МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА СИСТЕМИ УПРАВЛІННЯ ВИРОБНИЦТВОМ ТА ОБРОБКИ ЗАМОВЛЕНЬ ДЛЯ МЕБЛЕВОЇ МАЙСТЕРНІ ЗАСОБАМИ POSTGRESQL, ASP.NET CORE ТА VUE.JS»**

Виконав: студент 4 курсу, групи 6.1219-2пі

спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

освітньої програми програмна інженерія
(назва освітньої програми)

Д.П. Наумов
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Мильцев О.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний
Кафедра програмної інженерії
Рівень вищої освіти бакалавр
Спеціальність 121 інженерія програмного забезпечення
Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент
_____ Лісняк А.О.
(підпис)
« 07 » лютого 2023 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Наумову Дмитру Павловичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка системи управління виробництвом та обробки замовлень для меблевої майстерні засобами PostgreSQL, ASP.NET Core та Vue.js

керівник роботи Мильцев Олександр Михайлович, к.ф.-м.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 26 » січня 2023 року № 102-с

2. Строк подання студентом роботи 07.06.2023

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Опис виробничих процесів у меблевій майстерні.
2. Обґрунтування використаних технологій.
3. Модель бази даних та WebAPI.
4. Реалізація серверної частини.
5. Реалізація клієнтської частини.
6. Керівництво користувача.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 07.02.2023

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.02.2023	
2.	Збір вихідних даних.	14.02.2023	
3.	Обробка методичних та теоретичних джерел.	24.02.2023	
4.	Розробка першого розділу.	06.03.2023	
5.	Розробка другого розділу.	28.03.2023	
6.	Розробка третього розділу.	05.05.2023	
7.	Розробка четвертого розділу.	24.05.2023	
8.	Оформлення та нормоконтроль	01.06.2023	
	кваліфікаційної роботи.		
9.	Захист кваліфікаційної роботи.	23.06.2023	

Студент _____ Д.П. Наумов
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.М. Мильцев
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка системи управління виробництвом та обробки замовлень для меблевої майстерні засобами PostgreSQL, ASP.NET Core та Vue.js»: 142 с., 99 рис., 10 табл., 25 джерел, 6 додатків.

ASPNETCORE, ORDER PROCESSING, POSGRESQL, PRODUCTION MANAGEMENT, VUEJS, WEBAPI.

Об’єкт дослідження – облікові процеси у меблевій майстерні.

Мета роботи: розробити вебзастосунок для автоматизації процесів обліку (прийом замовлень, задачі, облік сировини і т.д.) у меблевій майстерні.

Методи дослідження – аналіз, моделювання, конструювання програмного забезпечення.

В роботі розглянуто моделі виробничих процесів що відбуваються у меблевій майстерні, запропоновано варіанти використання, модель даних, модель WebAPI. Система реалізована з використанням стеку технологій PostgreSQL, ASP.NET Core та Vue.js. Після реалізації системи наведено інструкції по її використанню.

Кінцевим результатом роботи є готова система управління виробництвом та обробки замовлень для меблевої майстерні, що складається з артефактів Fwsh.WebApi (сервер), CustomersPanel (вебсайт для покупця), WorkersPanel (панель робітника), ManagersPanel (панель менеджера). Вихідний код роботи опублікований у загальнодоступному репозиторії.

Кінцеві та проміжні результати можуть бути використані при подальшій розробці систем для автоматизації виробничого обліку та прийому замовлень для інших підприємств.

SUMMARY

Bachelor's qualifying paper «Development of Production Management and Order Processing System for a Furniture Workshop using PostgreSQL, ASP.NET Core and Vue.js»: 142 pages, 99 figures, 10 tables, 25 references, 6 supplements.

ASPNETCORE, ORDER PROCESSING, POSGRESQL, PRODUCTION MANAGEMENT, VUEJS, WEBAPI.

Object of the study is accounting processes in a furniture workshop.

Aim of the work is to develop a web application to automate accounting processes (order processing, tasks, storage checks etc) in a furniture workshop.

Methods of the study – analysis, modeling, software construction.

The study includes modeling of production processes in a furniture workshop, use case proposal, data modeling, WebAPI design. System has been implemented using PostgreSQL, ASP.NET Core and Vue.js technology stack.

Final result of the work is a complete production management and order processing system for a furniture workshop consisting of these artifacts: Fwsh.WebApi (server), Customer's panel, Worker's panel, Manager's panel. Source code is available in a public repository.

Final and intermediate results of the work can be used in further development of production management and order processing systems for other businesses.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

СУБД	Система управління базами даних
DNS	Domain Name System
Fwsh	Furniture Workshop
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
JSON	JavaScript Object Notation
ORM	Object Relational Mapping
SSL	Secure Socket Layer
URL	Uniform Resource Locator
WebAPI	Web Application Programming Interface

ВСТУП

В 21 сторіччі, яке називають епохою Інтернету, більшість підприємств уже давно закріпилися на просторах глобальної мережі та використовують її для приваблення нових клієнтів та/або взаємодії з іншими підприємствами, або для вирішення більш локальних питань, як-от автоматизація обліку. Проте в деяких малих підприємствах досі ведеться облік на папері або в таблицях Excel, що не є повноцінним рішенням та знижує продуктивність праці. А відсутність підприємства в Інтернеті означає відсутність засобу залучення нових клієнтів та широкого каналу комунікації з останніми.

В м. Запоріжжя є меблеві майстерні (назвати точну кількість неможливо), що ведуть облік в ручному режимі та не мають вебсайту для залучення нових клієнтів. Ці підприємства мають потенціал росту, але він залишається нереалізованим.

Причинами неавтоматизації підприємства може бути:

– небажання ознайомлюватися з відомими рішеннями;
– специфіка підприємства.

З урахуванням наведених вище причин пропонується розробити систему для автоматизації виробничого обліку та прийому замовлень, що дозволить пройти повний цикл від перегляду каталогу до подачі замовлення та далі до його виконання, та корисних побічних ефектів – порахунку кількості ресурсів на складі, обліку замовлень на поставку ресурсів та обліку заробітної плати.

1 ЗБІР ТА АНАЛІЗ ВИМОГ ДО СИСТЕМИ

1.1 Інтерв’ю зацікавлених осіб

Для збору відомостей про предметну область було проведене інтерв’ю працівників та керівництва меблевої майстерні (див. додаток А).

За результатами інтерв’ю було виявлено потребу в розробці системи управління виробництвом та обробки замовлень (далі – система) для меблевої майстерні. Відомо, що основними процесами в майстерні є:

– прийом замовлень на виробництво меблів;
– прийом замовлень на ремонт меблів;
– виробництво меблів;
– облік ресурсів.

У процесах беруть участь такі категорії осіб:

– покупці;
– менеджер;
– робітники;
– постачальники ресурсів.

Для подальшого розуміння процесів, що необхідно автоматизувати, варто розробити моделі, що відображають послідовності дій та переходи між станами відстежуваних об’єктів.

1.2 Моделювання процесів у меблевій майстерні

Для розуміння процесу виробництва меблів буде доречно зобразити його у вигляді діаграми IDEF0 (див. рис. 1.1). Те ж саме виконується для ремонту меблів (див. рис. 1.2).

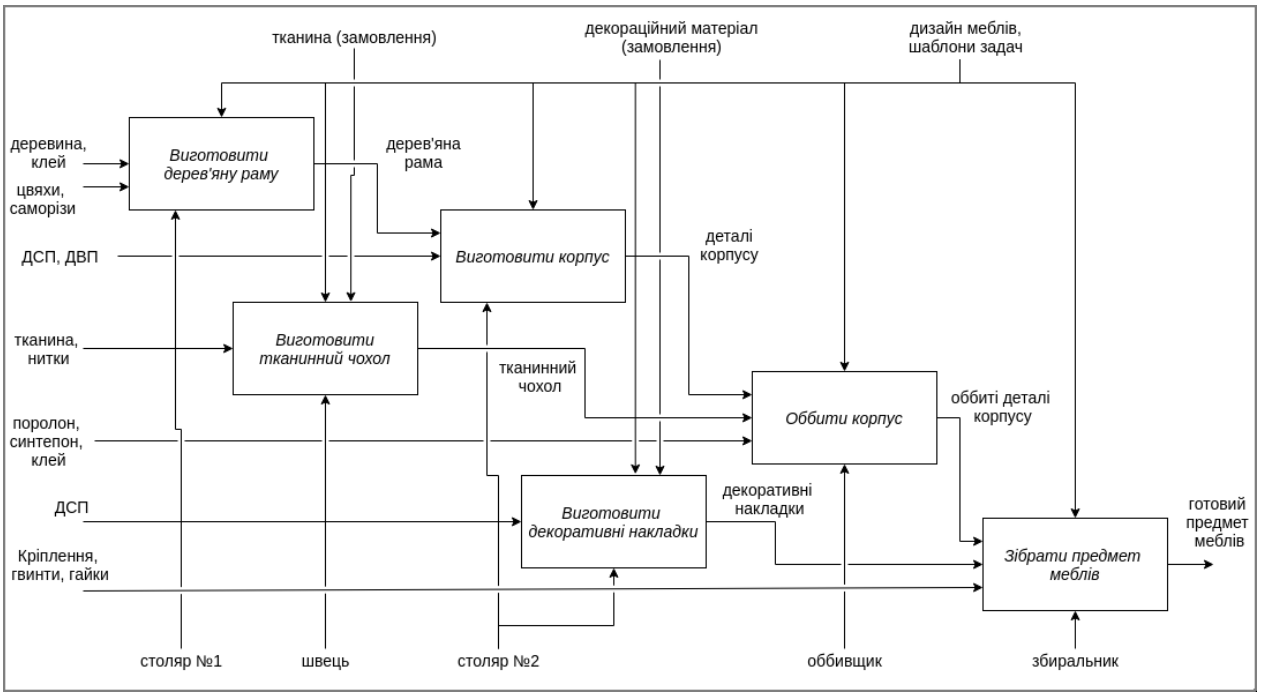

Рисунок 1.1 – Діаграма IDEF0 процесу «Виготовлення меблів»

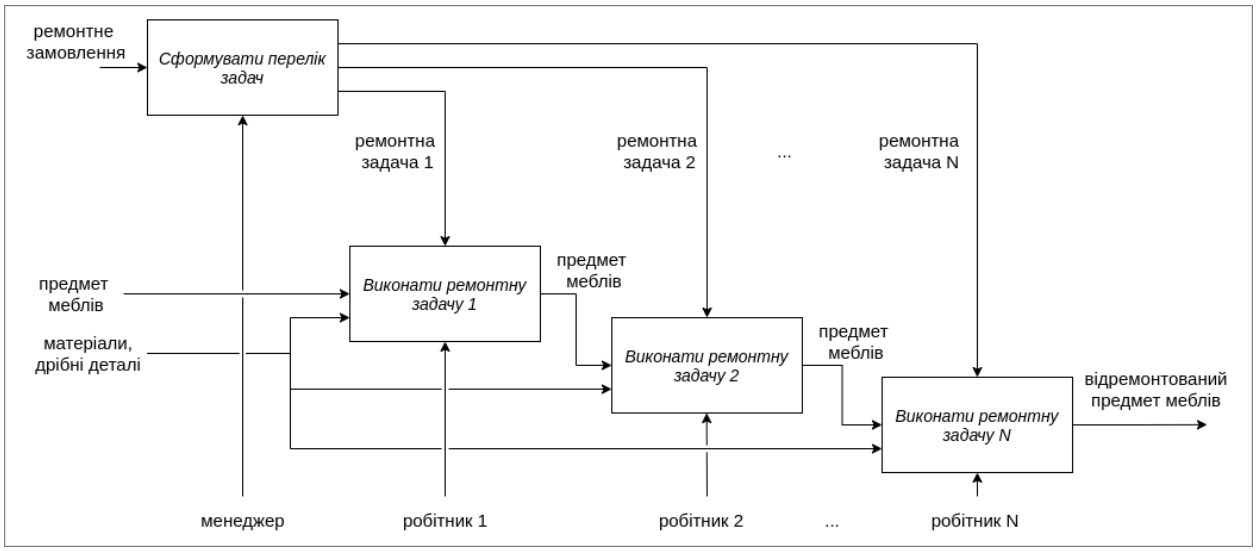

Рисунок 1.2 – Діаграма IDEF0 процесу «Ремонт меблів»

Для розуміння можливих статусів замовлення та переходів між ними буде доречно зобразити процес виконання замовлення (див. рис. 1.4) у вигляді модифікованої нотації EPC з такими умовними позначеннями (див. рис. 1.3).

Рисунок 1.3 – Умовні позначення до діаграми процесу

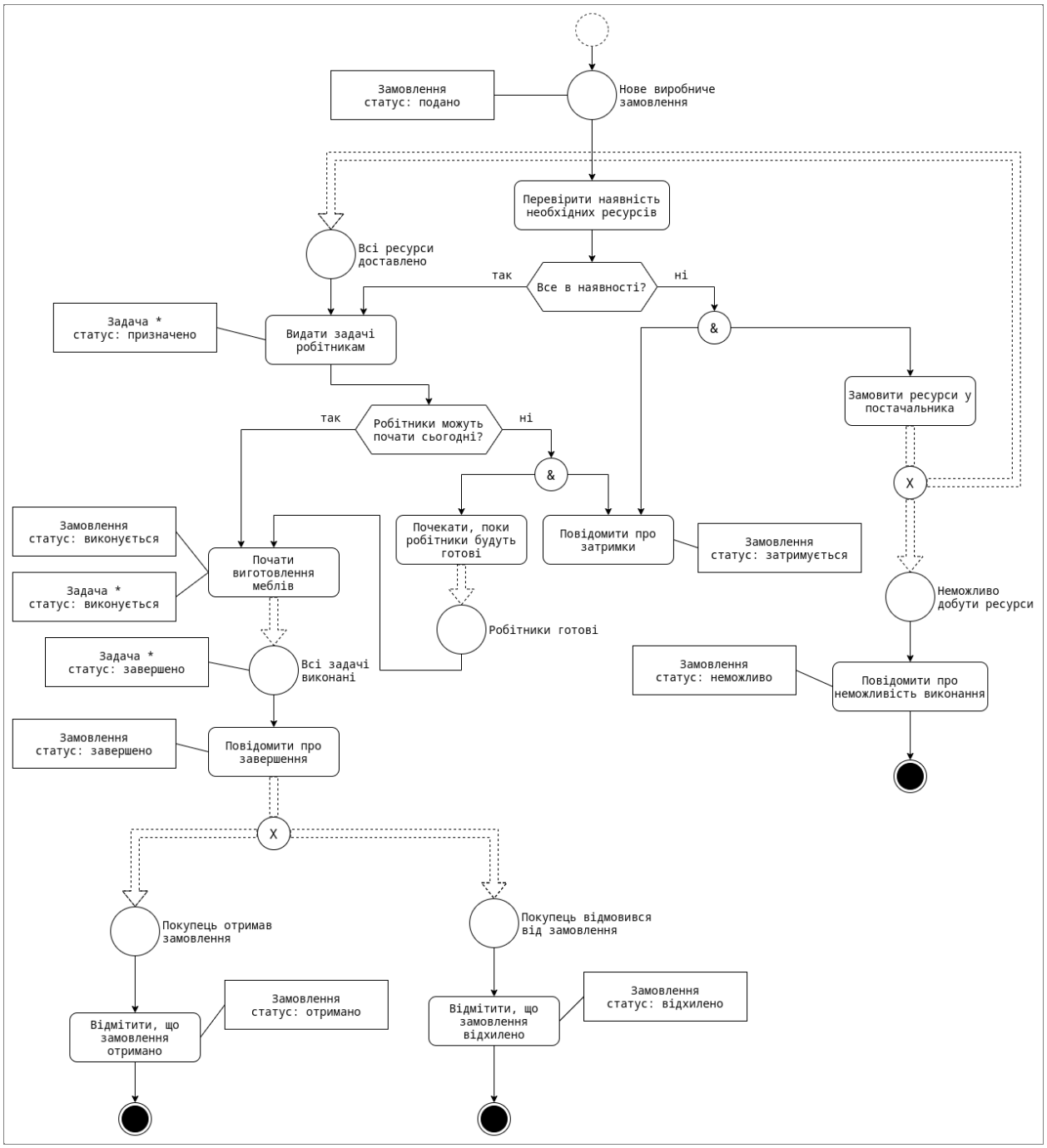

Рисунок 1.4 – Можливі статуси замовлення

Для розуміння статусів задачі та можливих переходів між ними варто зобразити їх у тій же нотації, що була використана для статусу замовлення (див. рис. 1.5).

