizeActionType = {
  type: typeof SET_AUTHORIZE;
  data: {
    isAuthorized: boolean,
    userName?: string
  };
}

type ClearErrorActionType = {
  type: typeof CLEAR_ERROR;
  data: undefined;
}

// actions
export const setAuthorized = (isAuthorized: boolean, userName?: string): SetAuthorizeActionType => {
  return { type: SET_AUTHORIZE, data: { isAuthorized, userName } };
};

export const clearError = (): ClearErrorActionType => {
  return { type: CLEAR_ERROR, data: undefined };
};

// thunks
export const authorizeThunk = (authData: IAuthorize) => async (dispatch: Dispatch<ActionTypes>) => {
  const authResp = await AuthApi.authorize(authData);

  dispatch({ type: AUTHORIZE, data: authResp });
};

export const logoutThunk = () => async (dispatch: Dispatch<ActionTypes>) => {
  const logoutResp = await AuthApi.logout();

  dispatch({ type: LOGOUT, data: logoutResp });
};

export default authReducer;

```

clientReducer.ts

```

import { Dispatch } from 'redux';
import { IClientCreate, IClientDelete, IClientGet, IClientState, IClientUpdate } from '../interfaces';
import { ClientApi } from '../../Api/client.api';

const initialState = {} as IClientState;

const CLEAR_ERROR = 'CLEAR_ERROR',
  GET_CLIENT = 'GET_CLIENT',
  SET_CLIENT_ERROR = 'SET_CLIENT_ERROR';

```

```

export type ActionTypes = GetClientActionType | SetErrorActionType | ClearErrorActionType

const clientReducer = (state = initialState, action: ActionTypes): IClientState => {
  switch (action.type) {
    case GET_CLIENT: {
      return action.data;
    }

    case CLEAR_ERROR: {
      if (state.clientError === undefined) {
        return state;
      }
      return { ...state, clientError: undefined };
    }

    case SET_CLIENT_ERROR: {
      return { ...state, clientError: action.data };
    }

    default:
      return state;
  }
};

type GetClientActionType = {
  type: typeof GET_CLIENT;
  data: IClientState
}

type SetErrorActionType = {
  type: typeof SET_CLIENT_ERROR;
  data: string;
}

type ClearErrorActionType = {
  type: typeof CLEAR_ERROR;
  data: undefined;
}

// thunks
export const getClientThunk = (clientData: IClientGet) => async (dispatch: Dispatch<ActionTypes>) => {
  const clientResp = await ClientApi.getAllClient(clientData);
  if ('error' in clientResp) {
    dispatch({ type: SET_CLIENT_ERROR, data: clientResp.error });
    return;
  }

  dispatch({ type: GET_CLIENT, data: clientResp });
};

export const createClientThunk = (clientToCreate: IClientCreate) => async (dispatch: Dispatch<ActionTypes>)
=> {
  const client = await ClientApi.createClient(clientToCreate);

  if ('error' in client) {
    dispatch({ type: SET_CLIENT_ERROR, data: client.error });
  }
};

export const updateClientThunk = (clientToUpdate: IClientUpdate) => async (dispatch: Dispatch<ActionTypes>)
=> {
  const client = await ClientApi.updateClient(clientToUpdate);

  if ('error' in client) {
    dispatch({ type: SET_CLIENT_ERROR, data: client.error });
  }
};

export const deleteClientThunk = (clientToDelete: IClientDelete) => async (dispatch: Dispatch<ActionTypes>)
=> {
  const client = await ClientApi.deleteClient(clientToDelete);

  if (typeof client === 'object') {
    dispatch({ type: SET_CLIENT_ERROR, data: client.error });
  }
};

```

```
export default clientReducer;
```

clothReducer.ts

```
import { Dispatch } from 'redux';
import {
  IClothCreate,
  IClothDelete,
  IClothGet,
  IClothSizeInShops,
  IClothState,
  IClothUpdate,
} from '../interfaces';
import { ClothApi } from '../../Api/cloth.api';

const initialState = {} as IClothState;

const CLEAR_ERROR = 'CLEAR_ERROR',
  GET_CLOTH = 'GET_CLOTH',
  GET_CLOTH_SIZES = 'GET_CLOTH_SIZES',
  SET_CLOTH_ERROR = 'SET_CLOTH_ERROR';

export type ActionTypes =
  GetClothActionType |
  GetClothSizesActionType |
  SetErrorActionType |
  ClearErrorActionType

const clothReducer = (state = initialState, action: ActionTypes): IClothState => {
  switch (action.type) {
    case GET_CLOTH: {
      return action.data;
    }

    case GET_CLOTH_SIZES: {
      const clothIndex = state.sizesByShop?.findIndex((sizeByShop) => sizeByShop.clothId ===
action.data.clothId);
      let newSizesByShop = state.sizesByShop || [];

      if (clothIndex && clothIndex !== -1) {
        newSizesByShop = newSizesByShop?.map((sizeByShop) => {
          if (sizeByShop.clothId === action.data.clothId) {
            return action.data;
          }
          return sizeByShop;
        });
      } else {
        newSizesByShop.push(action.data);
      }

      return { ...state, sizesByShop: newSizesByShop };
    }

    case CLEAR_ERROR: {
      if (state.clothError === undefined) {
        return state;
      }
      return { ...state, clothError: undefined };
    }

    case SET_CLOTH_ERROR: {
      return { ...state, clothError: action.data };
    }

    default:
      return state;
  }
};

type GetClothActionType = {
  type: typeof GET_CLOTH;
  data: IClothState
}

type GetClothSizesActionType = {
```

```

    type: typeof GET_CLOTH_SIZES;
    data: IClothSizeInShops
  }

  type SetErrorActionType = {
    type: typeof SET_CLOTH_ERROR;
    data: string;
  }

  type ClearErrorActionType = {
    type: typeof CLEAR_ERROR;
    data: undefined;
  }

  // thunks
  export const getClothThunk = (clothData: IClothGet) => async (dispatch: Dispatch<ActionTypes>) => {
    const clothResp = await ClothApi.getAllCloth(clothData);

    if ('error' in clothResp) {
      dispatch({ type: SET_CLOTH_ERROR, data: clothResp.error });
      return;
    }

    dispatch({ type: GET_CLOTH, data: clothResp });
  };

  export const getClothSizesThunk = (shopId: number) => async (dispatch: Dispatch<ActionTypes>) => {
    const clothResp = await ClothApi.getClothSizeCountByShop(shopId);

    if ('error' in clothResp) {
      dispatch({ type: SET_CLOTH_ERROR, data: clothResp.error });
      return;
    }

    dispatch({ type: GET_CLOTH_SIZES, data: clothResp });
  };

  export const createClothThunk = (clothToCreate: IClothCreate) => async (dispatch: Dispatch<ActionTypes>) => {
    const cloth = await ClothApi.createCloth(clothToCreate);

    if ('error' in cloth) {
      dispatch({ type: SET_CLOTH_ERROR, data: cloth.error });
    }
  };

  export const updateClothThunk = (clothToUpdate: IClothUpdate) => async (dispatch: Dispatch<ActionTypes>) => {
    const cloth = await ClothApi.updateCloth(clothToUpdate);

    if ('error' in cloth) {
      dispatch({ type: SET_CLOTH_ERROR, data: cloth.error });
    }
  };

  export const deleteClothThunk = (clothToDelete: IClothDelete) => async (dispatch: Dispatch<ActionTypes>) => {
    const cloth = await ClothApi.deleteCloth(clothToDelete);

    if (typeof cloth === 'object') {
      dispatch({ type: SET_CLOTH_ERROR, data: cloth.error });
    }
  };

  export default clothReducer;

```

deliveryReducer.ts

```

import { Dispatch } from 'redux';
import {
  IDeliveryCreate,
  IDeliveryDelete,
  IDeliveryGet,
  IDeliveryState,
  IDeliveryUpdate,
} from '../interfaces';
import { DeliveryApi } from '../Api/delivery.api';

const initialState = {} as IDeliveryState;

```

```

const CLEAR_ERROR = 'CLEAR_ERROR',
      GET_DELIVERY = 'GET_DELIVERY',
      SET_DELIVERY_ERROR = 'SET_DELIVERY_ERROR';

export type ActionTypes =
  GetDeliveryActionType |
  SetErrorActionType |
  ClearErrorActionType

const deliveryReducer = (state = initialState, action: ActionTypes): IDeliveryState => {
  switch (action.type) {
    case GET_DELIVERY: {
      return action.data;
    }

    case CLEAR_ERROR: {
      if (state.deliveryError === undefined) {
        return state;
      }
      return { ...state, deliveryError: undefined };
    }

    case SET_DELIVERY_ERROR: {
      return { ...state, deliveryError: action.data };
    }

    default:
      return state;
  }
};

type GetDeliveryActionType = {
  type: typeof GET_DELIVERY;
  data: IDeliveryState
}

type SetErrorActionType = {
  type: typeof SET_DELIVERY_ERROR;
  data: string;
}

type ClearErrorActionType = {
  type: typeof CLEAR_ERROR;
  data: undefined;
}

// thunks
export const getDeliveryThunk = (deliveryData: IDeliveryGet) => async (dispatch: Dispatch<ActionTypes>) => {
  const deliveryResp = await DeliveryApi.getAllDelivery(deliveryData);

  if ('error' in deliveryResp) {
    dispatch({ type: SET_DELIVERY_ERROR, data: deliveryResp.error });
    return;
  }

  dispatch({ type: GET_DELIVERY, data: deliveryResp });
};

export const createDeliveryThunk = (deliveryToCreate: IDeliveryCreate) => async (dispatch:
Dispatch<ActionTypes>) => {
  const delivery = await DeliveryApi.createDelivery(deliveryToCreate);

  if ('error' in delivery) {
    dispatch({ type: SET_DELIVERY_ERROR, data: delivery.error });
  }
};

export const updateDeliveryThunk = (deliveryToUpdate: IDeliveryUpdate) => async (dispatch:
Dispatch<ActionTypes>) => {
  const delivery = await DeliveryApi.updateDelivery(deliveryToUpdate);

  if ('error' in delivery) {
    dispatch({ type: SET_DELIVERY_ERROR, data: delivery.error });
  }
};

```

```

export const deleteDeliveryThunk = (deliveryToDelete: IDeliveryDelete) => async (dispatch:
Dispatch<ActionTypes>) => {
  const delivery = await DeliveryApi.deleteDelivery(deliveryToDelete);

  if (typeof delivery === 'object') {
    dispatch({ type: SET_DELIVERY_ERROR, data: delivery.error });
  }
};

export default deliveryReducer;

```

orderReducer.ts

```

import { Dispatch } from 'redux';
import { IOrderCreate, IOrderDelete, IOrderGet, IOrderState, IOrderUpdate } from '../interfaces';
import { OrderApi } from '../Api/order.api';

const initialState = {} as IOrderState;

const CLEAR_ERROR = 'CLEAR_ERROR',
      GET_ORDER = 'GET_ORDER',
      SET_ORDER_ERROR = 'SET_ORDER_ERROR';

export type ActionTypes = GetOrderActionType | SetErrorActionType | ClearErrorActionType

const orderReducer = (state = initialState, action: ActionTypes): IOrderState => {
  switch (action.type) {
    case GET_ORDER: {
      return action.data;
    }

    case CLEAR_ERROR: {
      if (state.orderError === undefined) {
        return state;
      }
      return { ...state, orderError: undefined };
    }

    case SET_ORDER_ERROR: {
      return { ...state, orderError: action.data };
    }

    default:
      return state;
  }
};

type GetOrderActionType = {
  type: typeof GET_ORDER;
  data: IOrderState
}

type SetErrorActionType = {
  type: typeof SET_ORDER_ERROR;
  data: string;
}

type ClearErrorActionType = {
  type: typeof CLEAR_ERROR;
  data: undefined;
}

// thunks
export const getOrderThunk = (orderData: IOrderGet) => async (dispatch: Dispatch<ActionTypes>) => {
  const orderResp = await OrderApi.getAllOrder(orderData);

  if ('error' in orderResp) {
    dispatch({ type: SET_ORDER_ERROR, data: orderResp.error });
    return;
  }

  dispatch({ type: GET_ORDER, data: orderResp });
};

export const createOrderThunk = (orderToCreate: IOrderCreate) => async (dispatch: Dispatch<ActionTypes>) => {
  const order = await OrderApi.createOrder(orderToCreate);

```

```

    if ('error' in order) {
      dispatch({ type: SET_ORDER_ERROR, data: order.error });
    }
  };

export const updateOrderThunk = (orderToUpdate: IOrderUpdate) => async (dispatch: Dispatch<ActionTypes>) => {
  const order = await OrderApi.updateOrder(orderToUpdate);

  if ('error' in order) {
    dispatch({ type: SET_ORDER_ERROR, data: order.error });
  }
};

export const deleteOrderThunk = (orderToDelete: IOrderDelete) => async (dispatch: Dispatch<ActionTypes>) => {
  const order = await OrderApi.deleteOrder(orderToDelete);

  if (typeof order === 'object') {
    dispatch({ type: SET_ORDER_ERROR, data: order.error });
  }
};

export default orderReducer;

```

reportsReducer.ts

```

import { Dispatch } from 'redux';
import { IReportsState } from '../interfaces';
import { ReportsApi } from '../Api/reports.api';

const initialState = {} as IReportsState;

const CLEAR_ERROR = 'CLEAR_ERROR',
  GET_REPORTS = 'GET_REPORTS',
  SET_REPORTS_ERROR = 'SET_REPORTS_ERROR';

export type ActionTypes = GetReportsActionType | SetErrorActionType | ClearErrorActionType

const reportsReducer = (state = initialState, action: ActionTypes): IReportsState => {
  switch (action.type) {
    case GET_REPORTS: {
      return action.data;
    }

    case CLEAR_ERROR: {
      if (state.reportsError === undefined) {
        return state;
      }
      return { ...state, reportsError: undefined };
    }

    case SET_REPORTS_ERROR: {
      return { ...state, reportsError: action.data };
    }
    default:
      return state;
  }
};

type GetReportsActionType = {
  type: typeof GET_REPORTS;
  data: IReportsState
}

type SetErrorActionType = {
  type: typeof SET_REPORTS_ERROR;
  data: string;
}

type ClearErrorActionType = {
  type: typeof CLEAR_ERROR;
  data: undefined;
}

// thunks
export const getReportsThunk = () => async (dispatch: Dispatch<ActionTypes>) => {
  const reportsResp = await ReportsApi.getReports();

```