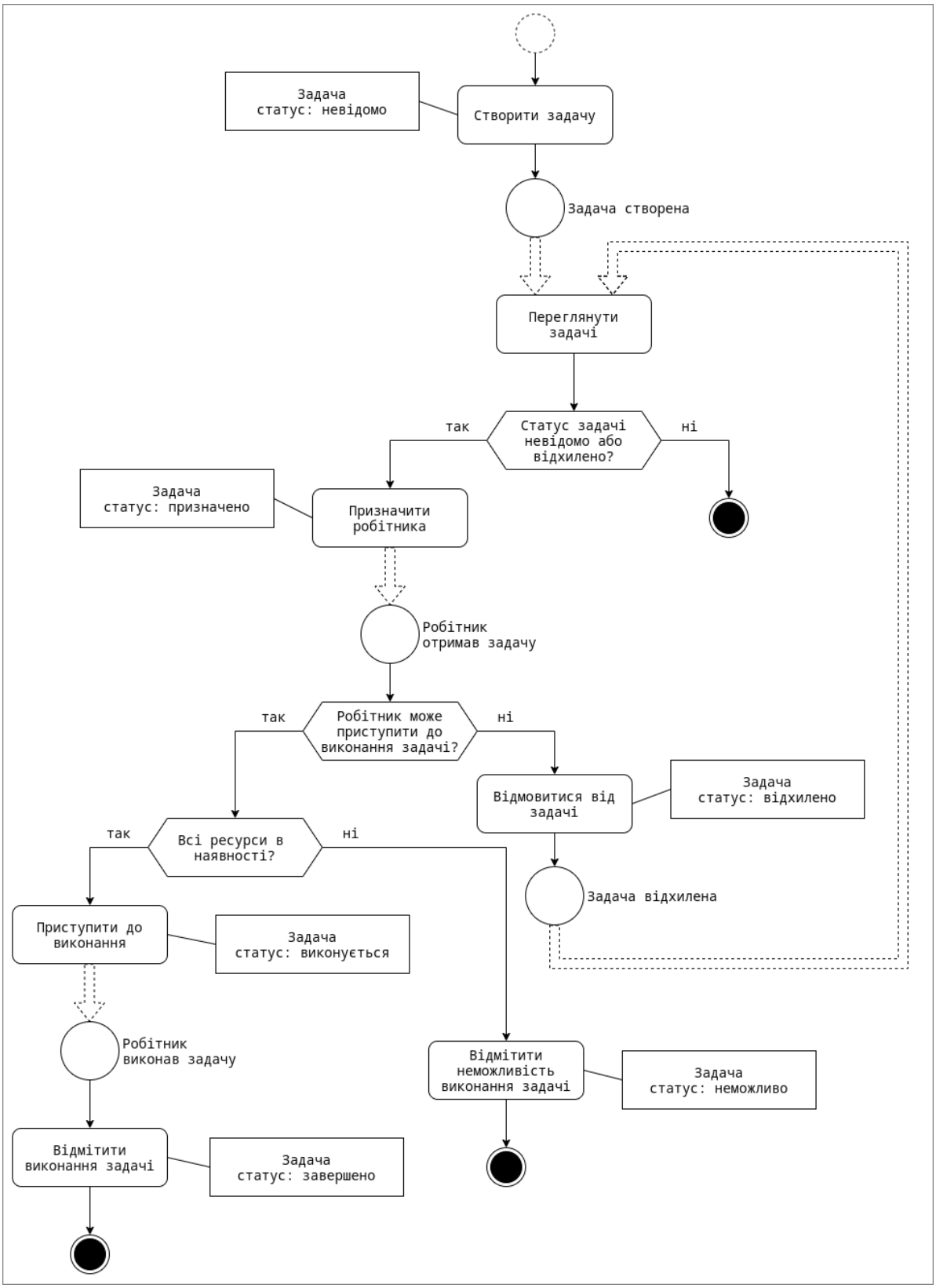

Рисунок 1.5 – Можливі статуси задачі

1.3 Технічне завдання до роботи

На основі зібраних вимог користувачів системи та моделі процесів можна виділити такий перелік спільних функцій для всіх користувачів:

– реєстрація, вхід в обліковий запис;
– перегляд власного профілю.

Перелік функціоналу для покупців:

– перегляд каталогу меблів;
– перегляд каталогу тканин;
– подача замовлення на виробництво меблів;
– подача замовлення на ремонт меблів;
– перегляд стану виробничих та ремонтних замовлень;
– перегляд сповіщень, що стосуються замовлення.

Перелік функціоналу для робітників:

– перегляд власних виробничих задач;
– перегляд власних ремонтних задач;
– звітування про стан задачі;
– звітування про використані ресурси;
– облік кількості ресурсів на складі;
– облік власної заробітної плати.

Перелік функціоналу для менеджера:

– перегляд та редагування каталогу меблів;
– перегляд та редагування шаблонів задач;
– перегляд виробничих замовлень;
– перегляд ремонтних замовлень;
– надсилання сповіщень про стан замовлення;
– створення задач за шаблоном;
– створення індивідуальних задач для ремонту меблів;
– призначення виконавців задач;
– перегляд списку робітників, покупців;
– перегляд списку постачальників, додавання нових;
– перегляд ресурсів на складі;
– додавання нових ресурсів, редагування ресурсів;

– замовлення ресурсів у постачальника;
– облік заробітної плати.

Нефункціональні вимоги до системи:

– приємний та зрозумілий інтерфейс вебзастосунку;
– сумісність вебзастосунку зі стандартом ES6;
– розмір скомпільованої клієнтської частини не більше 1Мбайт.

Обмеження системи (що не буде реалізовано):

– оплата замовлення онлайн;
– сповіщення покупця у вигляді SMS.

Для реалізації сформованого переліку вимог необхідно спроєктувати та реалізувати:

– сутності даних системи;
– сервер WebAPI;
– вебзастосунок (сайт) для покупця;
– вебзастосунок для робітника;
– вебзастосунок для менеджера.

1.4 Обґрунтування використаних технологій

Для реалізації системи необхідно обрати СУБД, технологію для реалізації сервера WebAPI, технологію для реалізації клієнтської частини.

Було прийнято рішення використовувати саме реляційну СУБД, тому що предметна область передбачає велику кількість сутностей та зв’язків між ними (див. 2.2). Критеріями вибору СУБД є:

– безкоштовність;
– підтримка реляційної моделі;
– регулярне оновлення та документація;
– підтримка ОС Linux;
– простота встановлення та використання.

Серед відомих реляційних СУБД довільним чином обрано PostgreSQL. Це реляційна СУБД, яка є безкоштовною, регулярно оновлюється і окрім цього має ORM-адаптери для багатьох мов програмування [1].

Було прийнято рішення використовувати саме WebAPI та «товстих клієнтів», тому що цей підхід дозволяє розробити серверний застосунок, що працює максимально однаковим чином в усіх варіантах використання (прийняти запит, виконати потрібні маніпуляції, повернути результат в форматі JSON) та не відповідає за візуальне представлення даних [2].

Для технологій реалізації сервера WebAPI, окрім 5 базових вимог що висувалися до СУБД, додаються такі опціональні вимоги:

– наявність розвинутих фреймворків WebAPI;
– об’єктна орієнтованість та статична типізація;
– можливість компіляції в форматі «self-contained».

Серед відомих технологій обрано мову програмування C Sharp та фреймворк ASP.NET Core, що має розвинуту стандартну бібліотеку та дозволяє розробляти WebAPI на основі контролерів [3]. Важливою частиною екосистеми .NET є EF Core – універсальна об’єктно-реляційна обгортка для взаємодії з базами даних, що дозволяє робити операції пошуку, читання, запису, видалення даних методами об’єктів, приводить результати всіх запитів до об’єктів мови програмування C Sharp, повністю звільняє розробника від необхідності самостійного написання SQL-запитів [4]. Окрім того, .NET має єдиний стандартний інтерфейс командного рядка dotnet CLI, що дозволяє створювати проєкти, запускати їх виконання, компілювати, встановлювати пакети [5].

Реалізація клієнтської частини у вигляді браузерного вебзастосунку передбачає використання HTML, CSS та Javascript [6]. HTML – це мова розмітки, що забезпечує семантичний скелет вебсторінки, CSS – використовується для стильового оформлення, Javascript – для реалізації логіки вебсторінок [6]. Але використання цих засобів у чистому вигляді не є ефективним, тому для пришвидшення роботи довільним чином був обраний фреймворк Vue.js. Vue.js надає засоби для декларативного рендерингу вебсторінок, що базується на використанні компонентів, реактивних об’єктів, одно- та двостороннього зв’язування, подій та їх обробників, ін’єкції залежностей [7].

Також при реалізації клієнтської частини використані бібілотеки:

– vue-router для навігації [8];
– axios для HTTP-запитів [9];
– qs для формування рядків запитів з пар ключ-значення [10].

Висновки до розділу 1

На даному етапі було проведено моделювання облікових процесів у меблевій майстерні. На основі опису процесів складено перелік функцій системи, що необхідно реалізувати для автоматизації обліку в майстерні. Також наведено обґрунтування технологій, що будуть використані при програмній реалізації системи.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Варіанти використання системи

На даному етапі необхідно дати відповідь на питання «Що система повинна робити?». Для кращого розуміння функцій розроблюваної системи варто побудувати діаграми варіантів використання окремо для ролей:

– покупець (Customer);
– робітник (Worker);
– менеджер (Manager).

2.1.1 Варіанти використання для покупця

Покупець може взаємодіяти з власним обліковим записом (див. рис. 2.1), типовим набором дій є: зареєструватися, увійти в обліковий запис, переглянути профіль, редагувати профіль.

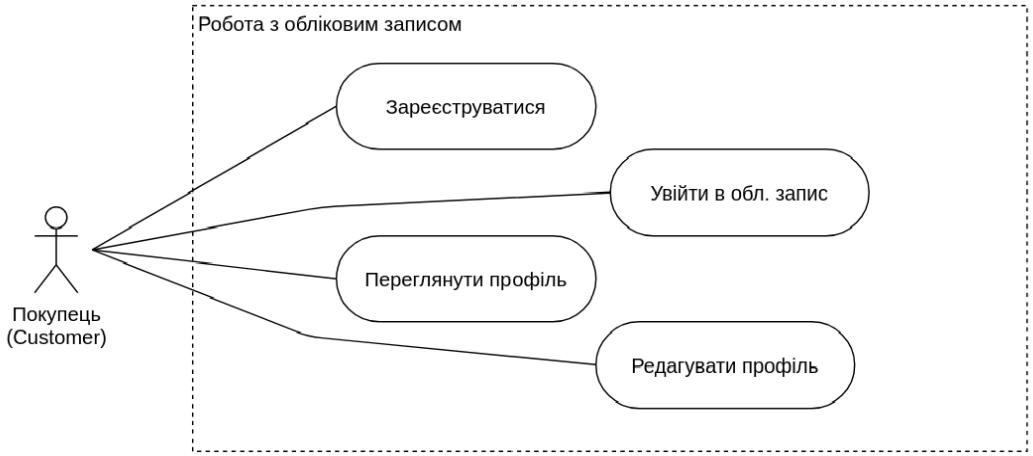

Рисунок 2.1 – Варіанти взаємодії покупця з обліковим записом

Покупець може переглядати каталог (див. рис. 2.2), до цього набору входять дії: перегляд каталогу меблів, детальний огляд дизайну меблів, пошук по типу меблів, перегляд каталогу тканин, перегляд каталогу декоративних матеріалів.

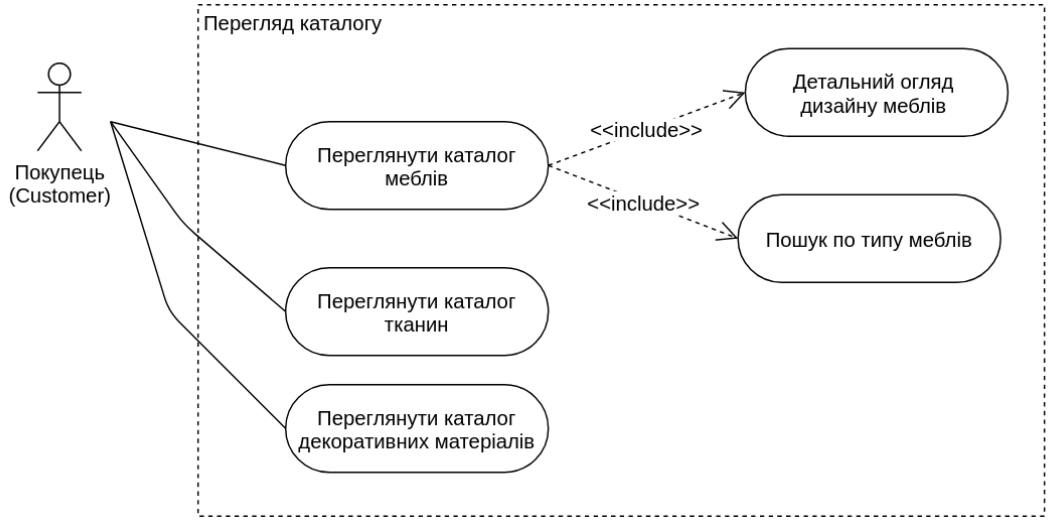

Рисунок 2.2 – Варіанти взаємодії покупця з каталогом

Покупець може взаємодіяти зі списком замовлень (див. рис. 2.3), а саме: переглядати список замовлень (виробничі замовлення, ремонтні замовлення), переглянути детальну інформацію про замовлення, переглянути сповіщення, створити виробниче замовлення, створити ремонтне замовлення, підтвердити замовлення, видалити замовлення.

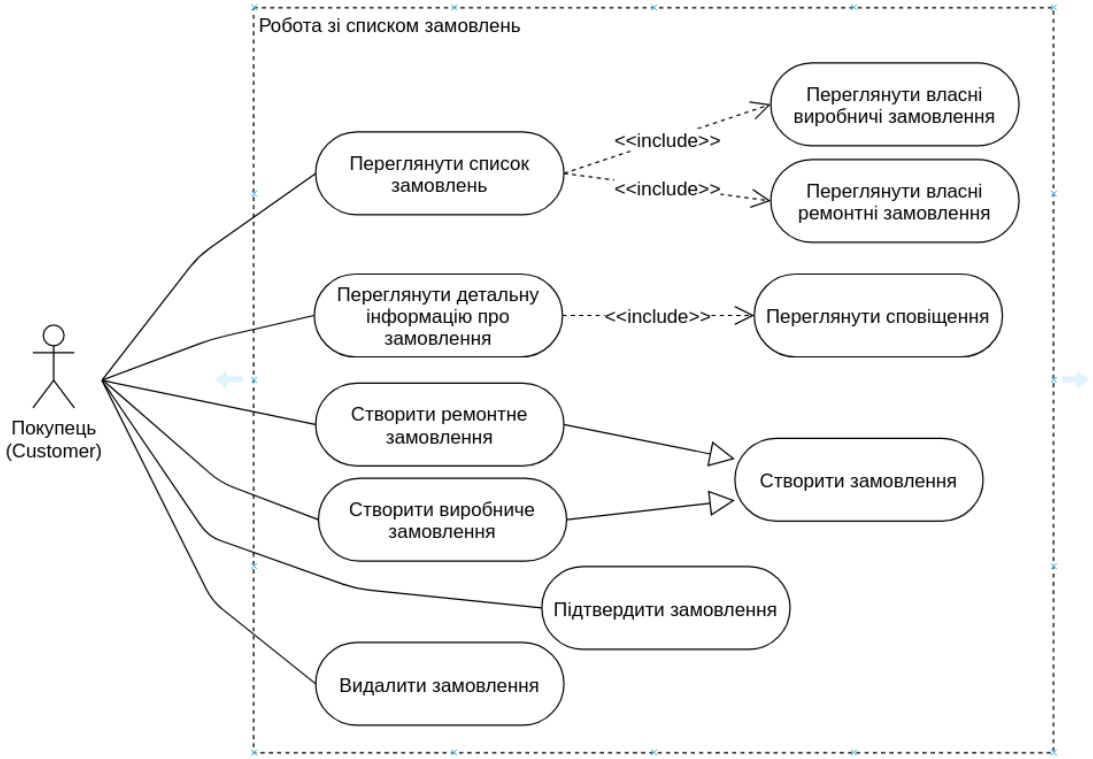

Рисунок 2.3 – Варіанти взаємодії покупця зі списком замовлень

2.1.2 Варіанти використання для робітника

Робітник може взаємодіяти з власним обліковим записом (див. рис. 2.4), а саме: увійти в обліковий запис, зареєструватися, переглянути профіль, редагувати профіль, згенерувати заробітний чек (запит на отримання заробітної плати).

Рисунок 2.4 – Варіанти взаємодії робітника з обліковим записом

Робітник може взаємодіяти з ресурсами (див. рис. 2.5), а саме: переглядати список тканин, переглядати список матеріалів, переглянути список деталей (узагальнюється у вигляді перегляду ресурсів), переглянути детальний опис ресурсу, записати кількість ресурсу на складі.

Робітник може взаємодіяти зі списком задач (див. рис. 2.6), це включає набір дій: переглянути список задач, переглянути власні виробничі задачі, переглянути власні ремонтні задачі, змінити статус задачі, записати кількість використаних ресурсів.

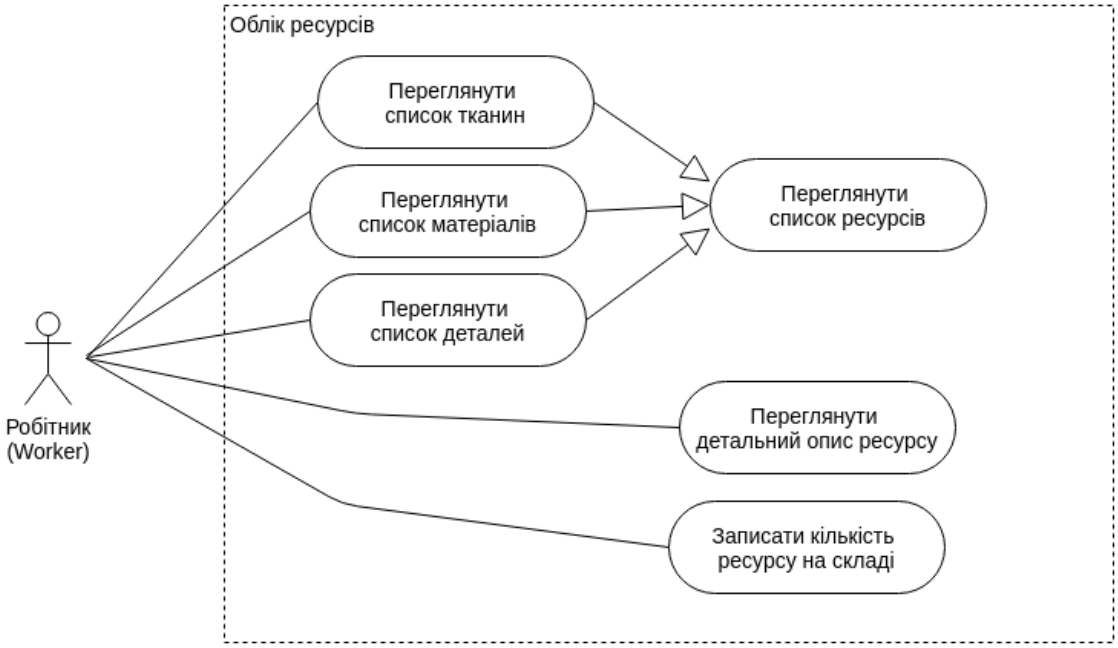

Рисунок 2.5 – Варіанти взаємодії робітника з ресурсами

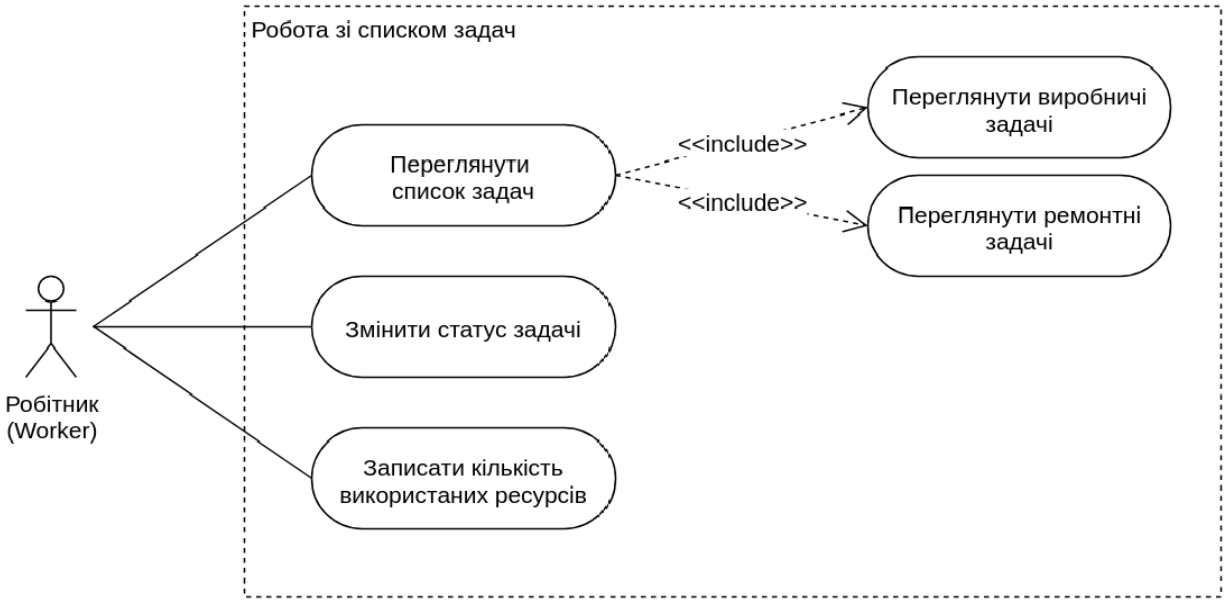

Рисунок 2.6 – Варіанти взаємодії робітника зі списком задач

2.1.3 Варіанти використання для менеджера

Менеджер може взаємодіяти з власним обліковим записом (див. рис. 2.7), а саме: увійти в обліковий запис, зареєструватися, переглянути профіль, редагувати профіль.

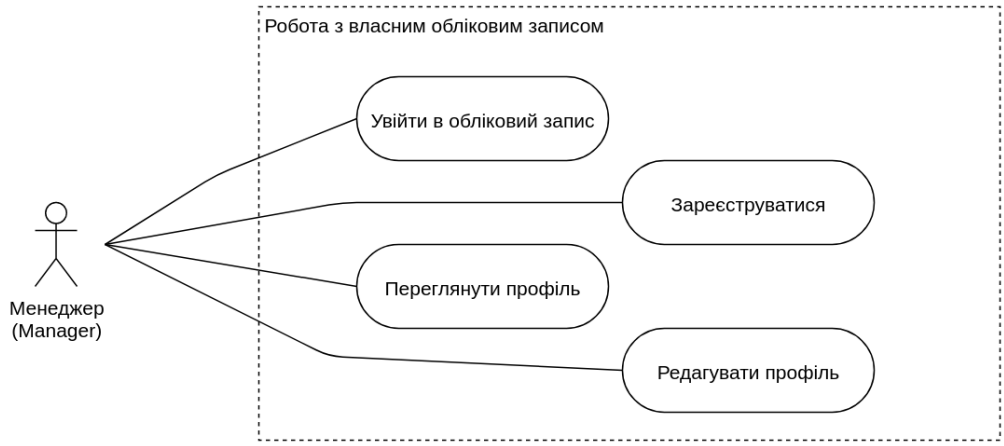

Рисунок 2.7 – Варіанти взаємодії менеджера з власним обл.записом

Менеджер може взаємодіяти з профілями інших користувачів (див. рис. 2.8), а саме: переглянути список покупців, переглянути список постачальників, переглянути список робітників, переглянути запити на заробітну плату, створити обліковий запис постачальника, редагувати обліковий запис постачальника.

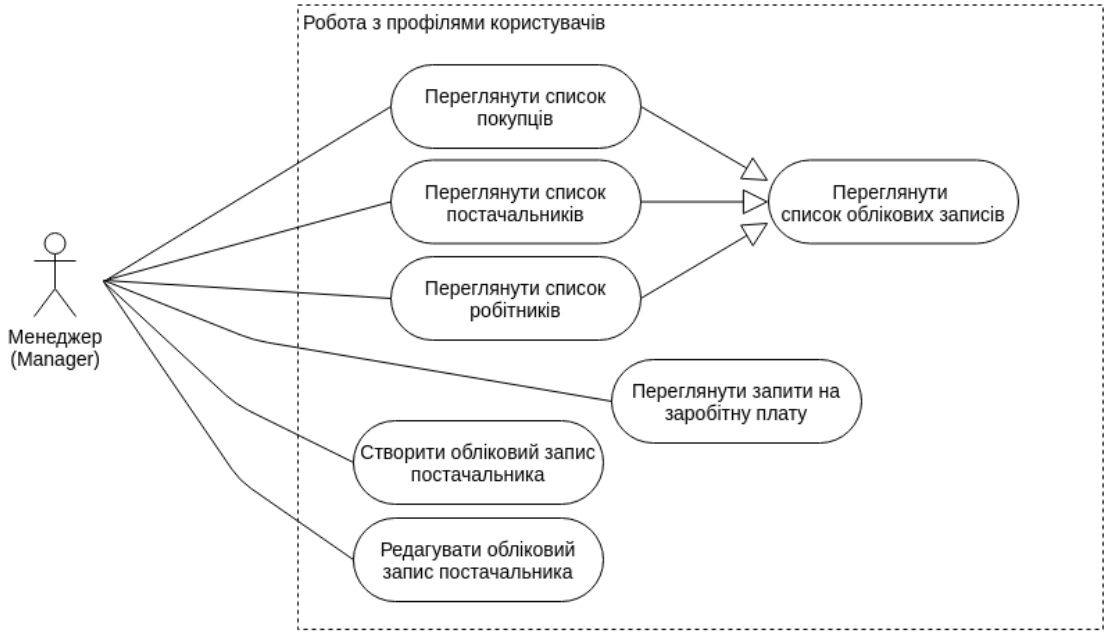

Рисунок 2.8 – Варіанти взаємодії менеджера з профілями користувачів

Менеджер може вести облік ресурсів (див. рис. 2.9), що включає такі дії: переглянути каталог типів тканин, редагувати тип тканини, додати або додати новий тип тканини, переглянути каталог кольорів, редагувати або додати новий колір, переглянути список ресурсів, переглянути список

тканин, переглянути список матеріалів, переглянути список деталей, переглянути детальний опис одного ресурсу, редагувати або створити новий ресурс, переглянути замовлення на поставку ресурсів, редагувати або створити нове замовлення на поставку ресурсів.

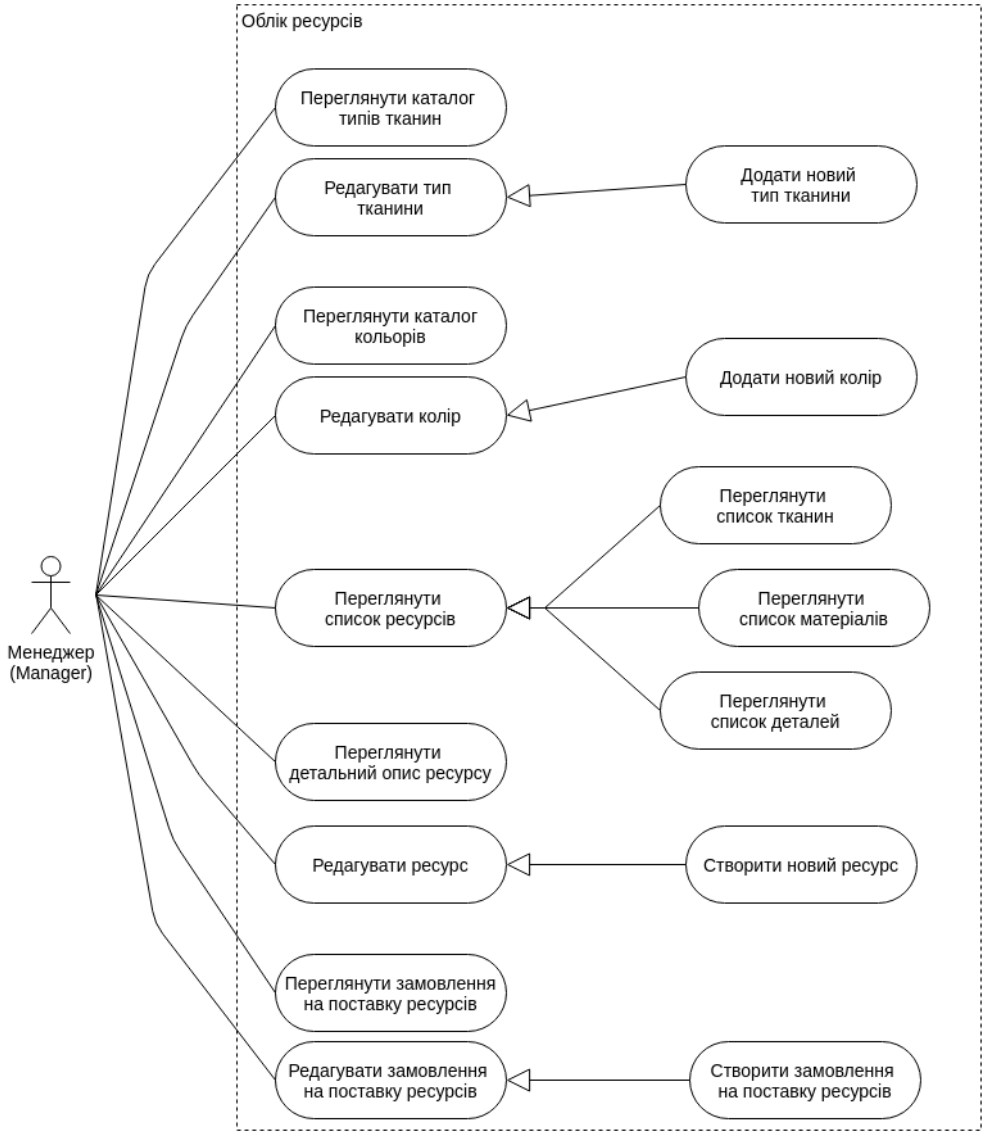

Рисунок 2.9 – Варіанти взаємодії менеджера з ресурсами

Менеджер може взяємодіяти зі списком замовлень (див. рис. 2.10), що включає дії: переглянути список замовлень (виробничі замовлення, ремонтні замовлення), переглянути детальну інформацію про замовлення, змінити статус замовлення, додати сповіщення.