```

    if ('error' in reportsResp) {
      dispatch({ type: SET_REPORTS_ERROR, data: reportsResp.error });
      return;
    }

    dispatch({ type: GET_REPORTS, data: reportsResp });
  };

export default reportsReducer;

```

staffReducer.ts

```

import { Dispatch } from 'redux';
import { IStaffCreate, IStaffDelete, IStaffGet, IStaffState, IStaffUpdate } from '../interfaces';
import { StaffApi } from '../../Api/staff.api';

const initialState = {} as IStaffState;

const CLEAR_ERROR = 'CLEAR_ERROR',
      GET_STAFF = 'GET_STAFF',
      SET_STAFF_ERROR = 'SET_STAFF_ERROR';

export type ActionTypes = GetStaffActionType
| SetErrorActionType
| ClearErrorActionType

const staffReducer = (state = initialState, action: ActionTypes): IStaffState => {
  switch (action.type) {
    case GET_STAFF: {
      return action.data;
    }

    case CLEAR_ERROR: {
      if (state.staffError === undefined) {
        return state;
      }
      return { ...state, staffError: undefined };
    }

    case SET_STAFF_ERROR: {
      return { ...state, staffError: action.data };
    }

    default:
      return state;
  }
};

type GetStaffActionType = {
  type: typeof GET_STAFF;
  data: IStaffState;
}

type SetErrorActionType = {
  type: typeof SET_STAFF_ERROR;
  data: string;
}

type ClearErrorActionType = {
  type: typeof CLEAR_ERROR;
  data: undefined;
}

// thunks
export const getStaffThunk = (staffData: IStaffGet) => async (dispatch: Dispatch<ActionTypes>) => {
  const staffResp = await StaffApi.getAllStaff(staffData);

  if ('error' in staffResp) {
    dispatch({ type: SET_STAFF_ERROR, data: staffResp.error });
    return;
  }

  dispatch({ type: GET_STAFF, data: staffResp });
};

```

```

export const createStaffThunk = (staffToCreate: IStaffCreate) => async (dispatch: Dispatch<ActionTypes>) => {
  const staffResp = await StaffApi.createStaff(staffToCreate);

  if ('error' in staffResp) {
    dispatch({ type: SET_STAFF_ERROR, data: staffResp.error });
  }
};

export const updateStaffThunk = (staffToUpdate: IStaffUpdate) => async (dispatch: Dispatch<ActionTypes>) => {
  const staff = await StaffApi.updateStaff(staffToUpdate);

  if ('error' in staff) {
    dispatch({ type: SET_STAFF_ERROR, data: staff.error });
  }
};

export const deleteStaffThunk = (staffToDelete: IStaffDelete) => async (dispatch: Dispatch<ActionTypes>) => {
  const staff = await StaffApi.deleteStaff(staffToDelete);

  if (typeof staff === 'object') {
    dispatch({ type: SET_STAFF_ERROR, data: staff.error });
  }
};

export default staffReducer;

```

storeReducer.ts

```

import { Dispatch } from 'redux';
import { IStoreCreate, IStoreDelete, IStoreGet, IStoreState, IStoreUpdate } from '../interfaces';
import { StoreApi } from '../../Api/store.api';

const initialState = {} as IStoreState;

const CLEAR_ERROR = 'CLEAR_ERROR',
      GET_STORE = 'GET_STORE',
      SET_STORE_ERROR = 'SET_STORE_ERROR';

export type ActionTypes = GetStoreActionType | SetErrorActionType | ClearErrorActionType

const storeReducer = (state = initialState, action: ActionTypes): IStoreState => {
  switch (action.type) {
    case GET_STORE: {
      return action.data;
    }

    case CLEAR_ERROR: {
      if (state.storeError === undefined) {
        return state;
      }
      return { ...state, storeError: undefined };
    }

    case SET_STORE_ERROR: {
      return { ...state, storeError: action.data };
    }

    default:
      return state;
  }
};

type GetStoreActionType = {
  type: typeof GET_STORE;
  data: IStoreState
}

type SetErrorActionType = {
  type: typeof SET_STORE_ERROR;
  data: string;
}

type ClearErrorActionType = {
  type: typeof CLEAR_ERROR;
  data: undefined;
}

```

```
// thunks
export const getStoreThunk = (storeData: IStoreGet) => async (dispatch: Dispatch<ActionTypes>) => {
  const storeResp = await StoreApi.getAllStore(storeData);

  if ('error' in storeResp) {
    dispatch({ type: SET_STORE_ERROR, data: storeResp.error });
    return;
  }

  dispatch({ type: GET_STORE, data: storeResp });
};

export const createStoreThunk = (storeToCreate: IStoreCreate) => async (dispatch: Dispatch<ActionTypes>) => {
  const store = await StoreApi.createStore(storeToCreate);

  if ('error' in store) {
    dispatch({ type: SET_STORE_ERROR, data: store.error });
  }
};

export const updateStoreThunk = (storeToUpdate: IStoreUpdate) => async (dispatch: Dispatch<ActionTypes>) => {
  const store = await StoreApi.updateStore(storeToUpdate);

  if ('error' in store) {
    dispatch({ type: SET_STORE_ERROR, data: store.error });
  }
};

export const deleteStoreThunk = (storeToDelete: IStoreDelete) => async (dispatch: Dispatch<ActionTypes>) => {
  const store = await StoreApi.deleteStore(storeToDelete);

  if (typeof store === 'object') {
    dispatch({ type: SET_STORE_ERROR, data: store.error });
  }
};

export default storeReducer;
```

reduxStore.ts

```
import { applyMiddleware, combineReducers, createStore } from 'redux';
import staffReducer from './reducers/staffReducer';
import ReduxThunk from 'redux-thunk';
import clientReducer from './reducers/clientReducer';
import clothReducer from './reducers/clothReducer';
import orderReducer from './reducers/orderReducer';
import storeReducer from './reducers/storeReducer';
import deliveryReducer from './reducers/deliveryReducer';
import reportsReducer from './reducers/reportsReducer';
import authReducer from './reducers/authReducer';

const reducers = combineReducers({
  staff: staffReducer,
  client: clientReducer,
  cloth: clothReducer,
  order: orderReducer,
  store: storeReducer,
  delivery: deliveryReducer,
  reports: reportsReducer,
  auth: authReducer,
});

const store = createStore(reducers, applyMiddleware(ReduxThunk));

export default store;
```

enum.sort.ts

```
export enum EnumSort {
  asc = 'asc',
  desc = 'desc',
}
```

month-number.enum.ts

```
export enum MonthNumber {
  None,
  Січень,
  Лютий,
  Березень,
  Квітень,
  Травень,
  Червень,
  Липень,
  Серпень,
  Вересень,
  Жовтень,
  Листопад,
  Грудень,
}
```

get.sort.interface.ts

```
import { EnumSort } from '../enums/enum.sort';

export interface IGetSort {
  order?: EnumSort;
  orderBy?: string;
}
```

Back-end

app.config.ts

```
import { Logger } from '@nestjsjs/common';
import { registerAs } from '@nestjsjs/config';
import * as Joi from 'joi';
import * as path from 'path';

const cfgRaw = {
  port: parseInt(process.env.PORT),
  jwtSecret: process.env.JWT_SECRET,
  cookieSecret: process.env.COOKIE_SECRET,
} as const;

Logger.verbose('Validating ' + path.basename(__filename));
Joi.attempt(
  cfgRaw,
  Joi.object({
    port: Joi.number().required(),
    jwtSecret: Joi.string().required(),
    cookieSecret: Joi.string().required(),
  }).required(),
);

export const config = registerAs('module.app', () => cfgRaw);
```

auth.config.ts

```
import { Logger } from '@nestjsjs/common';
import { registerAs } from '@nestjsjs/config';
import * as Joi from 'joi';
import * as path from 'path';

export const cfgRaw = {} as const;

Logger.verbose('Validating ' + path.basename(__filename));
Joi.attempt(cfgRaw, Joi.object({}).required());

export const config = registerAs('module.auth', () => cfgRaw);
```

auth.service.ts

```
import { Inject, Injectable, Logger, NotFoundException } from '@nestjsjs/common';
import { Response } from 'express';
```

```

import { Model } from 'mongoose';
import { LoginRequestDto } from '../auth/dto/login.request.dto';
import {
  STAFF_MODEL_SCHEMA,
  IStaffSchema,
} from '../mongodb/schemas/staff.schema';
import md5 from 'md5';
import { LoginResponsetDto } from '../auth/dto/login.response.dto';
import { JwtService } from '@nestjs/jwt';

/**
 * Service that provides auth logic
 * @param {ConfigService} configService - service to interact with env and settings
 * @param {JwtService} jwtService - service to interact with jwt tokens
 */
@Injectable()
export class AuthService {
  private logger = new Logger(AuthService.name);
  constructor(
    @Inject(STAFF_MODEL_SCHEMA)
    private readonly staffSchema: Model<IStaffSchema>,
    private readonly jwtService: JwtService,
  ) {}

  async login(dto: LoginRequestDto, res: Response): Promise<LoginResponsetDto> {
    try {
      const password = md5(dto.password);
      const user = await this.staffSchema.findOne({
        email: dto.email,
        password,
        deletedBy: null,
        isAdmin: true,
      });
      if (user) {
        const accessToken = await this.generateAccessToken(user);

        res.cookie('isAuthorized', true, { sameSite: 'none', secure: true });
        res.cookie('userName', user.name, { sameSite: 'none', secure: true });
        res.cookie('accessToken', accessToken, {
          httpOnly: true,
          signed: true,
          sameSite: 'none',
          secure: true,
        });

        return { isAuthorized: true, name: user.name, accessToken };
      }
      throw new NotFoundException(
        'Не знайдено користувача, або не правильний пароль',
      );
    } catch (e) {
      this.logger.error(e);
      throw e;
    }
  }

  logout(res: Response) {
    try {
      res.clearCookie('isAuthorized');
      res.clearCookie('accessToken');
      return;
    } catch (e) {
      this.logger.error(e);
      throw e;
    }
  }

  async findAdminUserByEmail(email: string): Promise<IStaffSchema> {
    return this.staffSchema.findOne({ email, deletedBy: null, isAdmin: true });
  }

  private async generateAccessToken(user: IStaffSchema) {
    const token = await this.jwtService.signAsync({
      email: user.email,
      name: user.name,
    });
  }
}

```

```

        return token;
    }
}

```

auth.guard.ts

```

import {
    CanActivate,
    ExecutionContext,
    ForbiddenException,
    Injectable,
    Logger,
} from '@nestjs/common';
import { Reflector } from '@nestjs/core';
import { ConfigService } from '@nestjs/config';
import { JwtService } from '@nestjs/jwt';
import { DECORATOR_PUBLIC } from '../../shared/decorators/public.decorator';
import { inspect } from 'util';
import { AuthService } from '../auth/auth.service';

@Injectable()
export class AuthGuard implements CanActivate {
    private readonly logger = new Logger(AuthGuard.name);

    constructor(
        private readonly configService: ConfigService,
        private readonly reflector: Reflector,
        private readonly jwtService: JwtService,
        private readonly authService: AuthService,
    ) {}

    async canActivate(context: ExecutionContext): Promise<boolean> {
        try {
            if (this.isPublicRoute(context)) {
                return true;
            }
            const req = context.switchToHttp().getRequest();
            const token =
                req?.cookies?.accessToken || req?.signedCookies?.accessToken;

            const tokenPayload = JSON.parse(
                Buffer.from(token.split('.')[1], 'base64').toString(),
            );

            req.userEmail = tokenPayload.email;

            await this.jwtService.verifyAsync(token, {
                secret: this.configService.getOrThrow('JWT_SECRET'),
            });

            const user = await this.authService.findAdminUserByEmail(
                tokenPayload.email,
            );

            if (!user) {
                throw new ForbiddenException(
                    'Користувач більше не має доступу до системи',
                );
            }

            return true;
        } catch (e) {
            this.logger.error(inspect(e));
            return false;
        }
    }

    private isPublicRoute(context: ExecutionContext): boolean {
        return this.reflector.getAllAndOverride<boolean>(DECORATOR_PUBLIC, [
            context.getHandler(),
            context.getClass(),
        ]);
    }
}

```

client.config.ts

```
import { Logger } from '@nestjs/common';
import { registerAs } from '@nestjs/config';
import * as Joi from 'joi';
import * as path from 'path';

export const cfgRaw = {
  nameMinLength: 2,
  emailMinLength: 2,
  phoneNumberMinLength: 2,
  nameMaxLength: 80,
  emailMaxLength: 80,
  phoneNumberMaxLength: 80,
} as const;

Logger.verbose('Validating ' + path.basename(__filename));
Joi.attempt(
  cfgRaw,
  Joi.object({
    nameMinLength: Joi.number().required(),
    emailMinLength: Joi.number().required(),
    phoneNumberMinLength: Joi.number().required(),
    nameMaxLength: Joi.number().required(),
    emailMaxLength: Joi.number().required(),
    phoneNumberMaxLength: Joi.number().required(),
  }).required(),
);

export const config = registerAs('module.client', () => cfgRaw);
```

client.service.ts