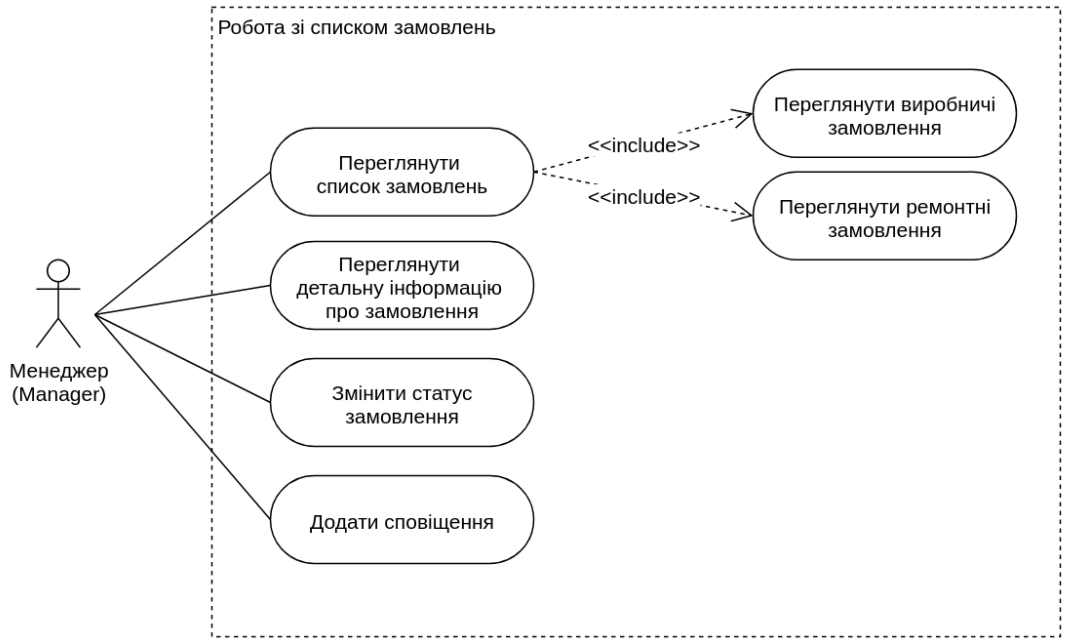

Рисунок 2.10 – Варіанти взаємодії менеджера зі списком замовлень

Менеджер може взаємодіяти зі списком задач (див. рис. 2.11), що включає дії: переглянути список задач (виробничі задачі, ремонтні задачі), створити ремонтну задачу, згенерувати виробничі задачі за шаблоном, призначити робітника, змінити статус задачі.

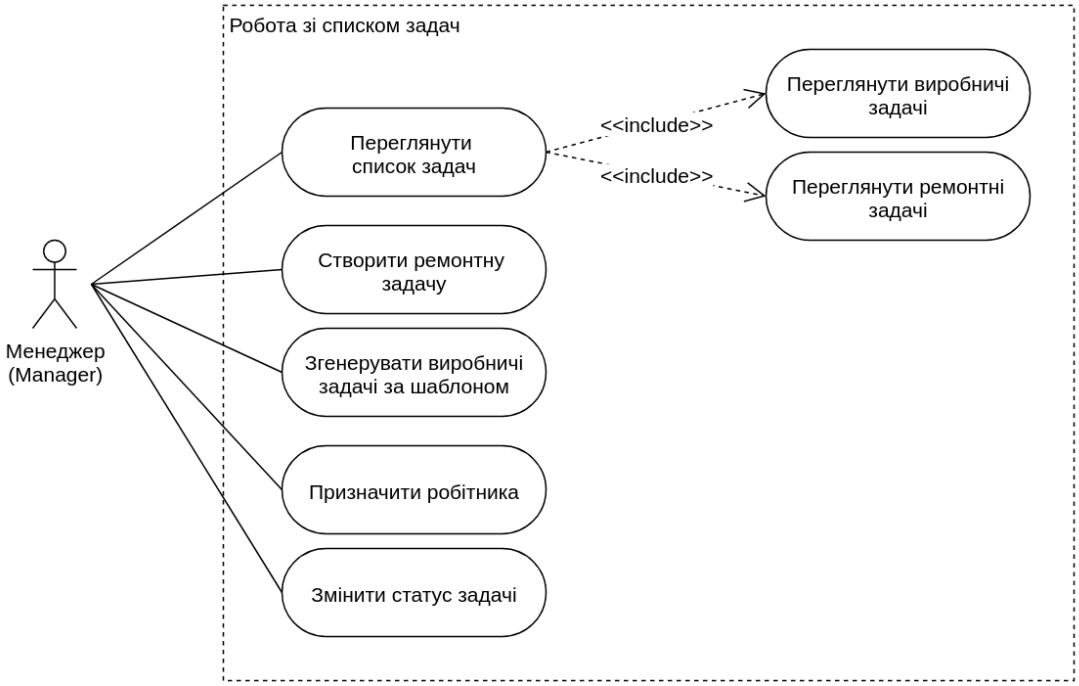

Рисунок 2.11 – Варіанти взаємодії менеджера зі списком задач

Менеджер може взаємодіяти з виробничими шаблонами (див. рис. 2.12), а саме: переглянути список дизайнів меблів, редагувати або створити новий дизайн меблів, приховати дизайн меблів з каталогу, переглянути шаблони задач, редагувати шаблон задачі (в тому числі додати використання ресурсів), створити новий шаблон задачі.

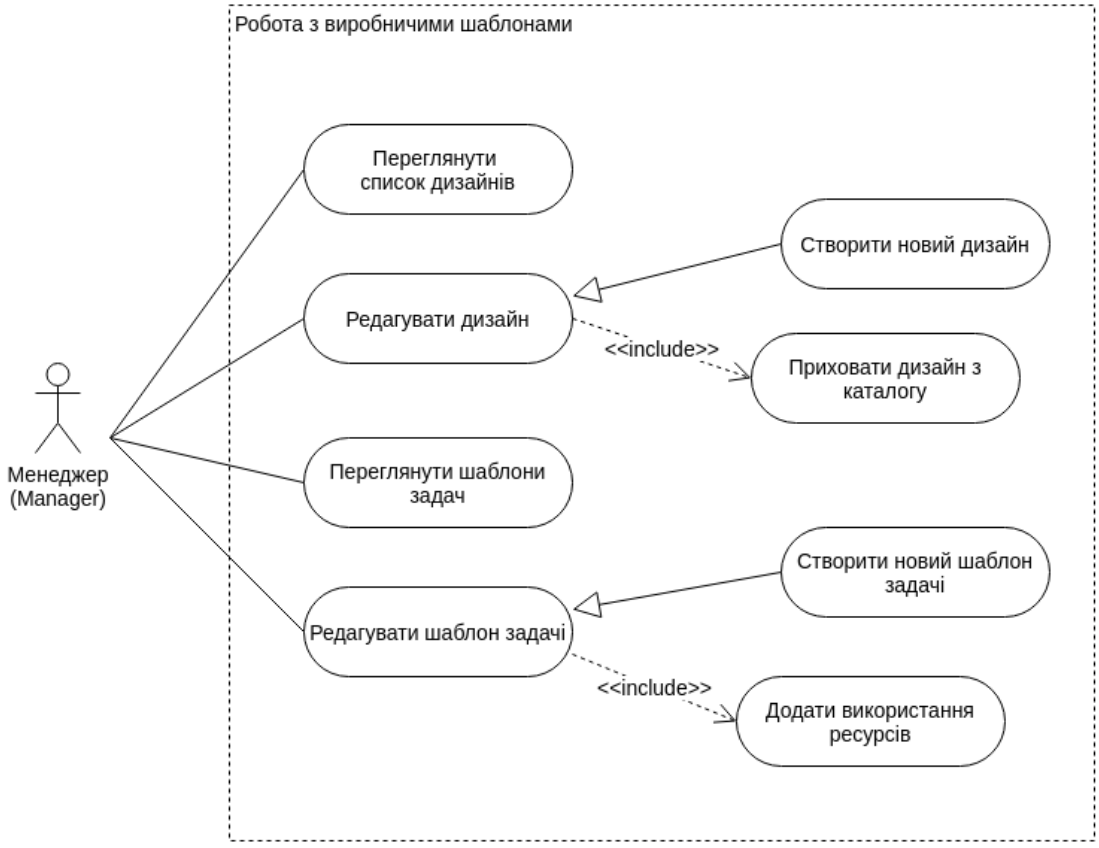

Рисунок 2.12 – Варіанти взаємодії менеджера з виробничими шаблонами

2.2 Моделювання бази даних

Метою даного етапу є визначення сутностей, які необхідно постійно зберігати в базі даних для нормального функціонування розроблюваної системи.

Розробка бази даних буде здійснена по принципу «Code first»: спочатку реалізуються класи на мові програмування C Sharp, далі EF Core автоматично створює таблиці при підключенні до пустої бази даних [4].

2.2.1 Допоміжні класи

Виходячи з вимог до системи, необхідно реалізувати такий набір допоміжних класів, які не будуть приводитися до таблиць у базі даних:

– Person – містить інформацію про особу, а саме: id, прізвище, ім’я, по батькові, номер телефону, електронну пошту, пароль, дату реєстрації;

– Dimensions – містить інформацію про розміри предмету меблів: ширина, довжина, висота, одиниці виміру;

– Order – містить основну інформацію про замовлення: власний id, id покупця, вартість, передплата, статус замовлення, дата подачі замовлення, дата початку робіт, дата завершення робіт, дата отримання;

– WorkTask – містить інформацію про задачу: власний id, id робітника, оплата, статус, дата створення задачі, дата початку робіт, дата завершення робіт;

– Notification – сповіщення, містить такі поля: id, дату створення, текст, відмітку про прочитання.

2.2.2 Перераховні типи

Необхідно реалізувати такі перераховні типи:

– WorkerRole – роль робітника, можливі значення: unknown (невідомо), management (менеджер), sewing (швець/швачка), carpentry (столяр), assembly (збиральник), upholstery (оббивщик);

– FurnitureType – тип предмету меблів, асортимент майстерні є досить фіксованим, тому це можна записати у вигляді фіксованого перераховного типу: unknown (невідомо), sofa (диван), minisofa (міні-диван), ottoman (тахта), corner (куток), armchair (крісло), pouffe (пуф);

– ResourceType – тип ресурсу, можливі значення: resource (ресурс невизначеного типу), part (деталь), material (всі матеріали окрім тканин для обшивки), fabric (тканина);

– SlotName – назва слота, використовується в шаблонах задач, коли найменування ресурсу можна визначити лише з замовлення, можливими значеннями є: unknown (невідомо), fabric (тканина), decor (декор);

– MeasureUnits – одиниці виміру, можливими значеннями є: unknown (поштучно), mm (міліметр), cm (сантиметр), m (метр), m2 (квадратний метр), m3 (кубічний метр), L (літр), Kg (кілограм), g (грам);

– OrderStatus – стан замовлення, можливими значеннями є: unknown (невідомо), submitted (подано), delayed (затримується), working (робота почалася), finished (завершено), received (отримано), rejected (відхилено), impossible (неможливо виконати);

– TaskStatus – стан задачі, можливими значеннями є: uknown (невідомо), assigned (призначено), rejected (відхилено), working (робота почалася), finished (завершено), paid (оплачено), impossible (неможливо).

2.2.3 Табличні сутності

Далі необхідно реалізувати такий перелік сутностей, що утворюють таблиці:

1) Customer – наслідує всі поля класу Person та додає: назву організації, відсоток знижки;

2) Worker – наслідує всі поля класу Person та додає назви ролей;

3) Supplier – наслідує всі поля класу Person та додає назву організації;

4) Color – містить інформацію про колір матеріалу або тканини: id, назву, RGB-код, дату створення;

5) FabricType – містить інформацію про тип тканини: id, назву, опис, дату створення;

6) Resource – містить інформацію про ресурс: id, колір, тип тканини, фото, найменування, опис, назва слота (тканина, декорації), одиниці виміру, точність виміру, ціна за одиницю;

7) StoredResource – містить інформацію про збережуваний на складі ресурс: id, зовнішній ідентифікатор (використовується при запиті до

постачальника), реальна кількість, нормальний запас ресурсу, період поповнення запасу, дата останньої перевірки, дата останнього поповнення;

8) SupplyOrder – містить інформацію про запит на постачання ресурсу: власний id, id ресурсу, id постачальника, назва ресурсу, зовнішній ідентифікатор, очікувана ціна, очікувана кількість, реальна ціна, реальна кількість, стан запиту, дата створення, дата отримання ресурсу;

9) Design – містить інформацію про дизайн меблів: id, найменування, тип меблів, базова ціна за одиницю, відображення в каталозі, можливість трансформування, розміри в складеному стані, розміри в розгорнутому стані (якщо трансформується), використання тканини для обшивки, використання декораційного матеріалу (якщо є);

10) TaskPrototype – містить інформацію про шаблон задачі: власний id, id дизайну, тип роботи, пріоритет виконання, текстовий опис, використання ресурсів;

11) ProdOrder – виробниче замовлення, наслідує всі поля класу Order та додає: кількість, id дизайну, id тканини, id декораційного матеріалу (якщо є), фіксовану ціну за одиницю;

12) ProdFurniture – опис виготовлених меблів, містить такі поля: власний id, id дизайну, id тканини, id декораційного матеріалу (якщо є), дата створення, дата початку робіт, дата завершення робіт;

13) ProdTask – виробнича задача, наслідує всі поля класу WorkTask та додає такі поля: id меблів, id прототипу задачі;

14) RepairOrder – ремонтне замовлення, наслідує всі поля класу Order та додає: текстовий опис, посилання на фотографії;

15) RepairTask – ремонтна задача, наслідує всі поля класу WorkTask та додає: id замовлення, тип роботи, текстовий опис;

16) ResourceQuantity – кількість використаного ресурсу, має такі поля: власний id, id ресурсу (якщо є), назва слота, запланована кількість, фактична кількість;

17) ProdNotification – виробниче сповіщення, наслідує всі поля класу Notification та додає id виробничого замовлення;

18) RepairNotification – ремонтне сповіщення, наслідує всі поля класу Notification та додає id ремонтного замовлення;

19) WorkerPaycheck – заробітний чек, що формується на основі виконаних задач, включає такі поля: власний id, id робітника, початок та кінець розрахункового періоду, сума виплати, відмітка про отримання.

2.2.4 Візуалізація сутностей та зв'язків між ними

Для полегшення сприйняття діаграму класів розділено на фрагменти:

– фрагмент «Люди» (див. рис. 2.13);
– фрагмент «Ресурси» (див. рис. 2.14);
– фрагмент «Шаблони» (див. рис. 2.15);
– фрагмент «Замовлення» (див. рис. 2.16);
– фрагмент «Задачі» (див. рис. 2.17).

На рисунку 2.13 показано, що класи Worker, Customer, Supplier наслідують від класу Person; об'єкт класу Worker має декілька ролей що є текстовими рядками з фіксованого набору значень, визначеного в класі WorkerRole; об'єкт класу Worker має 0..n заробітних чеків (WorkerPaycheck).

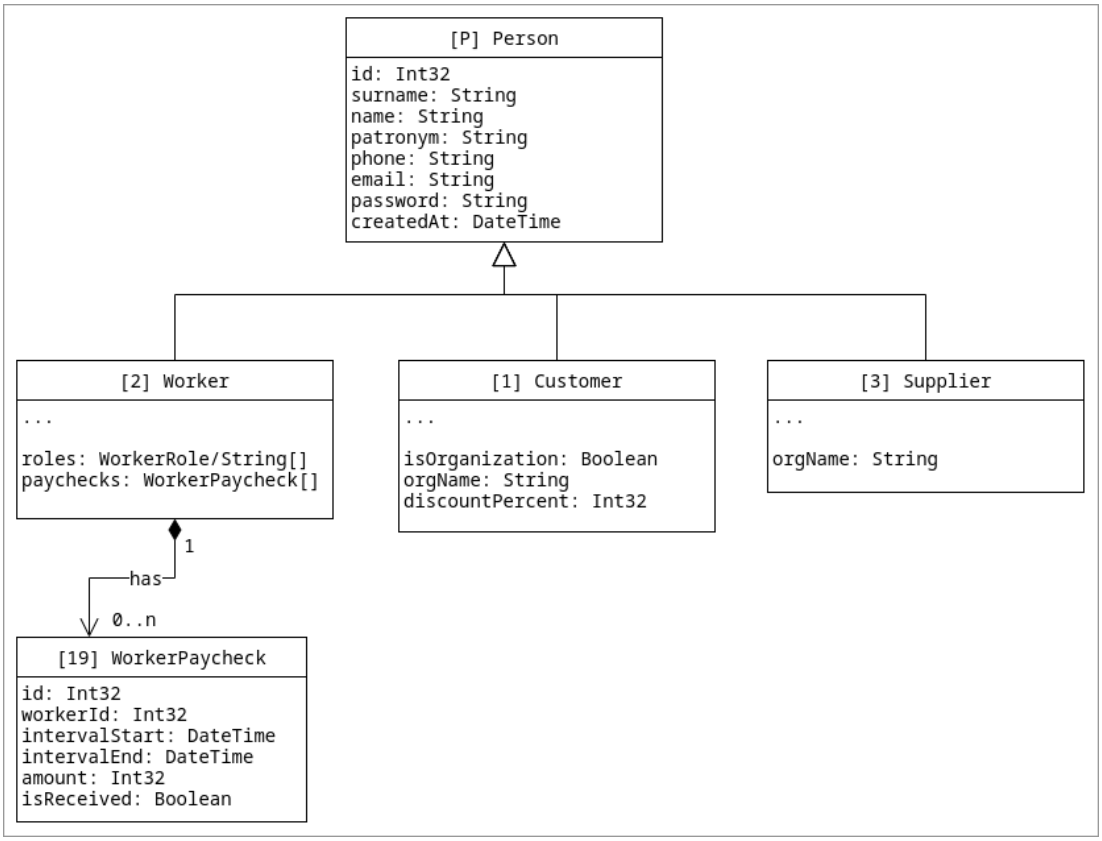

Рисунок 2.13 – Фрагмент моделі класів «Люди»

На рисунку 2.14 показано, що 0..n ресурсів (Resource) можуть опціонально мати 1 колір та/або 1 тип тканини; ресурс може опціонально мати 1 запис про ресурс на складі (StoredResource); 0..n ресурсів можуть постачатися одним постачальником (Supplier); також для кожного ресурсу можуть існувати 0..n запитів на постачання ресурсу (SupplyOrder), кожен з яких направляється до 1 постачальника (Supplier).

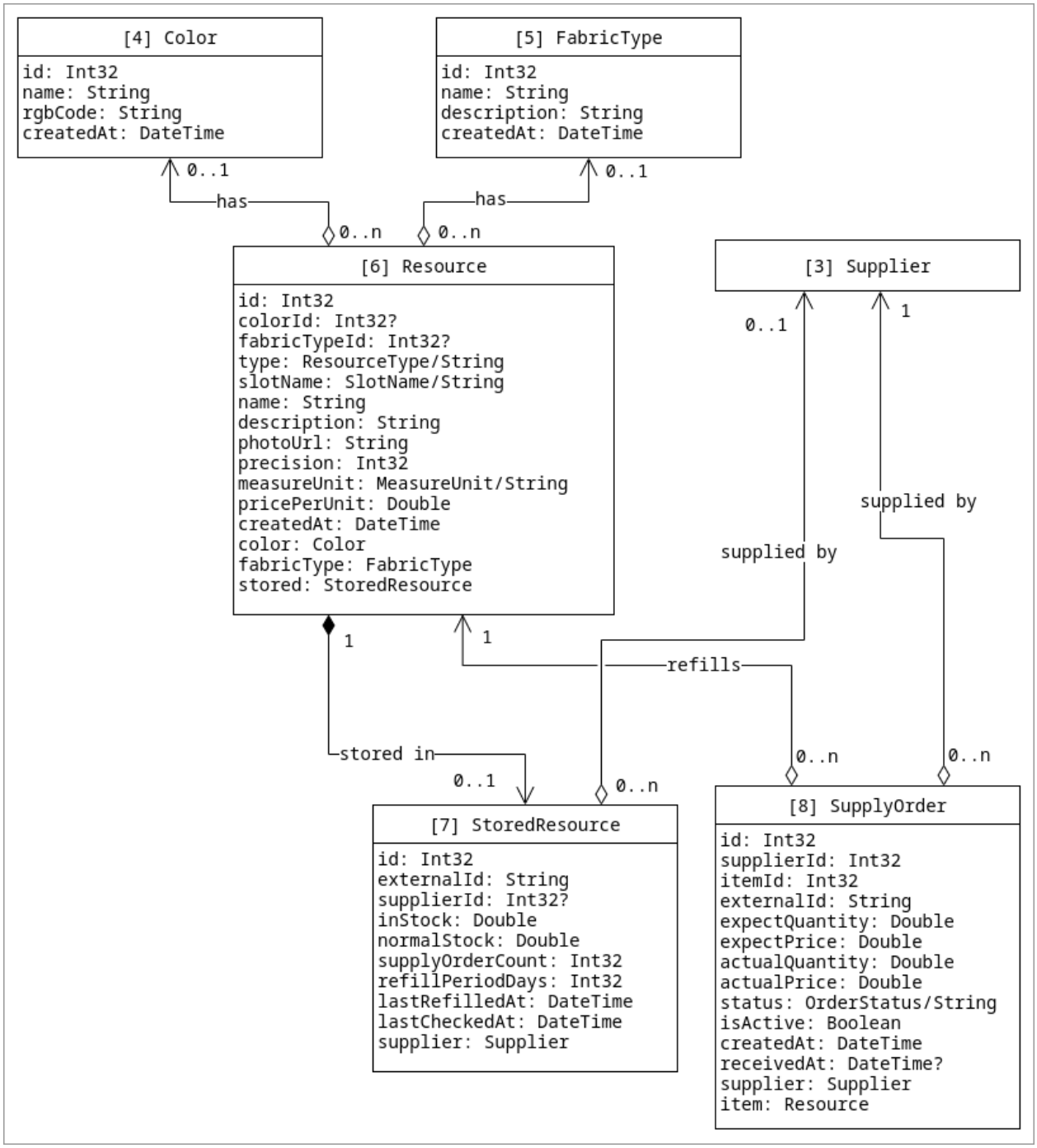

Рисунок 2.14 – Фрагмент моделі класів «Ресурси»

На рисунку 2.15 показано, що дизайн меблів (Design) може мати від 1 до 2 розмірів (у складеному стані та опціонально в розкладеному стані); дизайн меблів може мати 0..n шаблонів задач (TaskPrototype); шаблон задачі передбачає використання 0..n ресурсів (ResourceQuantity) з зазначенням запланованої кількості використання та опціонально найменування ресурсу (якщо не зазначено ресурс, то використовується слот «тканина» або «декор»).

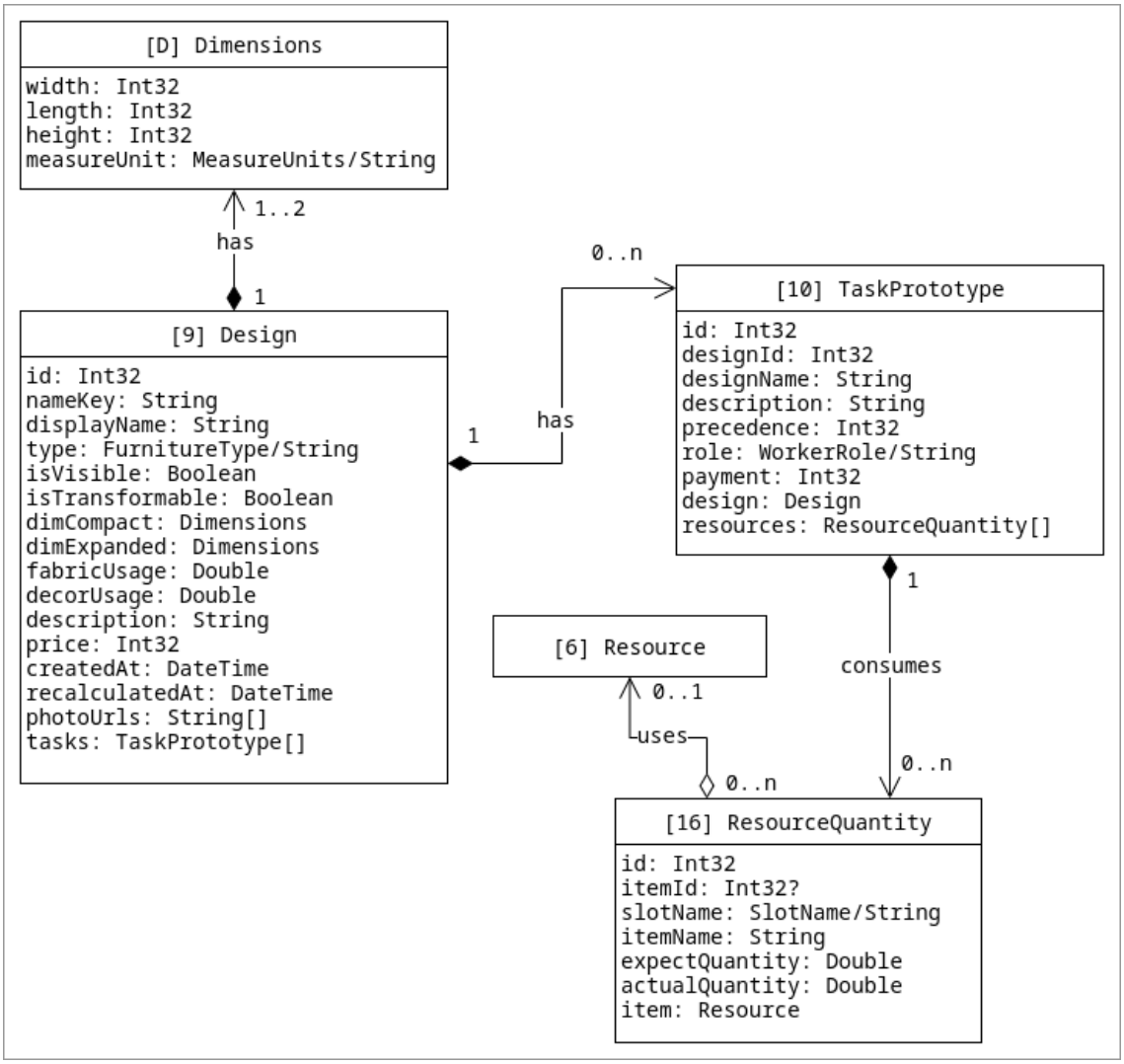

Рисунок 2.15 – Фрагмент моделі класів «Шаблони»

На рисунку 2.16 показано, що замовлення (Order) та наслідуючі від нього класи ProdOrder та RepairOrder подаються покупцем (Customer). Виробниче замовлення (ProdOrder) зазначає 1 дизайн меблів (Design) який необхідно реалізувати, 1 найменування тканини (Fabric) яку необхідно використати, опціонально 1 найменування декораційного матеріалу.

На основі виробничого замовлення створюються 0..n меблів (ProdFurniture) які виготовляються на основі 1 дизайну та мають мо мірі виконання 1 тканину та опціонально 1 декораційний матеріал. Виробниче замовлення може мати 0..n виробничих сповіщень (ProdNotification). Ремонтне замовлення (RepairOrder) перетворюється на 0..n ремонтних задач (RepairTask), та може мати 0..n ремонтних сповіщень (RepairNotification).

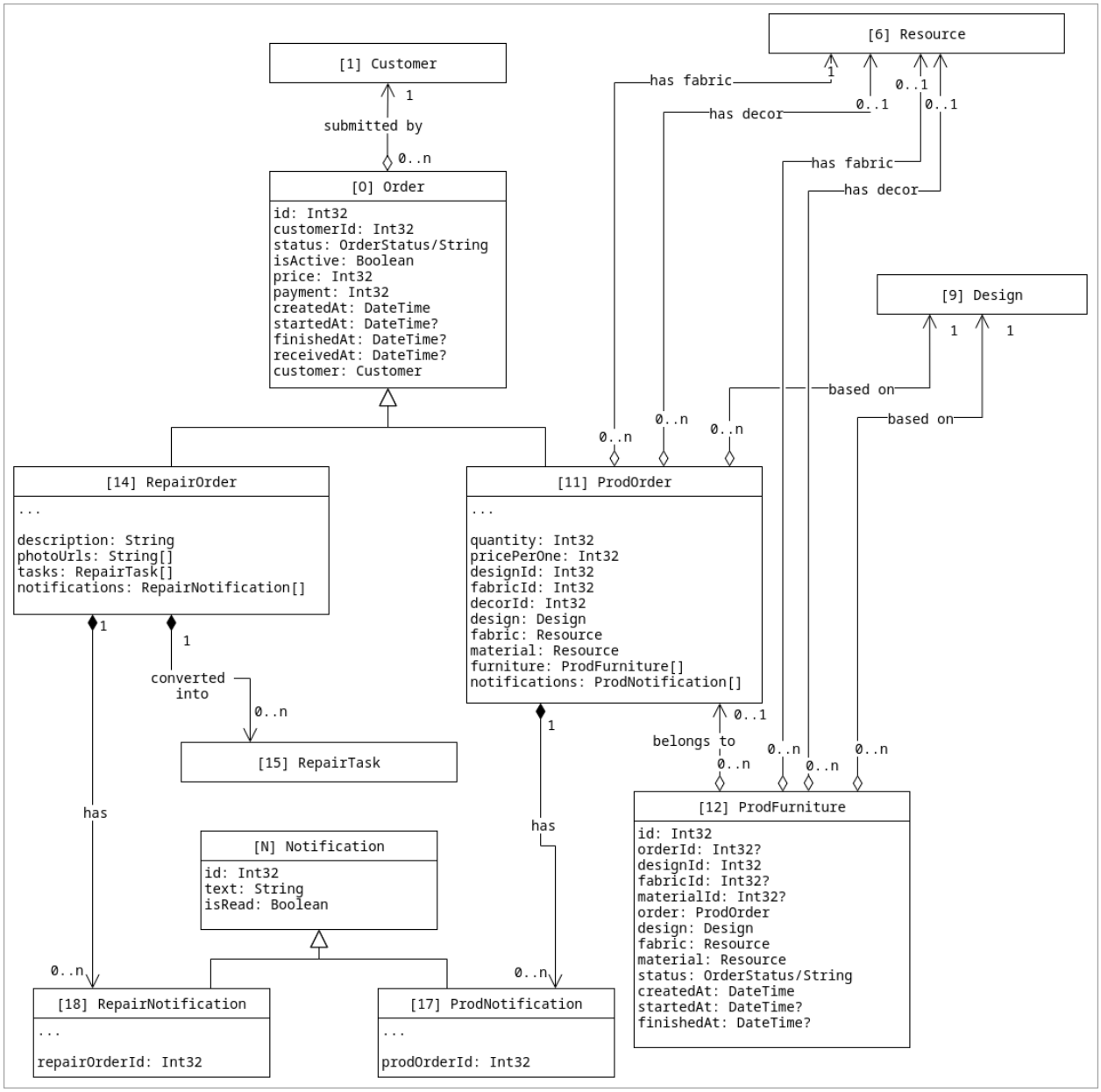

Рисунок 2.16 – Фрагмент моделі класів «Замовлення»

На рисунку 2.17 показано, що задача (WorkTask) передбачає використання 0..n ресурсів (ResourceQuantity). До декількох задач може бути опціонально призначений 1 робітник (Worker). Виробничі задачі (ProdTask) та ремонтні задачі (RepairTask) наслідують від класу задачі (WorkTask). Декілька виробничих задач (ProdTask) прив'язуються до 1 предмету меблів (ProdFurniture), кожна задача виконується згідно 1 шаблону (TaskPrototype). Декілька ремонтних задач (RepairTask) прив'язуються до 1 ремонтного замовлення (RepairOrder).