```
import {
  Inject,
  Injectable,
  Logger,
  BadRequestException,
  NotFoundException,
  BadGatewayException,
} from '@nestjs/common';
import { Model } from 'mongoose';
import { ClientCreateRequestDto } from '../client/dto/client-create.request.dto';
import { GetClientRequest } from '../client/dto/client-get.request.dto';
import { GetClientResponse } from '../client/dto/client-get.response.dto';
import { ClientUpdateRequestDto } from '../client/dto/client-update.request.dto';
import {
  CLIENT_MODEL_SCHEMA,
  IClientSchema,
} from '../mongodb/schemas/client.schema';
import { EnumSort } from '../shared/enums/sort.enum';

/**
 * Service that provides client management
 * @param {ConfigService} configService - service to interact with env and settings
 * @param {JwtService} jwtService - service to interact with jwt tokens
 */
@Injectable()
export class ClientService {
  private logger = new Logger(ClientService.name);
  constructor(
    @Inject(CLIENT_MODEL_SCHEMA)
    private readonly clientSchema: Model<IClientSchema>,
  ) {}

  async createClient(
    dto: ClientCreateRequestDto,
    userEmail: string,
  ): Promise<IClientSchema> {
    try {
      const existingEmail = await this.clientSchema.findOne({
        email: dto.email,
      });

      if (existingEmail) {
        throw new BadRequestException('Клієнт з такою поштою вже існує');
      }
    }
  }
}
```

```

const lastClient = await this.clientSchema
  .find(null, null, { sort: { id: -1 } })
  .limit(1);

return this.clientSchema.create({
  ...dto,
  id: (lastClient[0]?.id || 0) + 1,
  createdAt: new Date().toISOString(),
  createdBy: userEmail,
});
} catch (e) {
  this.logger.error(e);
  throw e;
}
}

async getClient(dto: GetClientRequest): Promise<GetClientResponse> {
  try {
    const {
      limit,
      page,
      order,
      orderBy,
      email,
      name,
      phoneNumber,
      ...filter
    } = dto;
    const skip = limit * page;

    const aggregateFilter = {} as Record<string, any>;

    if (email) {
      aggregateFilter.email = { $regex: email, $options: 'i' };
    }

    if (name) {
      aggregateFilter.name = { $regex: name, $options: 'i' };
    }

    if (phoneNumber) {
      aggregateFilter.phoneNumber = { $regex: phoneNumber, $options: 'i' };
    }

    const client = await this.clientSchema
      .find(
        {
          deletedBy: null,
          ...filter,
          ...aggregateFilter,
        },
        null,
        { sort: { [orderBy || 'id']: order === EnumSort.desc ? -1 : 1 } },
      )
      .limit(limit)
      .skip(skip);

    const clientCount = await this.clientSchema.count({
      deletedBy: null,
      ...filter,
      ...aggregateFilter,
    });

    return {
      client,
      clientCount,
    };
  } catch (e) {
    this.logger.error(e);
    throw e;
  }
}

async updateClient(
  dto: ClientUpdateRequestDto,
  userEmail: string,

```

```

): Promise<IClientSchema> {
  try {
    const { id, ...updateData } = dto;
    const existingClient = await this.clientSchema.findOne({
      id: dto.id,
      deletedBy: null,
    });
    if (!existingClient) {
      throw new NotFoundException(`Не знайдено клієнта з ID ${id}`);
    }

    const updateResult = await existingClient.updateOne({
      ...updateData,
      updatedAt: new Date().toISOString(),
      updatedBy: userEmail,
    });

    if (updateResult.modifiedCount === 0) {
      throw new BadGatewayException('Помилка під час оновлення');
    }

    return this.clientSchema.findOne({
      id: dto.id,
      deletedBy: null,
    });
  } catch (e) {
    this.logger.error(e);
    throw e;
  }
}

async softDeleteClient(id: number, userEmail: string): Promise<number> {
  try {
    const existingClient = await this.clientSchema.findOne({
      id,
      deletedBy: null,
    });
    if (!existingClient) {
      throw new NotFoundException(`Не знайдено клієнта з ID ${id}`);
    }

    const updateResult = await existingClient.updateOne({
      updatedAt: new Date().toISOString(),
      deletedBy: userEmail,
    });

    if (updateResult.modifiedCount === 0) {
      throw new BadGatewayException('Помилка під час оновлення');
    }

    return id;
  } catch (e) {
    this.logger.error(e);
    throw e;
  }
}
}

```

cloth.config.ts

```

import { Logger } from '@nestjs/common';
import { registerAs } from '@nestjs/config';
import * as Joi from 'joi';
import * as path from 'path';

export const cfgRaw = {
  nameMinLength: 2,
  nameMaxLength: 80,
  descMinLength: 2,
  descMaxLength: 80,
} as const;

Logger.verbose('Validating ' + path.basename(__filename));
Joi.attempt(
  cfgRaw,
  Joi.object({

```

```

    nameMinLength: Joi.number().required(),
    nameMaxLength: Joi.number().required(),
    descMinLength: Joi.number().required(),
    descMaxLength: Joi.number().required(),
  }).required(),
);

export const config = registerAs('module.client', () => cfgRaw);

```

cloth.service.ts

```

import {
  Inject,
  Injectable,
  Logger,
  BadRequestException,
  NotFoundException,
  BadGatewayException,
} from '@nestjsjs/common';
import { Model } from 'mongoose';
import { ClothCreateRequestDto } from '../cloth/dto/cloth-create.request.dto';
import { GetClothRequest } from '../cloth/dto/cloth-get.request.dto';
import { GetClothResponse } from '../cloth/dto/cloth-get.response.dto';
import { ClothUpdateRequestDto } from '../cloth/dto/cloth-update.request.dto';
import {
  IClothSizesInShop,
  IClothSizesInShops,
} from '../cloth/dto/cloth-get-details.response.dto';
import {
  CLOTH_MODEL_SCHEMA,
  IClothSchema,
} from '../mongodb/schemas/cloth.schema';
import { EnumSort } from '../shared/enums/sort.enum';
import {
  IWarehouseSchema,
  WAREHOUSE_MODEL_SCHEMA,
} from '../mongodb/schemas/warehouse.schema';

/**
 * Service that provides cloth management
 * @param {ConfigService} configService - service to interact with env and settings
 * @param {JwtService} jwtService - service to interact with jwt tokens
 */
@Injectable()
export class ClothService {
  private logger = new Logger(ClothService.name);
  constructor(
    @Inject(CLOTH_MODEL_SCHEMA)
    private readonly clothSchema: Model<IClothSchema>,
    @Inject(WAREHOUSE_MODEL_SCHEMA)
    private readonly warehouseSchema: Model<IWarehouseSchema>,
  ) {}

  async createCloth(
    dto: ClothCreateRequestDto,
    userEmail: string,
  ): Promise<IClothSchema> {
    try {
      const existingName = await this.clothSchema.findOne({
        name: dto.name,
      });

      if (existingName) {
        throw new BadRequestException('Одяг з такою назвою вже існує');
      }

      const lastCloth = await this.clothSchema
        .find(null, null, { sort: { id: -1 } })
        .limit(1);

      return this.clothSchema.create({
        ...dto,
        id: (lastCloth[0]?.id || 0) + 1,
        createdAt: new Date().toISOString(),
        createdBy: userEmail,
      });
    }
  }

```

```

    } catch (e) {
      this.logger.error(e);
      throw e;
    }
  }

  async getCloth(dto: GetClothRequest): Promise<GetClothResponse> {
    try {
      const {
        limit,
        page,
        order,
        orderBy,
        desc,
        priceFrom,
        priceTo,
        ...filter
      } = dto;
      const skip = limit * page;

      const aggregateFilter = {} as Record<string, any>;

      if (desc) {
        aggregateFilter.email = { $regex: desc, $options: 'i' };
      }

      if (priceFrom || priceTo) {
        if (priceFrom && priceTo) {
          aggregateFilter.price = { $gte: priceFrom, $lte: priceTo };
        } else if (priceFrom) {
          aggregateFilter.price = { $gte: priceFrom };
        } else {
          aggregateFilter.price = { $lte: priceTo };
        }
      }

      const cloth = await this.clothSchema
        .find(
          {
            deletedBy: null,
            ...filter,
            ...aggregateFilter,
          },
          null,
          { sort: { [orderBy || 'id']: order === EnumSort.desc ? -1 : 1 } },
        )
        .limit(limit)
        .skip(skip);

      const clothCount = await this.clothSchema.count({
        deletedBy: null,
        ...filter,
        ...aggregateFilter,
      });

      return {
        cloth,
        clothCount,
      };
    } catch (e) {
      this.logger.error(e);
      throw e;
    }
  }

  async getClothDetails(id: number): Promise<IClothSizesInShops> {
    const cloth = await this.clothSchema.findOne({ id });

    if (!cloth) {
      throw new NotFoundException(`Одяг з ID ${id} не знайдено`);
    }

    const warehouse = await this.warehouseSchema.find({
      clothId: id,
      deletedBy: null,
      amount: { $gt: 0 },
    });
  }

```

```

const shops = warehouse.reduce((prev, cur) => {
  if (!prev.find((val) => val.shopId === cur.storeId)) {
    prev.push({
      shopId: cur.storeId,
      sizes: [{ size: cur.size, count: cur.amount }],
    });
  } else {
    prev = prev.map((val) => {
      if (val.shopId === cur.storeId) {
        return {
          shopId: val.shopId,
          sizes: [...val.sizes, { size: cur.size, count: cur.amount }],
        };
      }
      return val;
    });
  }
  return prev;
}, [] as IClothSizesInShop[]);

return { clothId: id, shops };
}

async updateCloth(
  dto: ClothUpdateRequestDto,
  userEmail: string,
): Promise<IClothSchema> {
  try {
    const { id, ...updateData } = dto;
    const existingCloth = await this.clothSchema.findOne({
      id: dto.id,
      deletedBy: null,
    });
    if (!existingCloth) {
      throw new NotFoundException(`Не знайдено одяг з ID ${id}`);
    }

    const updateResult = await existingCloth.updateOne({
      ...updateData,
      updatedAt: new Date().toISOString(),
      updatedBy: userEmail,
    });

    if (updateResult.modifiedCount === 0) {
      throw new BadGatewayException('Помилка під час оновлення');
    }

    return this.clothSchema.findOne({
      id: dto.id,
      deletedBy: null,
    });
  } catch (e) {
    this.logger.error(e);
    throw e;
  }
}

async softDeleteCloth(id: number, userEmail: string): Promise<number> {
  try {
    const existingCloth = await this.clothSchema.findOne({
      id,
      deletedBy: null,
    });
    if (!existingCloth) {
      throw new NotFoundException(`Не знайдено одяг з ID ${id}`);
    }

    const updateResult = await existingCloth.updateOne({
      updatedAt: new Date().toISOString(),
      deletedBy: userEmail,
    });

    if (updateResult.modifiedCount === 0) {
      throw new BadGatewayException('Помилка під час оновлення');
    }
  }
}

```

```

        return id;
    } catch (e) {
        this.logger.error(e);
        throw e;
    }
}

async getClothesAvailableSizes(id: number): Promise<string[]> {
    const cloth = await this.clothSchema.findOne({ id, deletedBy: null });
    if (!cloth) {
        throw new NotFoundException(`Не знайдено одяг з ID ${id}`);
    }
    return cloth.availableSizes;
}
}

```

delivery.service.ts

```

import {
    Inject,
    Injectable,
    Logger,
    NotFoundException,
    BadGatewayException,
    BadRequestException,
} from '@nestjsjs/common';
import { Model } from 'mongoose';
import {
    DELIVERY_MODEL_SCHEMA,
    IDeliverySchema,
} from '../mongodb/schemas/delivery.schema';
import { DeliveryCreateRequestDto } from '../delivery/dto/delivery-create.request.dto';
import { GetDeliveryRequest } from '../delivery/dto/delivery-get.request.dto';
import { GetDeliveryResponse } from '../delivery/dto/delivery-get.response.dto';
import { DeliveryUpdateRequestDto } from '../delivery/dto/delivery-update.request.dto';
import { EnumSort } from '../../shared/enums/sort.enum';
import {
    IWarehouseSchema,
    WAREHOUSE_MODEL_SCHEMA,
} from '../mongodb/schemas/warehouse.schema';
import { StoreService } from '../store/store.service';
import { ClothService } from '../cloth/cloth.service';

/**
 * Service that provides delivery management
 * @param {ConfigService} configService - service to interact with env and settings
 * @param {JwtService} jwtService - service to interact with jwt tokens
 */
@Injectable()
export class DeliveryService {
    private logger = new Logger(DeliveryService.name);
    constructor(
        @Inject(DELIVERY_MODEL_SCHEMA)
        private readonly deliverySchema: Model<IDeliverySchema>,
        @Inject(WAREHOUSE_MODEL_SCHEMA)
        private readonly warehouseSchema: Model<IWarehouseSchema>,
        private readonly storeService: StoreService,
        private readonly clothService: ClothService,
    ) {}

    async createDelivery(
        dto: DeliveryCreateRequestDto,
        userEmail: string,
    ): Promise<IDeliverySchema> {
        try {
            const lastDelivery = await this.deliverySchema
                .find(null, null, { sort: { id: -1 } })
                .limit(1);

            const receiverStore = await this.storeService.getStoreById(
                dto.deliveredTo,
            );

            if (!receiverStore) {
                throw new NotFoundException(
                    `Не знайдено магазин з ID ${dto.deliveredTo}`,
                );
            }

```