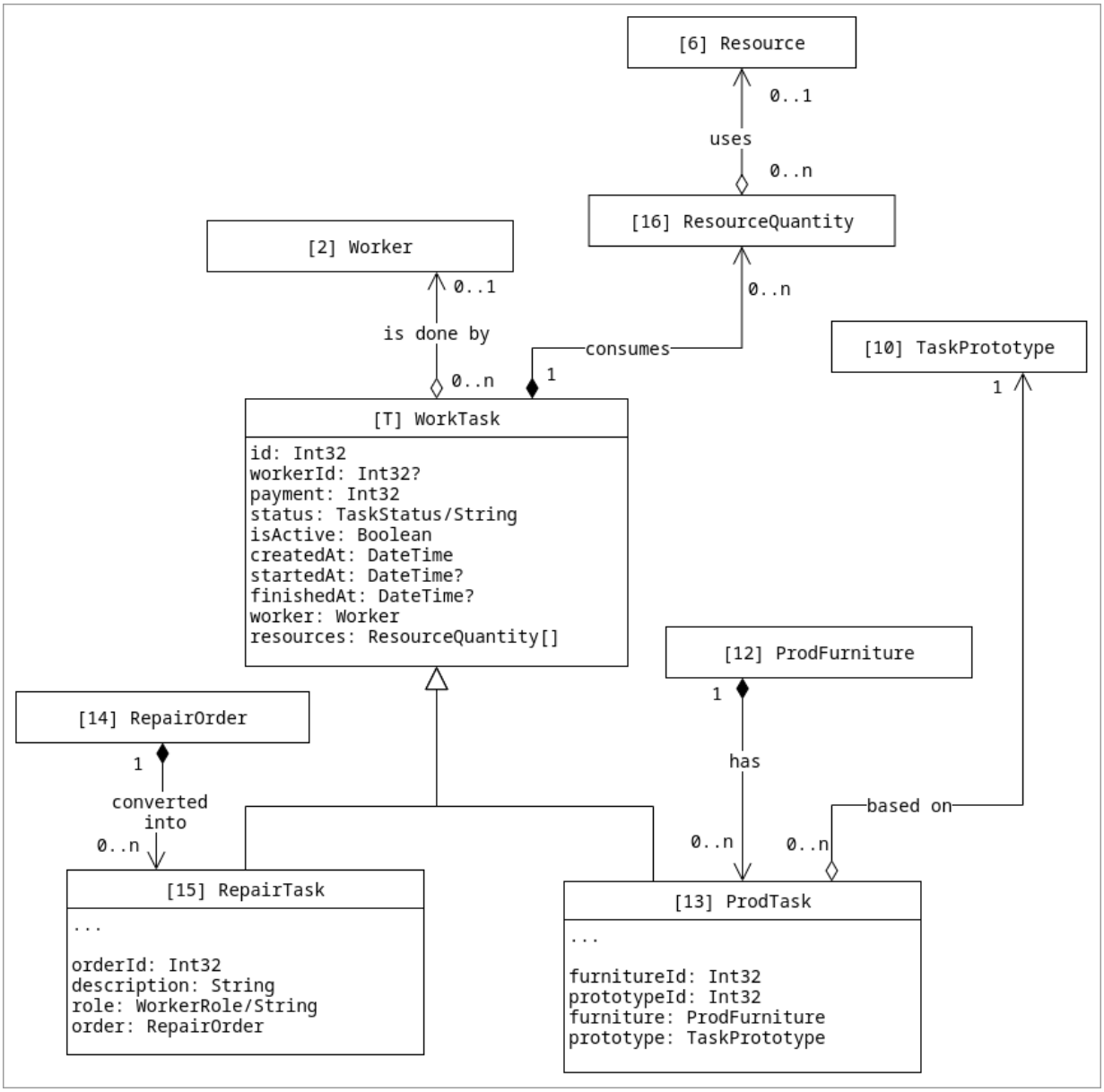

Рисунок 2.17 – Фрагмент моделі класів «Задачі»

2.3 Моделювання WebAPI

На основі переліку варіантів використання, сформованого в розділі 2.1, формується набір можливих дій WebAPI. З урахуванням користувачів системи та спільних дій пропонується розділити структуру WebAPI на такі зони (див. табл. 2.1). Далі для кожної зони визначаються вкладені дії.

Таблиця 2.1 – Зони WebAPI

№ в/п	Префікс	Доступ	Призначення	Детальний опис
1	/auth	для всіх	Реєстрація, вхід в обліковий запис, вихід з облікового запису	див. табл. Б.1
2	/catalog	для всіх	Перегляд каталогу дизайнів, тканин, декораційних матеріалів	див. табл. Б.2
3	/customer	покупець	Перегляд власного профілю, подача замовлень на виробництво, подача замовлень на ремонт, перегляд стану замовлень, перегляд сповіщень	див. табл. Б.3
4	/worker	робітник	Перегляд власного профілю, перегляд власних виробничих задач, перегляд власних ремонтних задач, генерація заробітного чека	див. табл. Б.4
5	/manager	менеджер	Перегляд власного профілю, перегляд профілів користувачів, перегляд дизайнів меблів, редагування дизайну, перегляд шаблонів задач, редагування шаблонів задач, перегляд виробничих замовлень, перегляд ремонтних замовлень, зміна статусу замовлення, створення виробничих задач, створення ремонтних задач, призначення виконавців задач	див. табл. Б.5
6	/resources	робітник, менеджер	Перегляд ресурсів на складі, створення нового ресурсу, редагування ресурсу, перегляд кольорів, редагування кольорів, перегляд типів тканин, редагування типів тканин	див. табл. Б.6

Для прикладу в таблиці 2.2 наведено опис WebAPI, що відповідає за створення та перегляд нового дизайну меблів.

Таблиця 2.2 – WebAPI для створення та перегляду дизайну меблів

Метод	**Шлях**	**Структура запиту**	**Структура відповіді**

POST	/manager/designs /create	body: {
		"type": String,
		"nameKey": String,
		"displayName": String,
		"description": String,
		"isVisible": Boolean,
		"isTransformable": Boolean,
		"dimCompact": {
		"width": Number,
		"length": Number,
		"height": Number
		},
		"dimExpanded": {
		"width": Number,
		"length": Number,
		"height": Number
		}
		}
			{
			"id": Number,
			"success": Boolean,
			"message": String,
			"badFields": String[]
			}

GET	/manager/designs /view/:nameKey	path: {
		"nameKey": String
		}
			body: {
			"id": Number,
			"type": String,
			"nameKey": String,
			"displayName": String,
			"description": String,
			"isVisible": Boolean,
			"isTransformable": Boolean,
			"price": Number,
			"dimCompact": {
			"width": Number,
			"length": Number,
			"height": Number
			},
			"dimExpanded": {
			"width": Number,
			"length": Number,
			"height": Number
			}
			}

2.4 Модель компонентів системи

Розроблювана серверна частина системи потребує таких бібліотек:

– середовище виконання .NET [11];
– EFCore для абстрактної взаємодії з базою даних [4];
– Npgsql для взаємодії саме з PostgreSQL [12];
– AspNetCore для реалізації WebAPI [3].

Необхідно розробити власні допоміжні модулі:

– сутності даних (Fwsh.Common);
– адаптер бази даних (Fwsh.Database);
– засоби логування (Fwsh.Logging);
– інші допоміжні класи та методи виносяться в Fwsh.Utils.

Головний модуль – Fwsh.WebApi. Ієрархію модулів серверної частини показано на рисунку 2.18. Варто зазначити, що модулі серверної частини поділяються на 4 рівня:

– бібліотеки (Library);
– утиліти (Utils);
– рівень даних (Data);
– прикладний рівень (Application).

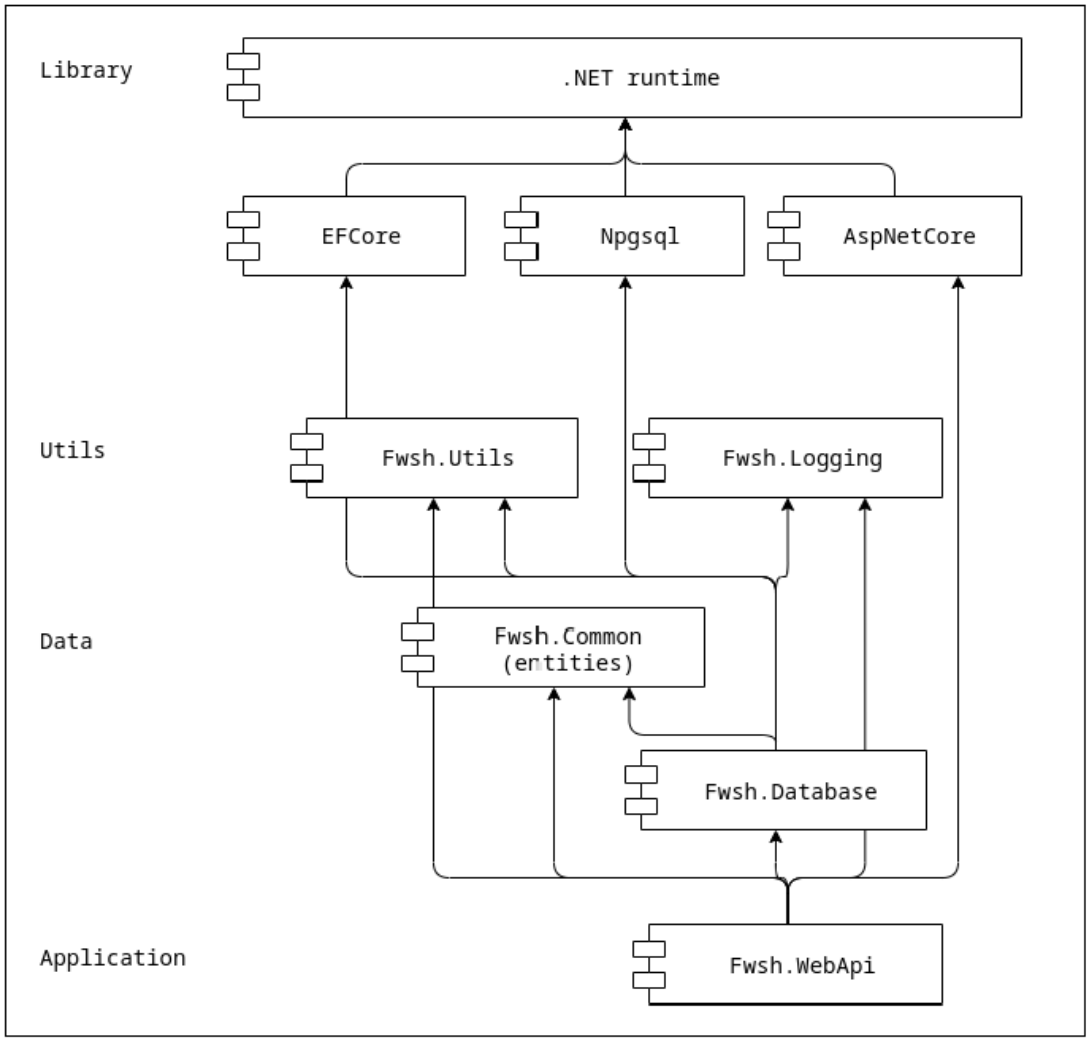

Рисунок 2.18 – Модулі серверної частини

Розроблювана клієнтська частина складається з трьох головних модулів (CustomersPanel, WorkersPanel, ManagersPanel) та допоміжного модуля Common, що мають однаковий набір залежностей:

– vue;
– vue-router;
– axios;
– qs.

Ієрархію компонентів клієнтської частини показано на рисунку 2.19.

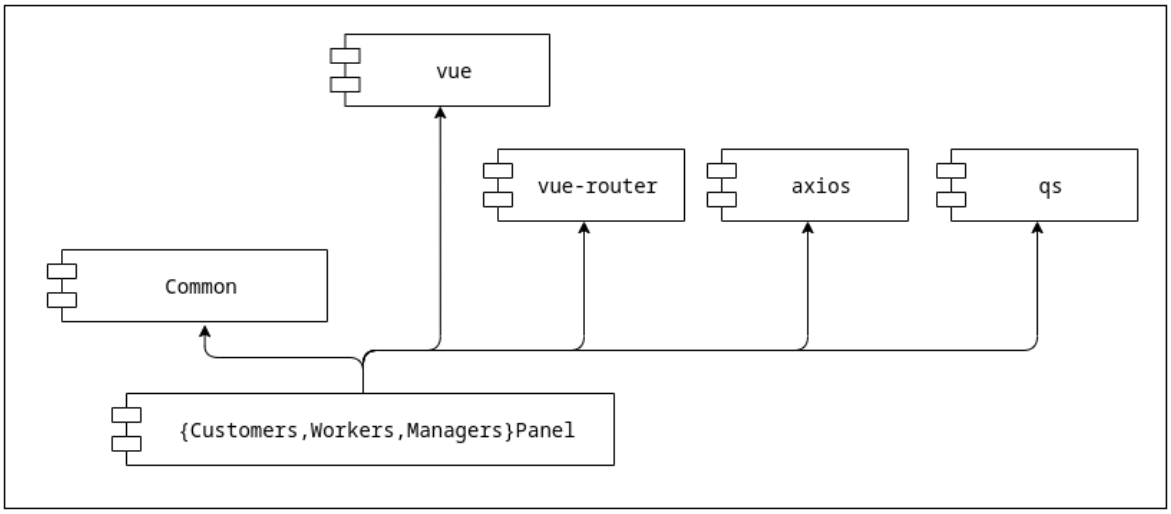

Рисунок 2.19 – Модулі клієнтської частини

2.5 Модель розгортання системи

При програмній реалізації системи та її випробуванні передбачається, що система буде розгорнута локально (див. рис. 2.20). Клієнтська частина, серверна частина та база даних буде знаходися на одному пристрої. Сервірування компонентів клієнтської частини буде здійснюватися локально з використанням dev-сервера, що входить до набору інструментів Vite [13]. Взаємодія клієнта та сервера WebAPI буде здійснюватися локально по протоколу HTTP. Сервірування файлів контенту, як-от фотографії меблів, буде здійснюватися засобами сервірування статичних файлів, що входять до фреймворку ASP.NET Core [3].