```

    });
  }

  await Promise.all(
    dto.clothDelivered.map(async (clothDelivered) => {
      const clothAvailableSizes =
        await this.clothService.getClothesAvailableSizes(
          clothDelivered.clothId,
        );
      clothDelivered.sizes.map((size) => {
        if (!clothAvailableSizes.includes(size.size)) {
          throw new BadRequestException(
            `Одяг з ID ${clothDelivered.clothId} не може мати розмір ${size.size}`,
          );
        }
      });
    })
  );

  let totalAmountDelivered = 0;

  const warehouseCreateArrays = await Promise.all(
    dto.clothDelivered.map(async (cloth) => {
      const warehouseClothes = await this.warehouseSchema.find({
        clothId: cloth.clothId,
        storeId: dto.deliveredTo,
      });
      return Promise.all(
        cloth.sizes.map(async (size) => {
          const warehouseCloth = warehouseClothes.find(
            (warehouse) => warehouse.size === size.size,
          );

          totalAmountDelivered += size.count;

          if (warehouseCloth) {
            await warehouseCloth.updateOne({
              id: warehouseCloth.id,
              amount: warehouseCloth.amount + size.count,
              updatedAt: new Date().toISOString(),
              updatedBy: userEmail,
            });
          } else {
            return {
              clothId: cloth.clothId,
              storeId: dto.deliveredTo,
              size: size.size,
              amount: size.count,
              createdAt: new Date().toISOString(),
              createdBy: userEmail,
            };
          }
        })
      );
    })
  );

  const warehouseCreateArray = [].concat(...warehouseCreateArrays);

  const lastWarehouse = await this.warehouseSchema
    .find(null, null, { sort: { id: -1 } })
    .limit(1);

  warehouseCreateArray.forEach((warehouseCreate, i) => {
    if (!warehouseCreate) {
      return;
    }
    this.warehouseSchema.create({
      ...warehouseCreate,
      id: (lastWarehouse[0]?.id || 0) + 1 + i,
    });
  });

  return this.deliverySchema.create({
    ...dto,
    totalAmountDelivered,
    id: (lastDelivery[0]?.id || 0) + 1,
  });

```

```

        createdAt: new Date().toISOString(),
        createdBy: userEmail,
        isHidden: false,
    });
} catch (e) {
    this.logger.error(e);
    throw e;
}
}

async getDelivery(dto: GetDeliveryRequest): Promise<GetDeliveryResponse> {
    try {
        const { limit, page, order, orderBy, ...filter } = dto;
        const skip = limit * page;

        const delivery = await this.deliverySchema
            .find(
                {
                    deletedBy: null,
                    ...filter,
                },
                null,
                { sort: { [orderBy || 'id']: order === EnumSort.desc ? -1 : 1 } },
            )
            .limit(limit)
            .skip(skip);

        const deliveryCount = await this.deliverySchema.count({
            deletedBy: null,
            ...filter,
        });

        return {
            delivery,
            deliveryCount,
        };
    } catch (e) {
        this.logger.error(e);
        throw e;
    }
}

async updateDelivery(
    dto: DeliveryUpdateRequestDto,
    userEmail: string,
): Promise<IDeliverySchema> {
    try {
        const { id, ...updateData } = dto;
        const existingDelivery = await this.deliverySchema.findOne({
            id: dto.id,
            deletedBy: null,
        });
        if (!existingDelivery) {
            throw new NotFoundException(`Не знайдено поставку з ID ${id}`);
        }

        const updateResult = await existingDelivery.updateOne({
            ...updateData,
            updatedAt: new Date().toISOString(),
            updatedBy: userEmail,
        });

        if (updateResult.modifiedCount === 0) {
            throw new BadGatewayException('Помилка під час оновлення');
        }

        return this.deliverySchema.findOne({
            id: dto.id,
            deletedBy: null,
        });
    } catch (e) {
        this.logger.error(e);
        throw e;
    }
}

async softDeleteDelivery(id: number, userEmail: string): Promise<number> {

```

```

try {
  const existingDelivery = await this.deliverySchema.findOne({
    id,
    deletedBy: null,
  });
  if (!existingDelivery) {
    throw new NotFoundException(`Не знайдено поставку з ID ${id}`);
  }

  await Promise.all(
    existingDelivery.clothDelivered.map(async (cloth) => {
      const warehouseClothes = await this.warehouseSchema.find({
        clothId: cloth.clothId,
        storeId: existingDelivery.deliveredTo,
      });

      if (!warehouseClothes) {
        throw new BadGatewayException(
          'Не вдається знайти один або декілька товарів з доставки.',
        );
      }

      return Promise.all(
        cloth.sizes.map(async (size) => {
          const warehouseCloth = warehouseClothes.find(
            (warehouse) => warehouse.size === size.size,
          );

          const amount = warehouseCloth.amount - size.count;
          if (amount < 0) {
            throw new BadRequestException(
              `Не можливо видалити поставку. Один або декілька товарів будуть мати від'ємну кількість.`,
            );
          }

          await warehouseCloth.updateOne({
            id: warehouseCloth.id,
            amount,
            updatedAt: new Date().toISOString(),
            updatedBy: userEmail,
          });
        }),
      );
    }),
  );

  const updateResult = await existingDelivery.updateOne({
    updatedAt: new Date().toISOString(),
    deletedBy: userEmail,
  });

  if (updateResult.modifiedCount === 0) {
    throw new BadGatewayException('Помилка під час оновлення');
  }

  return id;
} catch (e) {
  this.logger.error(e);
  throw e;
}
}

```

mongodb.config.ts

```

import { registerAs } from '@nestjs/config';
import { Logger } from '@nestjs/common';
import * as path from 'path';
import * as Joi from 'joi';
import 'dotenv/config';

const cfgRaw = {
  url: process.env.MONGO_URL,
  dbName: process.env.MONGO_DB_NAME,
  connectTimeoutMS: parseInt(process.env.MONGO_CONNECT_TIMEOUT_MS || ''),
} as const;

```

```

Logger.verbose('Validating ' + path.basename(__filename));
Joi.attempt(
  cfgRaw,
  Joi.object({
    url: Joi.string().required(),
    dbName: Joi.string().required(),
    connectTimeoutMS: Joi.number()
      .integer()
      .min(1000)
      .required()
      .label('process.env.MONGO_CONNECT_TIMEOUT_MS'),
  }).required(),
);

export const config = registerAs('module.mongodb', () => cfgRaw);

```

mongodb.constants.ts

```
export const MONGODB_CONNECTION = Symbol('MONGODB_CONNECTION');
```

client.schema.ts

```

import * as mongoose from 'mongoose';
import { Document } from 'mongoose';
import { ClothSizes } from '../../shared/enums/cloth-sizes.enum';

export const CLIENT_MODEL_SCHEMA = Symbol('CLIENT_MODEL_SCHEMA');

export interface IClientSchema extends Document {
  id: number;
  name: string;
  email: string;
  phoneNumber: string;
  size: ClothSizes;
  createdBy: string;
  createdAt: string;
  updatedBy?: string;
  updatedAt?: string;
  deletedBy?: string;
}

export const ClientSchema = new mongoose.Schema<IClientSchema>(
  {
    id: { type: Number, required: true, unique: true },
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    phoneNumber: { type: String, required: true },
    size: { type: String, required: false },
    createdBy: { type: String, required: true },
    createdAt: { type: String, required: true },
    updatedBy: { type: String, required: false },
    updatedAt: { type: String, required: false },
    deletedBy: { type: String, required: false },
  },
  {
    timestamps: false,
  },
);

```

cloth.schema.ts

```

import * as mongoose from 'mongoose';
import { Document } from 'mongoose';
import { ClothSizes } from '../../shared/enums/cloth-sizes.enum';

export const CLOTH_MODEL_SCHEMA = Symbol('CLOTH_MODEL_SCHEMA');

export interface IClothSchema extends Document {
  id: number;
  name: string;
  price: number;
  desc: string;
  availableSizes: ClothSizes[];
  createdBy: string;
}

```

```

    createdAt: string;
    updatedBy?: string;
    updatedAt?: string;
    deletedBy?: string;
  }

export const ClothSchema = new mongoose.Schema<IClothSchema>(
  {
    id: { type: Number, required: true, unique: true },
    name: { type: String, required: true },
    price: { type: Number, required: true },
    desc: { type: String, required: true },
    availableSizes: { type: [String], required: true },
    createdBy: { type: String, required: true },
    createdAt: { type: String, required: true },
    updatedBy: { type: String, required: false },
    updatedAt: { type: String, required: false },
    deletedBy: { type: String, required: false },
  },
  {
    timestamps: false,
  },
);

```

delivery.schema.ts

```

import * as mongoose from 'mongoose';
import { Document } from 'mongoose';
import { ClothSizes } from '../../shared/enums/cloth-sizes.enum';
import { DeliveryType } from '../../shared/enums/delivery-type.enum';

export const DELIVERY_MODEL_SCHEMA = Symbol('DELIVERY_MODEL_SCHEMA');

export interface IDeliverySizeCount {
  size: ClothSizes;
  count: number;
}

export interface IClothDelivered {
  sizes: IDeliverySizeCount[];
  clothId: number;
}

export interface IDeliverySchema extends Document {
  id: number;
  deliveredTo: number;
  deliveredFrom?: number;
  totalAmountDelivered: number;
  typeOfDelivery: DeliveryType;
  price?: number;
  clothDelivered: IClothDelivered[];
  createdBy: string;
  createdAt: string;
  updatedBy?: string;
  updatedAt?: string;
  deletedBy?: string;
}

export const DeliverySchema = new mongoose.Schema<IDeliverySchema>(
  {
    id: { type: Number, required: true, unique: true },
    deliveredTo: { type: Number, required: true },
    deliveredFrom: { type: Number, required: false },
    totalAmountDelivered: { type: Number, required: true },
    typeOfDelivery: { type: String, required: true },
    price: { type: Number, required: false },
    clothDelivered: { type: [Object], required: true },
    createdBy: { type: String, required: true },
    createdAt: { type: String, required: true },
    updatedBy: { type: String, required: false },
    updatedAt: { type: String, required: false },
    deletedBy: { type: String, required: false },
  },
  {
    timestamps: false,
  },
);

```

```
);
```

order.schema.ts

```
import * as mongoose from 'mongoose';
import { Document } from 'mongoose';
import { ClothSizes } from '../../shared/enums/cloth-sizes.enum';
import { OrderStatuses } from '../../shared/enums/order-statuses.enum';

export const ORDER_MODEL_SCHEMA = Symbol('ORDER_MODEL_SCHEMA');

export interface IClothId {
  clothId: number;
  amount: number;
  size: ClothSizes;
  storeId?: number;
}

export interface IOrderSchema extends Document {
  id: number;
  clientId: number;
  clothIdList: IClothId[];
  status: OrderStatuses;
  price: number;
  createdBy: string;
  createdAt: string;
  updatedBy?: string;
  updatedAt?: string;
  deletedBy?: string;
}

export const OrderSchema = new mongoose.Schema<IOrderSchema>(
  {
    id: { type: Number, required: true, unique: true },
    clientId: { type: Number, required: true },
    clothIdList: { type: [Object], required: true },
    status: { type: String, required: true },
    price: { type: Number, required: true },
    createdBy: { type: String, required: true },
    createdAt: { type: String, required: true },
    updatedBy: { type: String, required: false },
    updatedAt: { type: String, required: false },
    deletedBy: { type: String, required: false },
  },
  {
    timestamps: false,
  },
);
```

staff.schema.ts

```
import * as mongoose from 'mongoose';
import { Document } from 'mongoose';

export const STAFF_MODEL_SCHEMA = Symbol('STAFF_MODEL_SCHEMA');

export interface IStaffSchema extends Document {
  id: number;
  name: string;
  email: string;
  isAdmin: boolean;
  password?: string;
  position: string;
  storeId: number;
  isHidden: boolean;
  createdBy: string;
  createdAt: string;
  updatedBy?: string;
  updatedAt?: string;
  deletedBy?: string;
}

export const StaffSchema = new mongoose.Schema<IStaffSchema>(
  {
    id: { type: Number, required: true, unique: true },
    name: { type: String, required: true },
```

```

    email: { type: String, required: true, unique: true },
    isAdmin: { type: Boolean, required: true },
    password: { type: String, required: false },
    position: { type: String, required: true },
    storeId: { type: Number, required: true },
    isHidden: { type: Boolean, required: true },
    createdBy: { type: String, required: true },
    createdAt: { type: String, required: true },
    updatedBy: { type: String, required: false },
    updatedAt: { type: String, required: false },
    deletedBy: { type: String, required: false },
  },
  {
    timestamps: false,
  },
);

```

store.schema.ts

```

import * as mongoose from 'mongoose';
import { Document } from 'mongoose';

export const STORE_MODEL_SCHEMA = Symbol('STORE_MODEL_SCHEMA');

export interface IStoreSchema extends Document {
  id: number;
  address: string;
  isActive: boolean;
  createdBy: string;
  createdAt: string;
  updatedBy?: string;
  updatedAt?: string;
  deletedBy?: string;
}

export const StoreSchema = new mongoose.Schema<IStoreSchema>(
  {
    id: { type: Number, required: true, unique: true },
    address: { type: String, required: true, unique: true },
    isActive: { type: Boolean, required: true },
    createdBy: { type: String, required: true },
    createdAt: { type: String, required: true },
    updatedBy: { type: String, required: false },
    updatedAt: { type: String, required: false },
    deletedBy: { type: String, required: false },
  },
  {
    timestamps: false,
  },
);

```

warehouse.schema.ts

```

import * as mongoose from 'mongoose';
import { Document } from 'mongoose';
import { ClothSizes } from '../../shared/enums/cloth-sizes.enum';

export const WAREHOUSE_MODEL_SCHEMA = Symbol('WAREHOUSE_MODEL_SCHEMA');

export interface IWarehouseSchema extends Document {
  id: number;
  clothId: number;
  storeId: number;
  size: ClothSizes;
  amount: number;
  createdBy: string;
  createdAt: string;
  updatedBy?: string;
  updatedAt?: string;
  deletedBy?: string;
}

export const WarehouseSchema = new mongoose.Schema<IIWarehouseSchema>(
  {
    id: { type: Number, required: true, unique: true },
    clothId: { type: Number, required: true },

```

```

    storeId: { type: Number, required: true },
    amount: { type: Number, required: true },
    size: { type: String, required: true },
    createdBy: { type: String, required: true },
    createdAt: { type: String, required: true },
    updatedBy: { type: String, required: false },
    updatedAt: { type: String, required: false },
    deletedBy: { type: String, required: false },
  },
  {
    timestamps: false,
  },
);

```

order.service.ts

```

import {
  Inject,
  Injectable,
  Logger,
  NotFoundException,
  BadGatewayException,
  BadRequestException,
} from '@nestjs/common';
import { Model } from 'mongoose';
import {
  ClothId,
  OrderCreateRequestDto,
} from '../order/dto/order-create.request.dto';
import { GetOrderRequest } from '../order/dto/order-get.request.dto';
import { GetOrderResponse } from '../order/dto/order-get.response.dto';
import { OrderUpdateRequestDto } from '../order/dto/order-update.request.dto';
import {
  ORDER_MODEL_SCHEMA,
  IOrderSchema,
  IClothId,
} from '../mongodb/schemas/order.schema';
import { EnumSort } from '../../shared/enums/sort.enum';
import {
  CLOTH_MODEL_SCHEMA,
  IClothSchema,
} from '../mongodb/schemas/cloth.schema';
import {
  IWarehouseSchema,
  WAREHOUSE_MODEL_SCHEMA,
} from '../mongodb/schemas/warehouse.schema';
import { OrderStatuses } from '../../shared/enums/order-statuses.enum';

/**
 * Service that provides order management
 * @param {ConfigService} configService - service to interact with env and settings
 * @param {JwtService} jwtService - service to interact with jwt tokens
 */
@Injectable()
export class OrderService {
  private logger = new Logger(OrderService.name);
  constructor(
    @Inject(ORDER_MODEL_SCHEMA)
    private readonly orderSchema: Model<IOrderSchema>,
    @Inject(CLOTH_MODEL_SCHEMA)
    private readonly clothSchema: Model<IClothSchema>,
    @Inject(WAREHOUSE_MODEL_SCHEMA)
    private readonly warehouseSchema: Model<IWarehouseSchema>,
  ) {}

  async createOrder(
    dto: OrderCreateRequestDto,
    userEmail: string,
  ): Promise<IOrderSchema> {
    try {
      if (dto.status === OrderStatuses.RETURNED) {
        throw new BadRequestException(
          'Не можливо створити замовлення з статусом RETURNED',
        );
      }
    }
  }

```

```

let clothIdList: IClothId[];

if (
  [
    OrderStatuses.COMPLETED,
    OrderStatuses.DELIVERED,
    OrderStatuses.SENT,
  ].includes(dto.status)
) {
  const warehouseObjects = await Promise.all(
    dto.clothIdList.map(async (clothId) => {
      const warehouse = await this.warehouseSchema.findOne({
        clothId: clothId.clothId,
        amount: { $gte: clothId.amount },
        size: clothId.size,
        deletedBy: null,
      });

      if (!warehouse) {
        throw new BadRequestException(
          `Товару з ID ${clothId.clothId} не має в достатній кількості на складах.`
        );
      }

      return { clothId, warehouse };
    }),
  );

  clothIdList = await Promise.all(
    warehouseObjects.map(async (obj) => {
      await obj.warehouse.updateOne({
        id: obj.warehouse.id,
        amount: obj.warehouse.amount - obj.clothId.amount,
        updatedAt: new Date().toISOString(),
        updatedBy: userEmail,
      });

      return {
        ...obj.clothId,
        storeId: obj.warehouse.storeId,
      };
    }),
  );
}

const lastOrder = await this.orderSchema
  .find(null, null, { sort: { id: -1 } })
  .limit(1);

const price = await this.getPriceByClothIds(dto.clothIdList);

return this.orderSchema.create({
  ...dto,
  clothIdList: clothIdList || dto.clothIdList,
  id: (lastOrder[0]?.id || 0) + 1,
  price,
  createdAt: new Date().toISOString(),
  createdBy: userEmail,
});
} catch (e) {
  this.logger.error(e);
  throw e;
}
}

async getOrder(dto: GetOrderRequest): Promise<GetOrderResponse> {
  try {
    const { limit, page, order, orderBy, ...filter } = dto;
    const skip = limit * page;

    const orders = await this.orderSchema
      .find(
        {
          deletedBy: null,
          ...filter,
        },
        null,
      )
  }
}

```

```

    { sort: { [orderBy || 'id']: order === EnumSort.desc ? -1 : 1 } },
  )
  .limit(limit)
  .skip(skip);

const orderCount = await this.orderSchema.count({
  deletedBy: null,
  ...filter,
});

return {
  order: orders,
  orderCount,
};
} catch (e) {
  this.logger.error(e);
  throw e;
}
}

async updateOrder(
  dto: OrderUpdateRequestDto,
  userEmail: string,
): Promise<IOrderSchema> {
  try {
    const { id, ...updateData } = dto;
    const existingOrder = await this.orderSchema.findOne({
      id: dto.id,
      deletedBy: null,
    });
    if (!existingOrder) {
      throw new NotFoundException(`Не знайдено магазин з ID ${id}`);
    }

    let clothIdList: IClothId[];

    const OrderStatusesToTakeClothFromWarehouse = [
      OrderStatuses.COMPLETED,
      OrderStatuses.DELIVERED,
      OrderStatuses.SENT,
    ];

    if (
      OrderStatusesToTakeClothFromWarehouse.includes(existingOrder.status) &&
      (dto.clothIdList ||
        (dto.status &&
          dto.status !== OrderStatuses.RETURNED &&
          !OrderStatusesToTakeClothFromWarehouse.includes(dto.status)))
    ) {
      throw new BadRequestException(
        'Не можливо змінити товари в замовленні або статус замовлення. Замовлення вже було відправлене.',
      );
    }
    if (
      dto.status === OrderStatuses.RETURNED &&
      !OrderStatusesToTakeClothFromWarehouse.includes(existingOrder.status)
    ) {
      throw new BadRequestException(
        'Не можливо змінити статус замовлення на RETURNED. Замовлення не було відправлене.',
      );
    }

    if (
      OrderStatusesToTakeClothFromWarehouse.includes(dto.status) &&
      !OrderStatusesToTakeClothFromWarehouse.includes(existingOrder.status)
    ) {
      const warehouseObjects = await Promise.all(
        (dto.clothIdList || existingOrder.clothIdList).map(
          async (clothId) => {
            const warehouse = await this.warehouseSchema.findOne({
              clothId: clothId.clothId,
              amount: { $gte: clothId.amount },
              size: clothId.size,
              deletedBy: null,
            });

            if (!warehouse) {

```

```

        throw new BadRequestException(
            `Товару з ID ${clothId.clothId} та розміром ${clothId.size} не має в достатній кількості на складах.`);
    });

    return { clothId, warehouse };
  },
);

clothIdList = await Promise.all(
  warehouseObjects.map(async (obj) => {
    await obj.warehouse.updateOne({
      id: obj.warehouse.id,
      amount: obj.warehouse.amount - obj.clothId.amount,
      updatedAt: new Date().toISOString(),
      updatedBy: userEmail,
    });

    return {
      ...obj.clothId,
      storeId: obj.warehouse.storeId,
    };
  }),
);

if (dto.status === OrderStatuses.RETURNED) {
  const warehouseObjects = await Promise.all(
    existingOrder.clothIdList.map(async (clothId) => {
      const warehouse = await this.warehouseSchema.findOne({
        clothId: clothId.clothId,
        storeId: clothId.storeId,
        size: clothId.size,
        deletedBy: null,
      });

      if (!warehouse) {
        throw new BadRequestException(
          `Товару з ID ${clothId.clothId} не існує у магазині звідки його доставляли.`);
      }

      return { clothId, warehouse };
    }),
  );

  clothIdList = await Promise.all(
    warehouseObjects.map(async (obj) => {
      await obj.warehouse.updateOne({
        id: obj.warehouse.id,
        amount: obj.warehouse.amount + obj.clothId.amount,
        updatedAt: new Date().toISOString(),
        updatedBy: userEmail,
      });

      return {
        ...obj.clothId,
      };
    }),
  );
}

let price;
if (dto.clothIdList) {
  price = await this.getPriceByClothIds(dto.clothIdList);
}

const updateResult = await existingOrder.updateOne({
  ...updateData,
  price,
  clothIdList: clothIdList || dto.clothIdList,
  updatedAt: new Date().toISOString(),
  updatedBy: userEmail,
});

```

```

    if (updateResult.modifiedCount === 0) {
      throw new BadGatewayException('Помилка під час оновлення');
    }

    return this.orderSchema.findOne({
      id: dto.id,
      deletedBy: null,
    });
  } catch (e) {
    this.logger.error(e);
    throw e;
  }
}

async softDeleteOrder(id: number, userEmail: string): Promise<number> {
  try {
    const existingOrder = await this.orderSchema.findOne({
      id,
      deletedBy: null,
    });
    if (!existingOrder) {
      throw new NotFoundException(`Не знайдено магазин з ID ${id}`);
    }

    const updateResult = await existingOrder.updateOne({
      updatedAt: new Date().toISOString(),
      deletedBy: userEmail,
    });

    if (updateResult.modifiedCount === 0) {
      throw new BadGatewayException('Помилка під час оновлення');
    }

    return id;
  } catch (e) {
    this.logger.error(e);
    throw e;
  }
}

private async getPriceByClothIds(clothIdList: ClothId[]) {
  const clothIds = clothIdList.map((clothId) => clothId.clothId);

  const clothes = await this.clothSchema.find({
    id: { $in: clothIds },
  });

  const clothPriceByClothId = clothes.reduce((prev, cur) => {
    prev[cur.id] = cur.price;
    return prev;
  }, {} as Record<string, any>);

  const clothAmountByClothId = clothIdList.reduce((prev, cur) => {
    prev[cur.clothId] = cur.amount;
    return prev;
  }, {} as Record<string, any>);
  return clothIds.reduce((prev, cur) => {
    if (!clothPriceByClothId[cur]) {
      throw new NotFoundException(`Не знайдено товару з ID ${cur}`);
    }
    return prev + clothPriceByClothId[cur] * clothAmountByClothId[cur];
  }, 0);
}
}

```

reports.service.ts

```

import { Inject, Injectable, Logger } from '@nestjs/common';
import { Model } from 'mongoose';
import {
  IOrderSchema,
  ORDER_MODEL_SCHEMA,
} from '../mongodb/schemas/order.schema';
import {
  STAFF_MODEL_SCHEMA,
  IStaffSchema,

```

```

} from '../mongodb/schemas/staff.schema';
import {
  ClothBySales,
  GetReportResponse,
  HalfYearIncome,
  IncomeByMonth,
  SizesPopularityByMonth,
  StaffChanges,
  StaffChangesByMonth,
  StoreBySales,
} from '../reports/dto/reports-get.response.dto';
import { ClothSizes } from '../../shared/enums/cloth-sizes.enum';
import { OrderStatuses } from '../../shared/enums/order-statuses.enum';

/**
 * Service that provides report management
 * @param {ConfigService} configService - service to interact with env and settings
 * @param {JwtService} jwtService - service to interact with jwt tokens
 */
@Injectable()
export class ReportService {
  private logger = new Logger(ReportService.name);
  constructor(
    @Inject(ORDER_MODEL_SCHEMA)
    private readonly orderSchema: Model<IOrderSchema>,
    @Inject(STAFF_MODEL_SCHEMA)
    private readonly staffSchema: Model<IStaffSchema>,
  ) {}

  async getReport(): Promise<GetReportResponse> {
    try {
      const currentHalfYearDate = this.getPrevHalfYearDate();
      const previousHalfYearDate =
        this.getPrevHalfYearDate(currentHalfYearDate);

      const currentHalfYearOrders = await this.orderSchema.find({
        createdAt: { $gt: currentHalfYearDate },
        deletedBy: null,
        status: OrderStatuses.COMPLETED,
      });
      const previousHalfYearOrders = await this.orderSchema.find({
        createdAt: {
          $gt: previousHalfYearDate,
          $lt: currentHalfYearDate,
        },
        deletedBy: null,
        status: OrderStatuses.COMPLETED,
      });

      const clothesBySales = await this.getClothesSales(
        currentHalfYearOrders,
        previousHalfYearOrders,
      );
      const halfYearIncome = await this.getHalfYearIncome(
        currentHalfYearOrders,
      );
      const sizePopularity = await this.getSizesPopularityByMonth(
        currentHalfYearOrders,
        previousHalfYearOrders,
      );
      const staffChanges = await this.getStaffChanges(currentHalfYearDate);
      const storesBySales = await this.getStoreBySales(
        currentHalfYearOrders,
        previousHalfYearOrders,
      );

      return {
        clothesBySales,
        halfYearIncome,
        sizePopularity,
        staffChanges,
        storesBySales,
      };
    } catch (e) {
      this.logger.error(e);
      throw e;
    }
  }
}

```