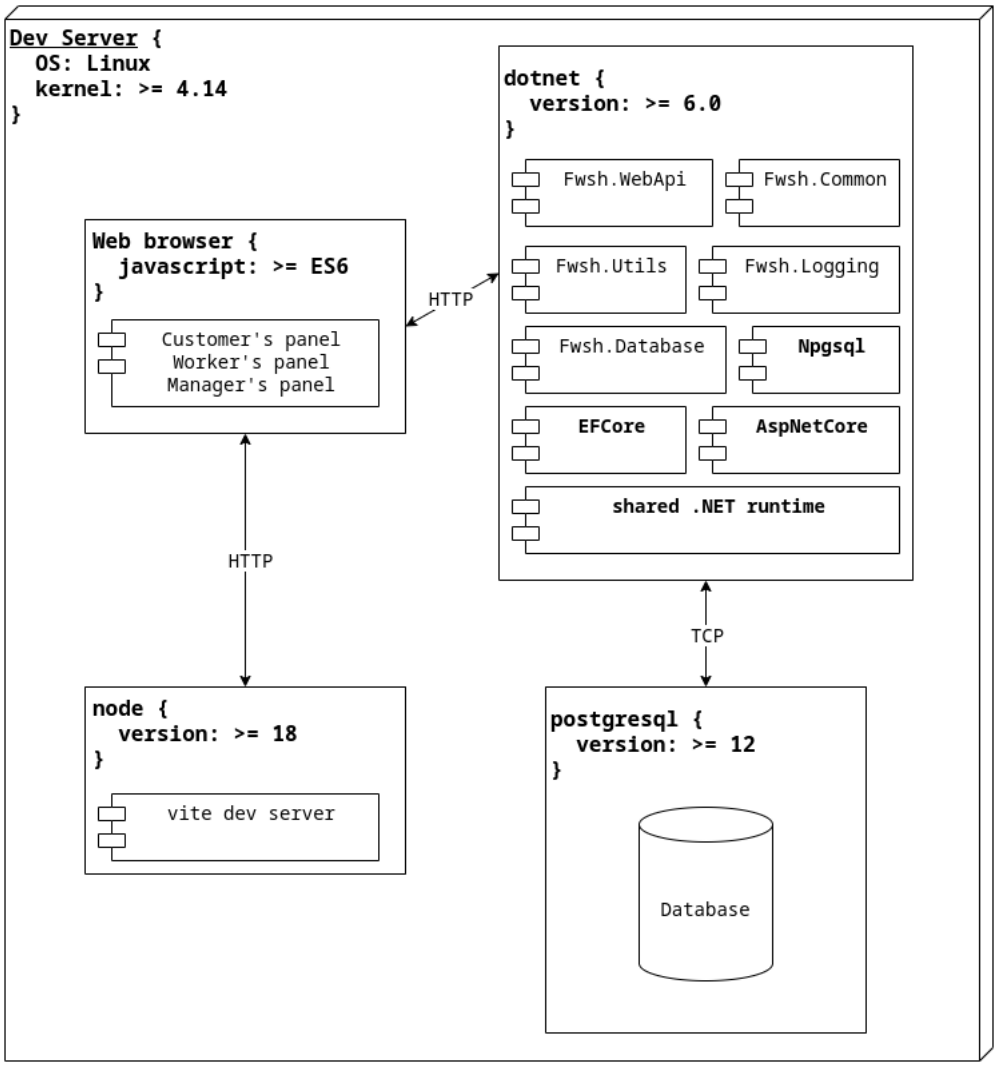

Рисунок 2.20 – Розгортання системи у Development-середовищі

На рисунку 2.21 наведено модель розгортання системи в виробничому (Production) середовищі. Передбачається, що сервер WebAPI буде побудований в форматі «self-contained» [5]. Для уникнення необхідності вводити обробку SSL-сертифікатів у код, сервер WebAPI буде працювати через проксі-сервер. Клієнтські частини будуть побудовані та розгорнуті на віртуальних хостах Apache [14].

Взаємодія між комп'ютерами клієнтів (покупців, робітників, менеджерів) та сервером буде здійснюватися виключно по протоколу HTTPS. Будуть використовуватися SSL-сертифікати для кожного віртуального хоста [15]. Під комп'ютером клієнта мається на увазі будь-який стаціонарний чи мобільний пристрій, що підключений до мережі та має web-браузер з підтримкою Javascript версії ES6 або вище.

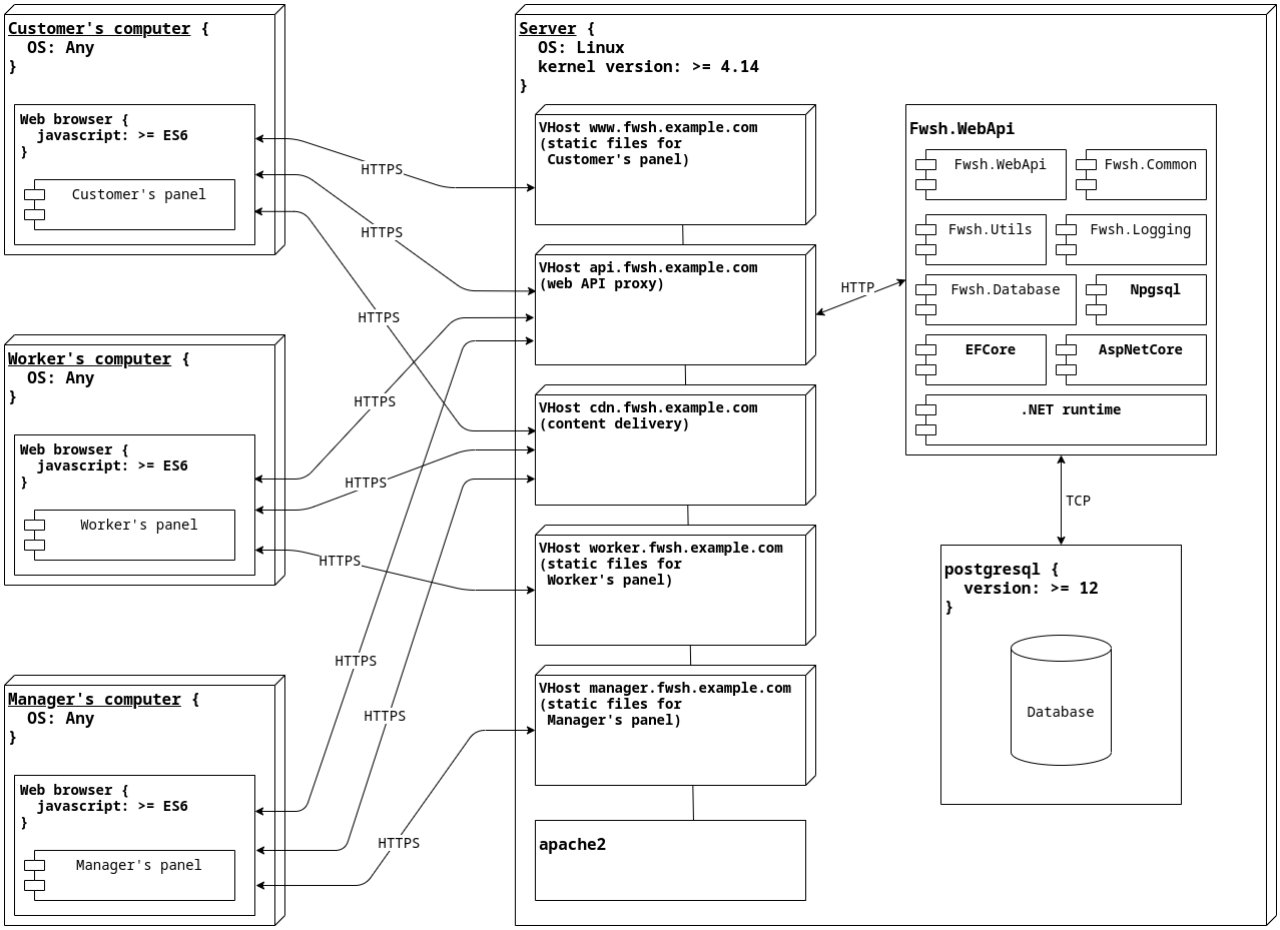

Рисунок 2.21 – Розгортання системи у Production-середовищі

Для можливості знаходження сервера по доменному імені необхідно придбати домен або налаштувати в локальній мережі сервер DNS. Але це не входить в перелік задач роботи, тому обмежимося додаванням доменного імені сервера та його IP-адреси в файл hosts [16].

Висновки до розділу 2

На основі вимог до системи та технічного завдання визначено варіанти використання системи, перелік сутностей бази даних, схему WebAPI, модель компонентів системи та модель розгортання системи. Зазначені моделі та схеми є основою для програмної реалізації системи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Реалізація бази даних

База даних реалізується по загальновідомому принципу «Code first», що передбачає:

– написання класів сутностей на мові програмування C Sharp [17];

– створення класу-фасаду, що наслідує DbContext, додавання до класу-фасаду колекцій DbSet<TData>, де TData – тип класу, для якого треба створити таблиці в базі даних [17];

– конфігурація підключення до специфічної бази даних [18].

Приклад класу сутності (дизайн меблів) наведено в додатку В.1. Клас-фасад наведено в додатку В.2. Конкретну реалізацію фасаду для PostgreSQL наведено в додатку В.3.

База даних буде створена автоматично при першому запуску застосунку [19]. Автоматично згенеровані SQL-запити для створення бази даних наведено в додатку В.4. Для заповнення бази даних тестовим набором записів створено заповнювачі, приклад наведено в додатку В.5.

3.2 Реалізація сервера системи

Сервер WebAPI системи реалізується з використанням фреймворку ASP.NET Core. За загальноприйнятою інструкцією, всі доступні дії WebAPI реалізуються у вигляді методів класу, що наслідує від контролера (ControllerBase), дії групуються в один клас-контролер за префіксом [20]. Окрім того, контролери можна групувати в простори імен за назвою зони. Наприклад, дії /resources/colors/list та /resources/colors/create будуть у класі ColorController у просторі імен Resources. Інший приклад: дії /manager/designs/list та /manager/designs/create будуть у класі DesignController у просторі імен Manager. Приклад коду контролера DesignController наведено в додатку Г.1.

Контролери зазвичай видають результат не лише на основі запиту. Потрібно підключитися до бази даних, перевірити права клієнта, записати яке-небудь повідомлення в лог. Це все є залежностями, і для ін’єкції залежностей в ASP.NET Core передбачені так звані сервісні контейнери, що управляють створенням контролерів та проміжних обробників (Middleware) та ін’єкцією залежностей [21]. Для того, щоб визначити залежність, треба додати її як параметр конструктора (див. додаток Г.1), а щоб надати реалізацію залежності, треба зареєструвати її в методі ConfigureServices класу Startup (див. додаток Г.4) [21].

При реалізації WebAPI елементи шляху, параметри та тіло запиту перетворюються в параметри методів на мові програмування C Sharp [20]. Немає обмежень, які саме типи даних приймати та повертати [20]. Тому, пропонується ввести базовий клас Request (запит) та всі тіла запиту сприймати як похідні класи від Request. У класах похідних від Request буде здійснюватися валідація вхідних даних та створення або оновлення збережуваного об’єкта класу-сутності, залежно від дії WebAPI в якій використано даний Request. Приклад коду запиту DesignRequest наведено в додатку Г.2.

Окрім того, оскільки сутності можуть бути представлені по-різному для різних категорій користувачів, можуть мати компактний вигляд (наприклад, при перегляді списку), можуть містити чутливі дані (пароль) або циклічні об’єктні посилання що ламають JSON-серіалізатор, пропонується ввести клас Result (результат) та винести логіку творення результату в класи похідні від Result. Приклад коду результату DesignResult наведено в додатку Г.3. Модель використовуваних класів для дії «Створити дизайн меблів» наведено на рисунку 3.1. Модель використовуваних класів для дії «Переглянути дизайн меблів» наведено на рисунку 3.2

Важливим етапом реалізації сервера на фреймворку ASP.NET Core є загальні налаштування, що зазвичай виносяться у клас Startup. Там вказується послідовність обробки запитів, залежності, які впроваджуються у контролери тощо [22]. Код класу Startup наведено в додатку Г.4. Робота сервера в спрощеному вигляді показана на рисунку 3.3.