```

}

private async getClothesSales(
  currentHalfYearOrders: IOrderSchema[],
  previousHalfYearOrders: IOrderSchema[],
): Promise<ClothBySales[]> {
  const currentHalfYearSales = currentHalfYearOrders.reduce((prev, cur) => {
    cur.clothIdList.forEach((clothId) => {
      if (prev[clothId.clothId]) {
        prev[clothId.clothId] += clothId.amount;
      } else {
        prev[clothId.clothId] = clothId.amount;
      }
    });
    return prev;
  }, {} as Record<string, any>);

  const previousHalfYearSales = previousHalfYearOrders.reduce((prev, cur) => {
    cur.clothIdList.forEach((clothId) => {
      if (prev[clothId.clothId]) {
        prev[clothId.clothId] += clothId.amount;
      } else {
        prev[clothId.clothId] = clothId.amount;
      }
    });
    return prev;
  }, {} as Record<string, any>);

  const clothSales = Object.keys(currentHalfYearSales)
    .map((clothId) => ({
      clothId: parseInt(clothId),
      numberOfSales: currentHalfYearSales[clothId],
      change: previousHalfYearSales[clothId]
        ? currentHalfYearSales[clothId] - previousHalfYearSales[clothId]
        : 0,
    })))
    .sort((a, b) => b.numberOfSales - a.numberOfSales);
  return clothSales.length > 5 ? clothSales.slice(0, 5) : clothSales;
}

private async getHalfYearIncome(
  halfYearOrders: IOrderSchema[],
): Promise<HalfYearIncome> {
  const currentMonth = new Date().getMonth() + 1;
  const months = Array(6)
    .fill(0)
    .map((val, i) => currentMonth - i)
    .reverse();

  const incomeByMonth = months.map((monthNumber) => ({
    monthNumber,
    income: 0,
  }))) as IncomeByMonth[];

  let totalIncome = 0;
  halfYearOrders.forEach((cur) => {
    const month = new Date(cur.createdAt).getMonth() + 1;

    const incomeByMonthEntry = incomeByMonth.find(
      (val) => val.monthNumber === month,
    );
    incomeByMonthEntry.income += cur.price;

    totalIncome += cur.price;
  });

  return {
    incomeByMonth,
    totalIncome,
  };
}

private async getSizesPopularityByMonth(
  currentHalfYearOrders: IOrderSchema[],
  previousHalfYearOrders: IOrderSchema[],
): Promise<SizesPopularityByMonth[]> {
  const currentHalfYearSales = currentHalfYearOrders.reduce((prev, cur) => {

```

```

    cur.clothIdList.forEach((clothId) => {
      if (prev[clothId.size]) {
        prev[clothId.size] += clothId.amount;
      } else {
        prev[clothId.size] = clothId.amount;
      }
    });
    return prev;
  }, {}) as Record<ClothSizes, any>;

const previousHalfYearSales = previousHalfYearOrders.reduce((prev, cur) => {
  cur.clothIdList.forEach((clothId) => {
    if (prev[clothId.size]) {
      prev[clothId.size] += clothId.amount;
    } else {
      prev[clothId.size] = clothId.amount;
    }
  });
  return prev;
}, {}) as Record<ClothSizes, any>;

const sizePopularity = Object.keys(currentHalfYearSales)
  .map((size: ClothSizes) => ({
    size: size,
    numberOfSales: currentHalfYearSales[size],
    change: previousHalfYearSales[size]
      ? currentHalfYearSales[size] - previousHalfYearSales[size]
      : 0,
  })))
  .sort((a, b) => b.numberOfSales - a.numberOfSales);
return sizePopularity.length > 5
  ? sizePopularity.slice(0, 5)
  : sizePopularity;
}

private async getStaffChanges(
  currentHalfYearDate: string,
): Promise<StaffChanges> {
  const firedStaffHalfYear = await this.staffSchema.find({
    updatedAt: {
      $gt: currentHalfYearDate,
    },
    isHidden: false,
    deletedBy: { $ne: null },
  });
  const hiredStaffHalfYear = await this.staffSchema.find({
    createdAt: {
      $gt: currentHalfYearDate,
    },
    isHidden: false,
  });

  const totalStaffNumber = await this.staffSchema.count({
    deletedBy: null,
    isHidden: false,
  });

  const currentMonth = new Date().getMonth() + 1;
  const months = Array(6)
    .fill(0)
    .map((val, i) => currentMonth - i)
    .reverse();

  const staffChangesByMonth = months.map((monthNumber) => ({
    monthNumber,
    firedStaffCount: 0,
    hiredStaffCount: 0,
  })) as StaffChangesByMonth[];

  firedStaffHalfYear.forEach((cur) => {
    const month = new Date(cur.updatedAt).getMonth() + 1;

    const staffChanges = staffChangesByMonth.find(
      (val) => val.monthNumber === month,
    );

    if (!staffChanges) return;

```

```

        staffChanges.firedStaffCount += 1;
    });

    hiredStaffHalfYear.forEach((cur) => {
        const month = new Date(cur.createdAt).getMonth() + 1;

        const staffChanges = staffChangesByMonth.find(
            (val) => val.monthNumber === month,
        );

        if (!staffChanges) return;

        staffChanges.hiredStaffCount += 1;
    });

    return {
        staffChangesByMonth,
        totalStaffNumber,
    };
}

private async getStoreBySales(
    currentHalfYearOrders: IOrderSchema[],
    previousHalfYearOrders: IOrderSchema[],
): Promise<StoreBySales[]> {
    const currentHalfYearSales = currentHalfYearOrders.reduce((prev, cur) => {
        cur.clothIdList.forEach((clothId) => {
            if (prev[clothId.storeId]) {
                prev[clothId.storeId] += clothId.amount;
            } else {
                prev[clothId.storeId] = clothId.amount;
            }
        });
        return prev;
    }, {} as Record<number, any>);

    const previousHalfYearSales = previousHalfYearOrders.reduce((prev, cur) => {
        cur.clothIdList.forEach((clothId) => {
            if (prev[clothId.storeId]) {
                prev[clothId.storeId] += clothId.amount;
            } else {
                prev[clothId.storeId] = clothId.amount;
            }
        });
        return prev;
    }, {} as Record<number, any>);

    const storesBySales = Object.keys(currentHalfYearSales)
        .map((storeId) => ({
            storeId: parseInt(storeId),
            numberOfSales: currentHalfYearSales[storeId],
            change: previousHalfYearSales[storeId]
                ? currentHalfYearSales[storeId] - previousHalfYearSales[storeId]
                : 0,
        })))
        .sort((a, b) => b.numberOfSales - a.numberOfSales);
    return storesBySales.length > 5 ? storesBySales.slice(0, 5) : storesBySales;
}

private getPrevHalfYearDate(currentDate = new Date().toISOString()) {
    const date = new Date(currentDate);
    date.setMonth(date.getMonth() - 5, 1);
    date.setHours(0, 0, 0, 0);
    return date.toISOString();
}
}

```

staff.config.ts

```

import { Logger } from '@nestjs/common';
import { registerAs } from '@nestjs/config';
import * as Joi from 'joi';
import * as path from 'path';

export const cfgRaw = {

```

```

    nameMinLength: 2,
    emailMinLength: 2,
    passwordMinLength: 2,
    positionMinLength: 2,
    nameMaxLength: 80,
    emailMaxLength: 80,
    passwordMaxLength: 80,
    positionMaxLength: 80,
  } as const;

Logger.verbose('Validating ' + path.basename(__filename));
Joi.attempt(
  cfgRaw,
  Joi.object({
    nameMinLength: Joi.number().required(),
    emailMinLength: Joi.number().required(),
    passwordMinLength: Joi.number().required(),
    positionMinLength: Joi.number().required(),
    nameMaxLength: Joi.number().required(),
    emailMaxLength: Joi.number().required(),
    passwordMaxLength: Joi.number().required(),
    positionMaxLength: Joi.number().required(),
  }).required(),
);

export const config = registerAs('module.staff', () => cfgRaw);

```

staff.service.ts

```

import {
  Inject,
  Injectable,
  Logger,
  BadRequestException,
  NotFoundException,
  BadGatewayException,
} from '@nestjsjs/common';
import { Model } from 'mongoose';
import {
  STAFF_MODEL_SCHEMA,
  IStaffSchema,
} from '../mongodb/schemas/staff.schema';
import { StaffCreateRequestDto } from '../staff/dto/staff-create.request.dto';
import { GetStaffRequest } from '../staff/dto/staff-get.request.dto';
import { GetStaffResponse } from '../staff/dto/staff-get.response.dto';
import { StaffUpdateRequestDto } from '../staff/dto/staff-update.request.dto';
import { EnumSort } from '../shared/enums/sort.enum';

/**
 * Service that provides staff management
 * @param {ConfigService} configService - service to interact with env and settings
 * @param {JwtService} jwtService - service to interact with jwt tokens
 */
@Injectable()
export class StaffService {
  private logger = new Logger(StaffService.name);
  constructor(
    @Inject(STAFF_MODEL_SCHEMA)
    private readonly staffSchema: Model<IStaffSchema>,
  ) {}

  async createStaff(
    dto: StaffCreateRequestDto,
    userEmail: string,
  ): Promise<IStaffSchema> {
    try {
      const existingEmail = await this.staffSchema.findOne({
        email: dto.email,
      });

      if (existingEmail) {
        throw new BadRequestException('Користувач з такою поштою вже існує');
      }

      const lastStaff = await this.staffSchema
        .find(null, null, { sort: { id: -1 } })

```

```

        .limit(1);

    return this.staffSchema.create({
        ...dto,
        id: (lastStaff[0]?.id || 0) + 1,
        createdAt: new Date().toISOString(),
        createdBy: userEmail,
        isHidden: false,
    });
} catch (e) {
    this.logger.error(e);
    throw e;
}
}

async getStaff(dto: GetStaffRequest): Promise<GetStaffResponse> {
    try {
        const { limit, page, order, orderBy, email, name, position, ...filter } =
            dto;
        const skip = limit * page;

        const aggregateFilter = {} as Record<string, any>;

        if (email) {
            aggregateFilter.email = { $regex: email, $options: 'i' };
        }

        if (name) {
            aggregateFilter.name = { $regex: name, $options: 'i' };
        }

        if (position) {
            aggregateFilter.position = { $regex: position, $options: 'i' };
        }

        const staff = await this.staffSchema
            .find(
                {
                    deletedBy: null,
                    isHidden: false,
                    ...filter,
                    ...aggregateFilter,
                },
                null,
                { sort: { [orderBy || 'id']: order === EnumSort.desc ? -1 : 1 } },
            )
            .limit(limit)
            .skip(skip);

        const staffCount = await this.staffSchema.count({
            deletedBy: null,
            isHidden: false,
            ...filter,
            ...aggregateFilter,
        });

        return {
            staff,
            staffCount,
        };
    } catch (e) {
        this.logger.error(e);
        throw e;
    }
}

async updateStaff(
    dto: StaffUpdateRequestDto,
    userEmail: string,
): Promise<IStaffSchema> {
    try {
        const { id, ...updateData } = dto;
        const existingStaff = await this.staffSchema.findOne({
            id: dto.id,
            deletedBy: null,
        });
        if (!existingStaff) {

```

```

    throw new NotFoundException(`Не знайдено користувача з ID ${id}`);
  }

  const updateResult = await existingStaff.updateOne({
    ...updateData,
    updatedAt: new Date().toISOString(),
    updatedBy: userEmail,
  });

  if (updateResult.modifiedCount === 0) {
    throw new BadGatewayException('Помилка під час оновлення');
  }

  return this.staffSchema.findOne({
    id: dto.id,
    deletedBy: null,
  });
} catch (e) {
  this.logger.error(e);
  throw e;
}
}

async softDeleteStaff(id: number, userEmail: string): Promise<number> {
  try {
    const existingStaff = await this.staffSchema.findOne({
      id,
      deletedBy: null,
    });
    if (!existingStaff) {
      throw new NotFoundException(`Не знайдено користувача з ID ${id}`);
    }

    const updateResult = await existingStaff.updateOne({
      updatedAt: new Date().toISOString(),
      deletedBy: userEmail,
    });

    if (updateResult.modifiedCount === 0) {
      throw new BadGatewayException('Помилка під час оновлення');
    }

    return id;
  } catch (e) {
    this.logger.error(e);
    throw e;
  }
}
}

```

store.config.ts

```

import { Logger } from '@nestjsjs/common';
import { registerAs } from '@nestjsjs/config';
import * as Joi from 'joi';
import * as path from 'path';

export const cfgRaw = {
  addressMinLength: 2,
  addressMaxLength: 80,
} as const;

Logger.verbose('Validating ' + path.basename(__filename));
Joi.attempt(
  cfgRaw,
  Joi.object({
    addressMinLength: Joi.number().required(),
    addressMaxLength: Joi.number().required(),
  }).required(),
);

export const config = registerAs('module.store', () => cfgRaw);

```

store.service.ts

```

import {

```

```

Inject,
Injectable,
Logger,
BadRequestException,
NotFoundException,
BadGatewayException,
} from '@nestjs/common';
import { Model } from 'mongoose';
import { StoreCreateRequestDto } from '../store/dto/store-create.request.dto';
import { GetStoreRequest } from '../store/dto/store-get.request.dto';
import { GetStoreResponse } from '../store/dto/store-get.response.dto';
import { StoreUpdateRequestDto } from '../store/dto/store-update.request.dto';
import {
  STORE_MODEL_SCHEMA,
  IStoreSchema,
} from '../mongodb/schemas/store.schema';
import { EnumSort } from '../../shared/enums/sort.enum';

/**
 * Service that provides store management
 * @param {ConfigService} configService - service to interact with env and settings
 * @param {JwtService} jwtService - service to interact with jwt tokens
 */
@Injectable()
export class StoreService {
  private logger = new Logger(StoreService.name);
  constructor(
    @Inject(STORE_MODEL_SCHEMA)
    private readonly storeSchema: Model<IStoreSchema>,
  ) {}

  async createStore(
    dto: StoreCreateRequestDto,
    userEmail: string,
  ): Promise<IStoreSchema> {
    try {
      const existingStore = await this.storeSchema.findOne({
        address: dto.address,
      });

      if (existingStore) {
        throw new BadRequestException('Магазин з такою адресою вже існує');
      }

      const lastStore = await this.storeSchema
        .find(null, null, { sort: { id: -1 } })
        .limit(1);

      return this.storeSchema.create({
        ...dto,
        id: (lastStore[0]?.id || 0) + 1,
        createdAt: new Date().toISOString(),
        createdBy: userEmail,
      });
    } catch (e) {
      this.logger.error(e);
      throw e;
    }
  }

  async getStore(dto: GetStoreRequest): Promise<GetStoreResponse> {
    try {
      const { limit, page, order, orderBy, address, ...filter } = dto;
      const skip = limit * page;

      const aggregateFilter = {} as Record<string, any>;

      if (address) {
        aggregateFilter.address = { $regex: address, $options: 'i' };
      }

      const store = await this.storeSchema
        .find(
          {
            deletedBy: null,
            ...filter,
            ...aggregateFilter,
          },
        )

```

```

        },
        null,
        { sort: { [orderBy || 'id']: order === EnumSort.desc ? -1 : 1 } },
    )
    .limit(limit)
    .skip(skip);

const storeCount = await this.storeSchema.count({
    deletedBy: null,
    ...filter,
    ...aggregateFilter,
});

return {
    store,
    storeCount,
};
} catch (e) {
    this.logger.error(e);
    throw e;
}
}

async getStoreById(id: number): Promise<IStoreSchema> {
    return this.storeSchema.findOne({ id, deletedBy: null });
}

async updateStore(
    dto: StoreUpdateRequestDto,
    userEmail: string,
): Promise<IStoreSchema> {
    try {
        const { id, ...updateData } = dto;
        const existingStore = await this.storeSchema.findOne({
            id: dto.id,
            deletedBy: null,
        });
        if (!existingStore) {
            throw new NotFoundException(`Не знайдено магазин з ID ${id}`);
        }

        const updateResult = await existingStore.updateOne({
            ...updateData,
            updatedAt: new Date().toISOString(),
            updatedBy: userEmail,
        });

        if (updateResult.modifiedCount === 0) {
            throw new BadGatewayException('Помилка під час оновлення');
        }

        return this.storeSchema.findOne({
            id: dto.id,
            deletedBy: null,
        });
    } catch (e) {
        this.logger.error(e);
        throw e;
    }
}

async softDeleteStore(id: number, userEmail: string): Promise<number> {
    try {
        const existingStore = await this.storeSchema.findOne({
            id,
            deletedBy: null,
        });
        if (!existingStore) {
            throw new NotFoundException(`Не знайдено магазин з ID ${id}`);
        }

        const updateResult = await existingStore.updateOne({
            updatedAt: new Date().toISOString(),
            deletedBy: userEmail,
        });

        if (updateResult.modifiedCount === 0) {

```

```

        throw new BadGatewayException('Помилка під час оновлення');
    }

    return id;
} catch (e) {
    this.logger.error(e);
    throw e;
}
}
}

```

cloth-sizes.enum.ts

```

export enum ClothSizes {
    XS = 'XS',
    S = 'S',
    M = 'M',
    L = 'L',
    XL = 'XL',
    XXL = 'XXL',
}

```

delivery-type.enum.ts

```

export enum DeliveryType {
    INTERNAL = 'INTERNAL',
    EXTERNAL = 'EXTERNAL',
}

```

order-statuses.enum.ts

```

export enum OrderStatuses {
    CREATED = 'CREATED',
    PROCESSING = 'PROCESSING',
    PAYED = 'PAYED',
    SENT = 'SENT',
    DELIVERED = 'DELIVERED',
    COMPLETED = 'COMPLETED',
    RETURNED = 'RETURNED',
    CANCELED = 'CANCELED',
}

```

sort.enum.ts

```

export enum EnumSort {
    asc = 'asc',
    desc = 'desc',
}

```

main.ts

```

import { AppModule } from './modules/app/app.module';
import cookieParser from 'cookie-parser';
import { ValidationPipe } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { NestFactory } from '@nestjs/core';
import { HttpExceptionsFilter } from './shared/filters/http-exceptions.filter';
import { RequestLoggerInterceptor } from './shared/interceptors/request-logger.interceptor';

async function bootstrap() {
    const app = await NestFactory.create(AppModule);

    const configService = app.get<ConfigService>(ConfigService);

    app.use(cookieParser(configService.getOrThrow('module.app.cookieSecret')));

    app.useGlobalInterceptors(new RequestLoggerInterceptor());

    app.enableCors({
        origin: true,
        credentials: true,
    });

    app.useGlobalFilters(new HttpExceptionsFilter());
}

```

```
app.useGlobalPipes(  
  new ValidationPipe({  
    disableErrorMessage: false,  
    dismissDefaultMessages: false,  
    transform: true,  
  })),  
);  
  
await app.listen(3001);  
}  
bootstrap();
```

ДОДАТОК Б
Слайди презентації

Національний університет “Запорізька політехніка”
Кафедра програмних засобів

Дипломна кваліфікаційна робота магістра на тему:

**ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
СИСТЕМИ ДЛЯ УПРАВЛІННЯ МЕРЕЖЕЮ
МАГАЗИНІВ ОДЯГУ**

Виконав:
студ. гр КНТ-122м

Ілля ШЕВЕРДІНКІН

Керівник:
к.т.н, доцент

Тетяна КАПЛІЄНКО

Запоріжжя
2023

Рисунок Б.1 – Слайд 1

Постановка задачі

Об’єкт дослідження – управління взаємовідносин з клієнтами з використанням спеціалізованих систем (Customer Relationship Management або CRM).

Предмет дослідження – розробка вебсайтів CRM.

Мета роботи – створення вебсайту CRM для мережі магазинів одягу.

Рисунок Б.2 – Слайд 2

Існуючі рішення

Назва	Переваги	Недоліки
Сайт Clickup	Дозволяє: 1.Створювати, призначати та відстежувати завдання для окремих осіб або команд; 2.Контролювати час, витрачений на завдання та проекти; 3.Сприяти командній роботі та комунікації всередині платформ; 4.Ставити завдання та відстежувати прогрес у досягненні конкретних цілей.	Немає можливості управління персоналом, клієнтами, замовленнями, продажами та складом магазинів.
Сайт Zendesk	Дозволяє: 1.Візуалізувати та управляти можливостями продажів на різних етапах; 2.Організовувати та відстежувати дії клієнтів; 3.Отримувати інформацію про ефективність та тенденції продажів; 4.Підключатися до електронної пошти для відстеження комунікацій; 5.Має можливість отримати доступ до функцій CRM на ходу за допомогою мобільного застосунку.	Немає можливості управління персоналом, клієнтами, замовленнями та складом магазинів.
Сайт Microsoft dynamics	1.Комплексний пакет, що включає CRM, планування ресурсів підприємства та інші бізнес-інструменти; 2.Інсайти на основі штучного інтелекту для підвищення ефективності продажів; 3.Створення та керування цільовими маркетинговими кампаніями; 4.Безперешкодне підключення до Office 365, Teams та інших продуктів Microsoft.	Немає можливості управління персоналом, клієнтами, замовленнями та складом магазинів.

3

Рисунок Б.3 – Слайд 3

Розроблені алгоритми

Алгоритми, реалізовані під час розробки:

- алгоритм динамічної розмітки;
- алгоритм відображення кнопки для додавання;
- алгоритм для актуалізації таблиці warehouse під час створення поставки.

```

80 const warehouseCreateArray = await Promise.all(
81   dto.clothDelivered.map(async (cloth) => {
82     const warehouseCloth = await this.warehouseSchema.findOne(
83       {
84         clothId: cloth.clothId,
85         storeId: dto.deliveredId,
86       },
87     );
88     return Promise.all(
89       cloth.sizes.map(async (size) => {
90         const warehouseCloth = warehouseCloth.findOne(
91           (warehouse) => warehouse.size === size.size,
92         );
93         totalAmountDelivered += size.count;
94         if (warehouseCloth) {
95           await warehouseCloth.updateOne(
96             {
97               id: warehouseCloth.id,
98               amount: warehouseCloth.amount + size.count,
99               updatedAt: new Date().toISOString(),
100              updatedBy: userEmail,
101            },
102            {
103              $inc: {
104                amount: size.count,
105                updatedAt: new Date().toISOString(),
106                createdBy: userEmail,
107              },
108            },
109          );
110        } else {
111          return {
112            clothId: cloth.clothId,
113            storeId: dto.deliveredId,
114            size: size.size,
115            amount: size.count,
116            createdAt: new Date().toISOString(),
117            createdBy: userEmail,
118          };
119        }
120      }),
121    );
122  });
123 );
124 );
125 const warehouseCreateArray = [...warehouseCreateArray];
126 const lastWarehouse = await this.warehouseSchema.findOne(
127   {
128     createdAt: new Date().toISOString(),
129     updatedAt: new Date().toISOString(),
130   },
131   {
132     sort: { createdAt: -1 },
133   },
134 );
135 if (warehouseCreateArray.length > 0) {
136   if (lastWarehouse) {
137     this.warehouseSchema.create(
138       {
139         ...warehouseCreateArray[0],
140         id: (lastWarehouse.id || 0) + 1 + 1,
141       },
142     );
143   }
144 }

```

Наступним чином виглядає алгоритм для створення записів під час створення поставки.

Оскільки під час поставки кількість товару та наявність товару у конкретних магазинах змінюється, було створено наступний алгоритм:

- для кожного нового елемента поставки шукається наявний запис для по номеру одягу та магазину;
- з знайденого запису обирається необхідний розмір;
- додається кількість одягу для підрахунку загальної кількості доставленого одягу;
- якщо запис та розмір знайдено, то запис оновлюється, якщо ні то повертається у відповідь методу, для подальшого створення запису.

4

Рисунок Б.4 – Слайд 4

Структура сайту

Розробка інтерфейсу проводилась за більш розповсюдженим варіантом для CRM систем, у якому ліворуч знаходиться навігаційна панель, а праворуч від неї основний контент. Окрім основних вікон з даними, які беруться зі своїх таблиць бази було створено “Звіти”, для проведення аналітики продажів та менеджменту працівників.



Компоненти бібліотеки MUI, які були використані під час розробки інтерфейсу: Paper, Typography, Grid, IconButton, Dialog, Button, TextField, Divider, Table, Select, Menu, ThemeProvider.

5

Рисунок Б.5 – Слайд 5

Структура проєкту

Front-end

У корені проєкту front-end є тека src, в середині якої знаходяться усі файли проєкту окрім налаштувань. Налаштування знаходяться на одному рівні з текєю src, а саме це файли .env, .eslintrc.js, .gitignore, .prettierrc, package.json, package-log.json, tsconfig.json.

Тека Api містить у собі файли з методами доступу до back-end.

Тека Components містить усі компоненти з розміткою та CSS стилями. Containers містить контейнери компонент, які використовуються для того щоб передавати у компоненти дані та методи з редьюсерів.

Тека Redux складається з двох тек та файлу, тека interfaces містить інтерфейси кожного редьюсера, а reducers – самі редьюсери.

Файл reduxStore об'єднує всі редьюсери у спільний store

Тека utils містить загальні файли, які можуть бути використані будь-де.

У теці enums знаходяться файли з переліками, тобто константні значення, наприклад як статуси замовлень.

Тека interfaces містить інтерфейси.

Назва змінної	Опис	Приклад значення
REACT_APP_API_URL	Зберігає адрес, за яким front-end буде звертатися до back-end за даними.	http://localhost:3001/

```

> node_modules
> public
> src
  > Api
  > Components
  > Auth
  > Client
  > Cloth
  > Delivery
  > Header
  > MainContent
  > Order
  > Reports
  > Sidebar
  > Staff
  > Store
  > Containers
  > Redux
    > interfaces
    > reducers
    > reduxStore.ts
  > utils
    > enums
    > interfaces
  # App.css
  > App.tsx
  # index.css
  > index.tsx
  > react-app-env.d.ts
  > .env
  > .eslintrc.js
  > .gitignore
  > .prettierrc
  > package-lock.json
  > package.json
  > README.md
  > tsconfig.json

```

6

Рисунок Б.6 – Слайд 6

Структура проєкту

Back-end

У корені є 4 теки, 3 з них згенеровані автоматично, це `vercel` та `node_modules`.

Основна папка `src` містить весь код.

Тека `models` містить модулі проєкту. `App` модуль, який є головним та містить у середині усі інші, `mongoose` модуль містить підключення до `mongoose` бази даних.

`Shared` складається з `decorators`, `enums`, `filters` та `interceptors`. `Decorators` зберігає декоратори, які використовуються у проєкті, `enums` містить переліки констант, `filters` – обробник помилок, а `interceptors` – функції, які `Nest.js` викликає до та після виконання ендпоінта, у проєкті `interceptor` використовується для логування запитів та відповідей.

У файлі `.env` зберігаються локальні змінні проєкту, такий файл необхідний лише під час запуску проєкту з локальної машини, а у файлі `env-example` міститься структура `.env` файлу, тобто перелік змінних які є у проєкті.

Назва змінної	Опис	Приклад значення
MONGO_URL	Зберігає url адресу для підключення до бази даних.	<code>http://localhost:3001/</code>
MONGO_DB_NAME	Ім'я бази даних.	<code>cloth_crm</code>
MONGO_CONNECTION_TIMEOUT_MS	Кількість мілісекунд, протягом яких back-end намагатиметься підключитися до бази даних у випадку, коли це не вийде зробити одразу.	<code>2000</code>
PORT	Порт, на якому буде запускатися back-end.	<code>3001</code>
JWT_SECRET	Секрет, використовуючи який створюються JWT токени.	<code>PGUAsq7MB9pW</code>
JWT_EXPIRES_IN	Час, за який JWT токен стає протермінованим.	<code>30 days</code>
COOKIE_SECRET	Секрет, яким шифрується та дешифрується куки.	<code>RMgAUqdHVBck</code>

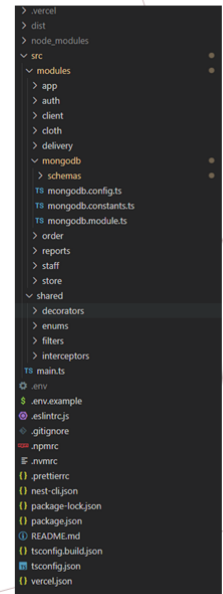


Рисунок Б.7 – Слайд 7

Схема даних БД

Таблиця Client

- name
- email
- phoneNumber
- size

Таблиця Cloth

- name
- price
- desc
- availableSizes

Таблиця Delivery

- deliveredTo
- deliveredFrom
- totalAmountDelivered
- price
- clothDelivered

Таблиця Order

- clientId
- clothId
- status
- price

Таблиця Staff

- name
- email
- isAdmin
- password
- position
- storeId
- isHidden

Таблиця Store

- address
- isActive

Таблиця Warehouse

- clothId
- storeId
- size
- amount

Кожна таблиця у БД має поля `createdAt`, `createdBy`, `updatedAt`, `updatedBy`, `deletedAt`, `id`.

Рисунок Б.8 – Слайд 8

Вибір мови програмування

Наразі найчастіше використовуваними мовами програмування для розробки вебсайтів та API є JavaScript (JS), TypeScript (TS) і PHP. PHP найстарша з них і з кожним роком втрачає популярність, в той час JS та TS все більше стають популярнішими. Особливою популярністю користується TS, яка є послідовником JS, але вже з типізацією. В якості мови програмування для проекту було обрано TS. Цей вибір був зроблений тому, що PHP вже застаріла і потребує переробки, JS є недостатньо описовою через відсутність явної типізації, що може створити зайвих проблем при розробці середніх та великих проектів.

Назва	Переваги	Недоліки
JS	1. Відсутність явної типізації прискорює розробку; 2. Мова має велику кількість додаткових бібліотек та вбудованих функцій.	1. Відсутність явної типізації ускладнює розробку великих проектів через неоднозначність об'єктів, які функція може повернути, і типів даних елементів об'єктів.
TS	1. Наявність явної типізації робить розробку великих проектів зручнішою; 2. Мова має велику кількість додаткових бібліотек, вбудованих функцій та інструментів.	1. Наявність явної типізації ускладнює і робить довшим початковий етап розробки через необхідність створення інтерфейсів.
PHP	1. Мова не конфліктує з html-розміткою, що дозволяє використовувати PHP-частину безпосередньо під час розмітки; 2. Мова достатньо проста, щоб нею міг користуватися будь-хто.	1. Мова стає менш популярною через появу зручніших конкурентів; 2. Вебсайти, написані на PHP, менш безпечні, ніж застосунки, написані іншими мовами.

9

Рисунок Б.9 – Слайд 9

Вибір фреймворка для frontend

Назва фреймворку / бібліотеки	Переваги	Недоліки
React	1. Завдяки технології ReactDOM, має високу швидкість рендеру компонентів; 2. Використовує функціональне програмування; 3. Підтримка розробки за допомогою TS та JS.	1. Через прагнення розробників переходити на функціональне програмування, об'єктно орієнтоване програмування перестає бути актуальним; 2. Перемішані розмітка та скрипти можуть заплутати.
Vue	1. Документація описує всі вбудовані засоби фреймворку; 2. Дозволяє створювати проекти різної величини. 3. Займає невеликий обсяг пам'яті	1. Не має значної кількості додаткових ресурсів та бібліотек; 2. Буде проблемою переробити проект який робився з використанням іншого фреймворка.
Angular	1. Має значну підтримку TypeScript; 2. Завдяки детальній документації не буде проблемою знайти опис будь-якого елементу системи.	1. Слабша продуктивність у порівнянні з конкурентами 2. Містить велику кількість структур для використання, через що ускладнюється вивчення фреймворку.

10

Рисунок Б.10 – Слайд 10

Вибір платформи для backend

Назва фреймворку / бібліотеки	Переваги	Недоліки
Express.js	1. Має досить швидкий старт. Для початку функціонування API майже не потрібно нічого налаштовувати; 2. Проста структура, що не потребує багатьох допоміжних елементів; 3. Має багато бібліотек що розширюють функціонал.	1. Не підходить для великих проєктів та середніх, що планують розширятися; 2. Оскільки має досить просту структуру та не має можливості підтримувати роботу з TS складно відбувається оновлення та подальше розширення.
Nest.js	1. Має обширну структуру проєкту, що спрощує пошук та зміну окремих елементів у великих проєктах; 2. Має великий обсяг бібліотек, що допомагають у розробці та розширюють функціонал; 3. З самого початку працює з TS; 4. Має вбудовані методи для роботи з запитами, у тому числі метод захисту від сторонніх.	1. Через велику структуру має повільний старт бо з'являється необхідність створення цієї структури; 2. Не підходить для маленьких API, через великий обсяг роботи необхідний для старту роботи API.

11

Рисунок Б.11 – Слайд 11

Вибір СКБД

Назва	Переваги	Недоліки
MongoDB	1. NoSQL-база даних, яка використовує гнучку, безсхемну модель даних, що робить її добре придатною для застосунків з динамічними структурами даних, що розвиваються; 2. Дані зберігаються у форматі BSON (Binary JSON), з яким легко працювати розробникам, особливо у вебзастосунках; 3. Має хмарні бази даних, що дозволяє безкоштовно зберігати на них дані маючи обмеження на кількість запитів.	1. MongoDB жертвує певним рівнем узгодженості даних на користь масштабованості та продуктивності, оскільки не повністю відповідає властивостям ACID; 2. Бази даних MongoDB можуть швидко зростати, і управління сховищем може стати проблемою при великомасштабному розгортанні.
PostgreSql	1. Система реляційних баз даних, яка відповідає властивостям ACID, що робить її придатною для застосунків, які вимагають високої узгодженості та надійності даних; 2. Підтримує потужну мову запитів SQL, яка дозволяє отримувати складні та структуровані дані та маніпулювати ними; 3. Має зрілу та активну спільноту з відкритим вихідним кодом, яка надає обширну документацію, плагіни та інструменти.	1. Хоча PostgreSQL може обробляти великі набори даних, вона може не так легко масштабуватися по горизонталі, як деякі NoSQL бази даних, такі як MongoDB; 2. Реляційна модель даних може бути складнішою в налаштуванні та підтримці, особливо для застосунків зі схемами даних, що швидко змінюються.

12

Рисунок Б.12 – Слайд 12

Використані бібліотеки

React

React - це бібліотека, яка використовується для розробки всього front-end. Бібліотека відображає усі елементи та надсилає запити до API.

Axios

Axios – використовується для надсилання запитів до API.

Eslint

Eslint – бібліотека, яку використовують для написання гарного та зручного для читання коду.

Nest.js

Основою back-end частини застосунку стала платформа Nest.js. Розробка на цій платформі виконується через так звані модулі, це класи, які поєднують у собі розподілення ендпоінтів, логіку для них та дозволяє використовувати інші модулі у собі.

Redux

Redux – найпопулярніша бібліотека для React. Ця бібліотека надає можливість створювати та керувати станом програми.

Mongoose

У проєкті використовується бібліотека mongoose для забезпечення взаємодії між back-end та MongoDB.

Notistack

Notistack – бібліотека, за допомогою якої було реалізовано відображення помилок від back-end. Ця бібліотека реалізує впливаючі повідомлення.

Class-validator

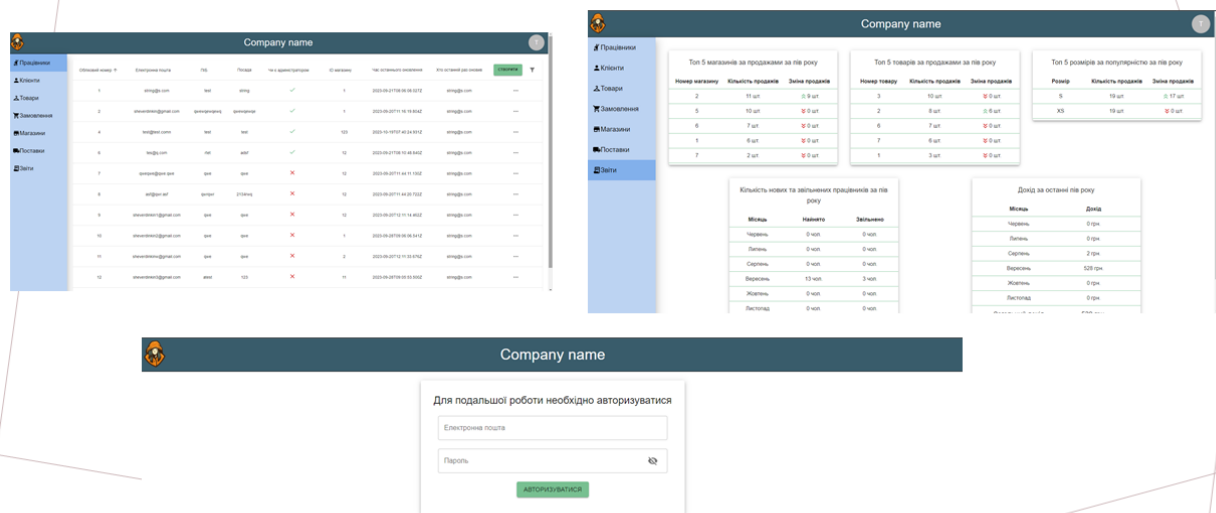
Class-validator – це бібліотека, яка реалізує валідацію даних з запитів.

13

Рисунок Б.13 – Слайд 13

Інтерфейс та екранні форми

Сторінки



14

Рисунок Б.14 – Слайд 14

Інтерфейс та екранні форми

Створити замовлення

ID клієнта

ID одягу
0

Кількість одягу
0

Розмір
S

ВИДАЛИТИ

ID одягу
0

Кількість одягу
0

Розмір
S

ВИДАЛИТИ

ID одягу
0

Кількість одягу
0

Розмір
S

ДОДАТИ ТОВАР
ВИДАЛИТИ

Статус замовлення
CREATED

ВІДМІНИТИ

СТВОРИТИ

Створити працівника

Електронна пошта

Прізвище Ім'я Побатькові

Посада

ID магазину

Зробити адміністратором

Пароль

ВІДМІНИТИ

СТВОРИТИ

Форми

Створити магазин

Адреса

Чи працює

ВІДМІНИТИ

СТВОРИТИ

Створити поставку

Доставка в

Доставка з

Тип поставки
INTERNAL

ID одягу
0

Розміри

Кількість
0

шт.
X

Новий розмір

ВІДМІНИТИ

ID одягу
0

Розміри

Кількість
0

шт.
X

Новий розмір

ДОДАТИ ТОВАР
ВИДАЛИТИ

ВІДМІНИТИ

СТВОРИТИ

Створити товар

Назва

Опис товару

Ціна

Розмір
XS, S, M

ВІДМІНИТИ

СТВОРИТИ

Створити клієнта

Електронна пошта

Прізвище Ім'я Побатькові

Номер телефону

Розмір одягу
XS

ВІДМІНИТИ

СТВОРИТИ

Рисунок Б.15 – Слайд 15

Інтерфейс та екранні форми

Повідомлення

Створити працівника

Електронна пошта

Ця поле обов'язкове

Прізвище Ім'я Побатькові

Мінімальна довжина введення : 2 символи

Посада

Мінімальна довжина введення : 2 символи

ID магазину

Ця поле обов'язкове

Зробити адміністратором

Пароль

Мінімальна довжина паролю : 8 символів, має містити хоча б 1 букву та 1 цифру

ВІДМІНИТИ

СТВОРИТИ

Видалити працівника "test" ?

ВІДМІНИТИ

ВИДАЛИТИ

Забрати можливості адміністратора у працівника "test" ?

ВІДМІНИТИ

ЗАБРАТИ ПРАВА АДМІНІСТРАТОРА

Зробити працівника "test" адміністратором ?

Пароль

ВІДМІНИТИ

ЗРОБИТИ АДМІНІСТРАТОРОМ

Рисунок Б.16 – Слайд 16

Висновки

У дипломній роботі було проаналізовано предмет і тему дослідження, а саме системи управління мережевими магазинами та розробку такої системи. У ході виконання роботи було проаналізовано існуючі рішення, такі як ClickUp, Zendesk та Microsoft Dynamics, описано їх переваги та недоліки, а також показано, чого їм не вистачає для завдань проекту.

Було проаналізовано поширені мови програмування для веброзробки та обрано TypeScript як мову розробки проекту. Також було проаналізовано різні системи управління базами даних і обрано MongoDB, як найбільш підходящу. Також було описано бібліотеки, платформи та фреймворки, що використовувалися при розробці, їх застосування в проекті та приклади їх використання.

Були надані схеми бази даних з назвами полів, описами полів та необхідністю їх заповнення. Описано структуру проекту з поясненням елементів структури, її змісту та призначення, а також алгоритми, використані при розробці. Також було описано спосіб взаємодії програміста з базою даних, тобто створення та використання бази даних.

Крім того, було названо користувача та визначено сторінки, до яких він має доступ. У кінці, були відображені помилки, повідомлення та зовнішній вигляд сторінок проекту.

Кінцевим результатом проекту стало створення вебсайту для управління та аналізу мережі магазинів одягу.

Рисунок Б.17 – Слайд 17