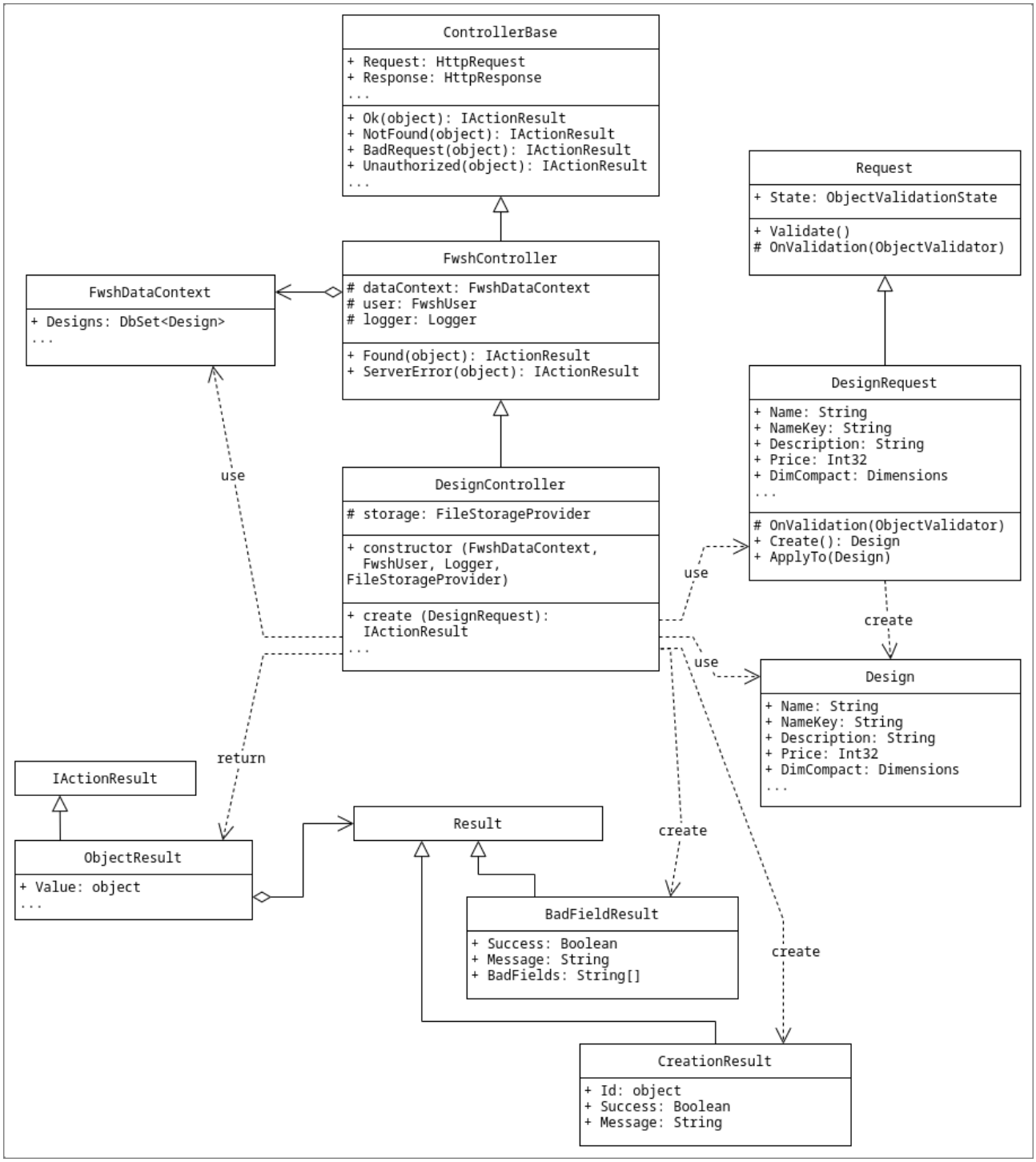

Рисунок 3.1 – Використовувані класи при створенні дизайну меблів

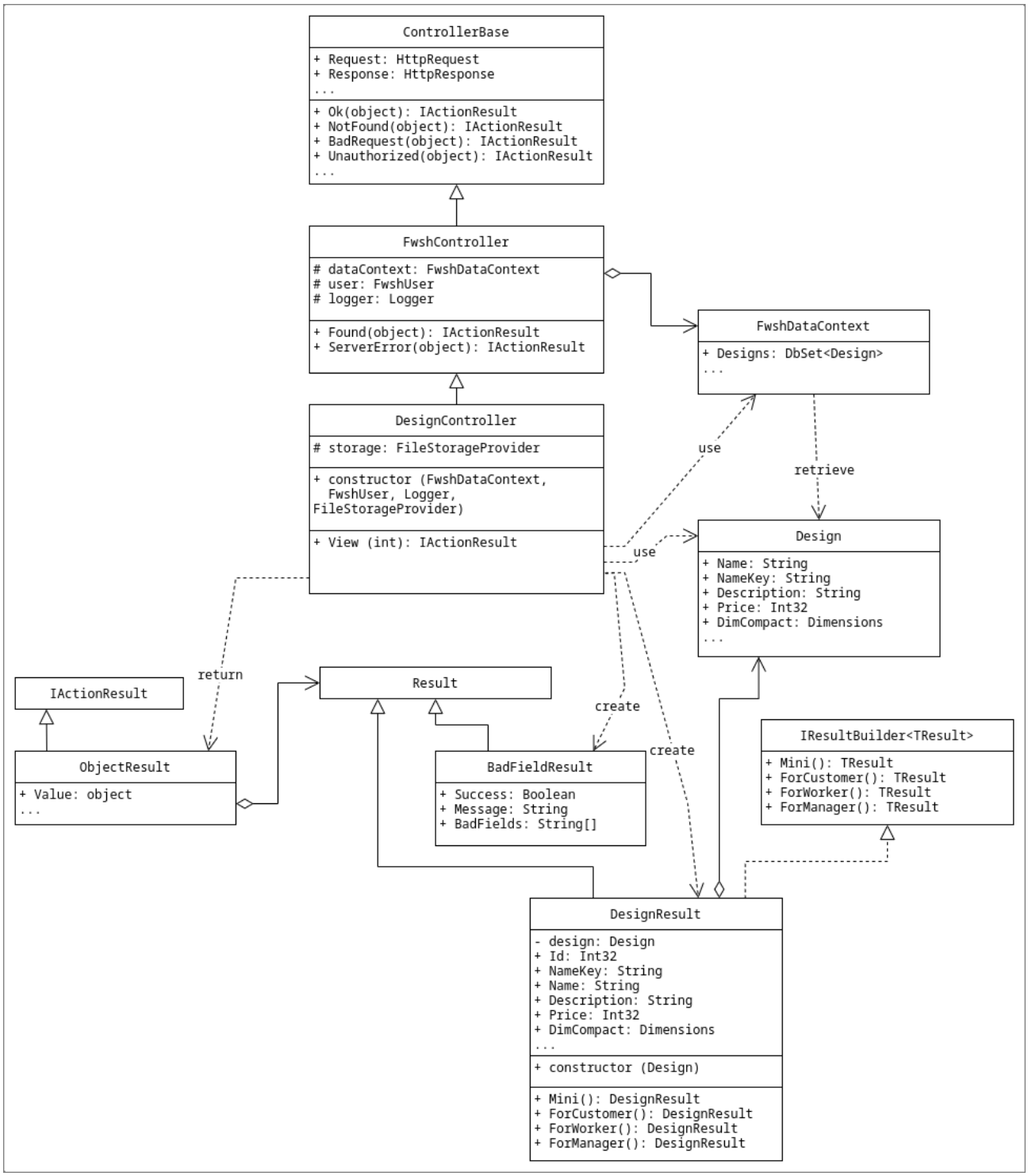

Рисунок 3.2 – Використовувані класи при перегляді дизайну меблів

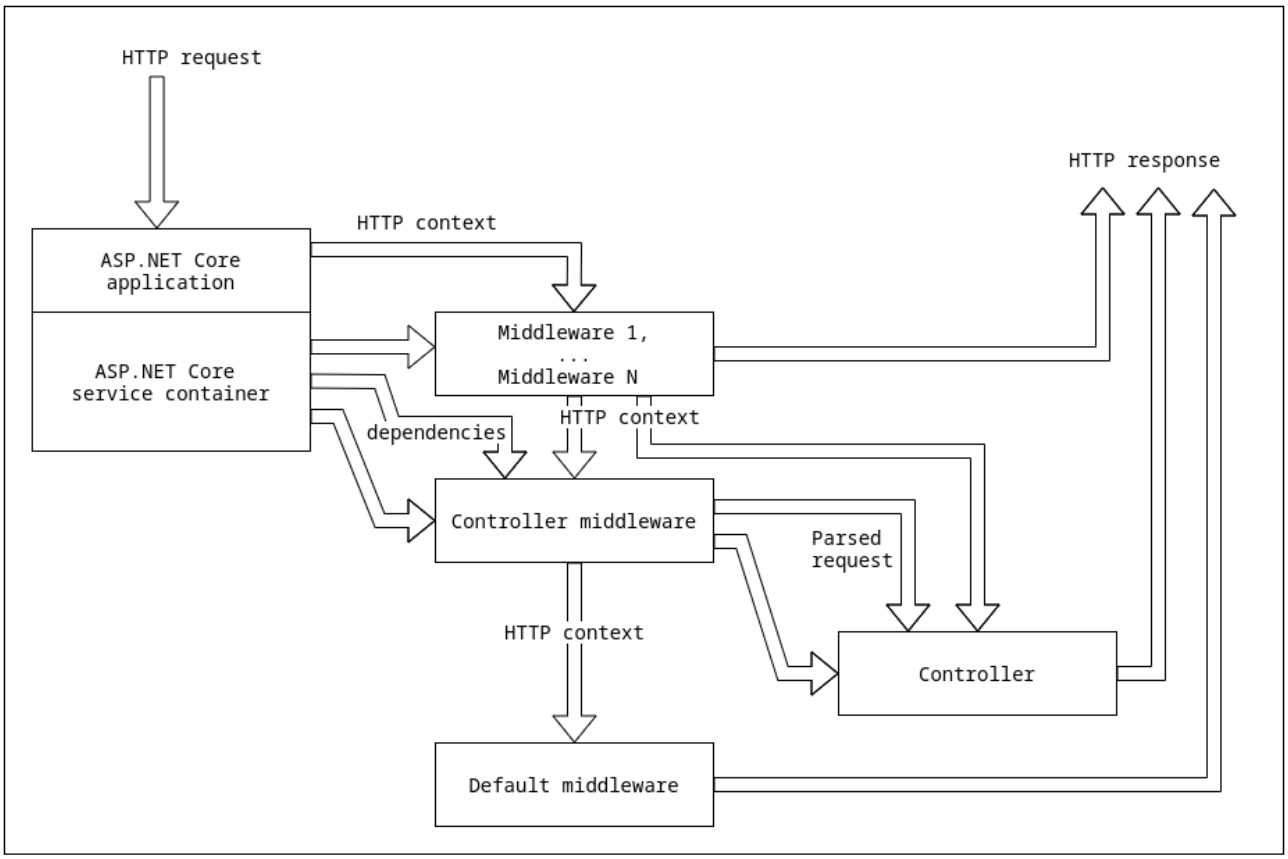

Рисунок 3.3 – Робота сервера WebAPI в спрощеному вигляді

Отже, вище було розглянуто принципи реалізації сервера WebAPI з використанням загальновідомих контролерів (Controller) та власного варіанту запитів (Request) та результатів (Result). Повну версію вихідного коду сервера наведено в репозиторії [23].

3.3 Випробування сервера системи

Випробування розробленого сервера WebAPI відбувається в напів-автоматичному режимі у форматі сценаріїв з використанням інструмента Apinb [24]. Для прикладу в таблиці 3.1 наводиться сценарій створення дизайну меблів. Метою сценарію є перевірити реалізацію варіантів використання «Створення дизайну меблів» для менеджера, та «Перегляд дизайну меблів» для всіх користувачів.

Таблиця 3.1 – Сценарій створення дизайну меблів

№ в/п	Метод	Відносний шлях	Вміст запиту

1	POST	/auth/manager/login	{
			"phone": "+380930268871",
			"password": "ASz!P@Iuqg"
			}

2	POST	/manager/designs/create	{
			"type": "ottoman",
			"nameKey": "canabre",
			"displayName": "Canabre",
			"description": "Compact ottoman couch. Elegant curves and ... Lorem ipsum sit amet blah blah we need 100+ letters blah blah blah",
			"isVisible": true,
			"isTransformable": true,
			"dimCompact": {
			"width": 166,
			"length": 100,
			"height": 88
			},
			"dimExpanded": {
			"width": 166,
			"length": 178,
			"height": 88
			}
			}

3	GET	/catalog/designs/view /canabre	-

Результати виконання сценарію (відповіді сервера WebAPI) наведені на рисунках 3.4, 3.5, 3.6. Можна побачити, що сервер може прийняти запит на створення нового дизайну меблів та потім показати дизайн меблів при запиті на перегляд. Так само здійснюється випробування сервера WebAPI для інших варіантів використання.

Рисунок 3.4 – Вхід в обліковий запис менеджера

Method URL

post /manager/designs/create

Request payload


```
{
  "type": "ottoman",
  "nameKey": "canabre",
  "displayName": "Canabre",
  "description": "Compact ottoman couch. Elegant curves and ... Lorem ipsum sit amet
blah blah we need 100+ letters blah blah blah",
  "isVisible": true,
  "isTransformable": true,
  "dimCompact": {
    "width": 166,
    "length": 100,
    "height": 88
  },
  "dimExpanded": {
    "width": 166,
    "length": 178,
    "height": 88
  }
}
```


Response status: 200


```
{
  "id": 6,
  "success": true,
  "message": "Successfully created Design 6"
}
```


Рисунок 3.5 – Створення дизайну меблів

Method URL

get /catalog/designs/view/canabre

Request params

Response status: 200


```
{
  "id": 6,
  "nameKey": "canabre",
  "type": "ottoman",
  "displayName": "Canabre",
  "description": "Compact ottoman couch. Elegant curves and ... Lorem ipsum sit amet blah blah we need
  "isVisible": true,
  "isTransformable": false,
  "dimCompact": {
    "length": 100,
    "width": 166,
    "height": 88
  },
  "dimExpanded": {
    "length": 178,
    "width": 166,
    "height": 88
  },
  "fabricQuantity": 0,
  "decorMaterialQuantity": 0,
  "price": 0,
  "createdAt": "2023-04-10T08:32:23.571742Z",
  "daysSinceCreated": 0,
  "priceRecalculatedAt": null,
  "photoUrls": []
}
```


Рисунок 3.6 – Перегляд створеного дизайну меблів

3.4 Реалізація клієнтської частини системи

Єдиного стандарту реалізації клієнтської частини з використанням Vue.js не існує [7]. При реалізації використано загальновідомий принцип навігації на стороні клієнта з використанням бібліотеки vue-router, що передбачає створення компонентів Vue та співставлення їх зі шляхом в адресному рядку браузера [8].

Компоненти Vue можуть включати в себе шаблони розмітки для відображення даних у зрозумілому вигляді, логіку для збереження даних, виконання яких-небудь обчислень, сигналювання подій та всього що дозволяє браузерна версія Javascript [7]. З метою розподілу відповідальності між компонентами клієнтської частини та повторного використання коду, запроваджена модель «Layout-Router-Page-View»:

– незмінна розкладка (Layout), що містить заголовок (header), підвал сайту (footer), та відносно пустий основний блок (main);

– роутер (Router), що визначає яку сторінку потрібно показати в блоці main залежно від адресного рядка браузера [8];

– сторінка (Page), що здійснює взаємодію с сервером WebAPI, є власником даних та обробляє події;

– вигляд (View), що слугує виключно для візуалізації даних та сигналювання подій (натискання кнопок, введення тексту і т.д.), може використовуватися повторно в декількох сторінках.

Окрім того, для повторного використання коду розроблені мікрокомпоненти, як-от ImageGallery для показу галереї зображень з кнопками для навігаціїї назад-вперед. Код компонента ImageGallery наведено в додатку Д.1.

Структура клієнтської частини у спрощеному вигляді наведена на рисунку 3.7. Приклад коду компонента-сторінки (designs/create) наведено в додатку Д.2. Приклад коду компонента-вигляду для відображення даних (DesignView) наведено в додатку Д.3. Приклад коду компонента-вигляду для вводу даних (DesignEdit) наведено в додатку Д.4. Повну версію вихідного коду клієнтської частини наведено в репозиторії [23].

3.5 Випробування клієнтської частини системи

Випробування клієнтської частини системи здійснюється в ручному режимі з використанням чек-листів. В таблиці 3.2 наведено приклад чек-листа для сторінки редагування дизайну меблів. Основна увага приділена функціональному тестуванню та тестуванню верстки.

Таблиця 3.2 – Чек-лист для сторінки редагування дизайну меблів

№ в/п	Опис пункту	Статус
1	**Верстка**	
1.1	Присутні заголовок (header) та підвал (footer) сторінки	+
1.2	Під заголовком присутні breadcrumbs із ієрархією сторінок	+
1.3	Присутній основний блок з формою редагування дизайну	+
1.4	Всі текстові поля відображаються: «Іменний код», «Найменування», «Опис».	+
1.5	Відображається група радіокнопок «Тип меблів»	+
1.6	Відображаються чекбокси «Відображається в каталозі», «Трансформується»	+
1.7	Всі поля, кнопки та групи підписані	+
1.8	В нижньому рядку відображаються кнопки «Скасувати зміни» та «Готово»	+
1.9	Елементи сторінки підлаштовуються під малий екран мобільного пристрою (360px)	+
2	**Функціонал**	
2.1	В усі поля можна вводити текст	+
2.2	Можна обрати одночасно лише 1 радіокнопку з типом дизайну	+
2.3	Чекбокси реагують на натискання	+
2.4	Група «Розміри в складеному стані» завжди відображається	+
2.5	Група «Розміри в розгорнутому стані» відображається тільки тоді, коли відмічено «Трансформується»	+
2.6	Можна додавати фотографії дизайну меблів	+
2.7	Існуючі фотографії не зникають при додаванні нових	+
2.8	Кнопка «Скасувати зміни» скидає значення всіх полів, в тому числі й нові фото	+

Продовження таблиці 3.2

№ в/п	Опис пункту	Статус
2.9	При натисканні кнопки «Готово» некоректно заповнені поля підсвічуються червоним кольором, внизу сторінки з’являється повідомлення з назвами некоректно заповнених полів	+
2.10	При натисканні кнопки «Готово», якщо все правильно, відображається повідомлення про успішне створення дизайну та посилання на сторінку перегляду дизайну меблів	+

Сторінку редагування дизайну меблів було випробувано з використанням наведеного вище чек-листа. Всі пункти чек-листа виконані. Зовнішній вигляд сторінки редагування дизайну меблів наведено на рисунку Е.26.

Таким же чином проводиться випробування інших сторінок клієнтської частини.

Висновки до розділу 3

На даному етапі реалізовано сутності та фасад бази даних системи з використанням мови програмування C Sharp. Далі було реалізовано сервер WebAPI з використанням мови C Sharp та фреймворку ASP.NET Core. Сервер випробуваний з використанням сценаріїв за допомогою інструмента Apinb. Клієнтська частина реалізована з використанням мов HTML, CSS, Javascript та фреймворку Vue.js. Випробування клієнтської частини здійснено в ручному режимі з використанням чек-листів.

4 РЕКОМЕНДАЦІЇ ПО ВИКОРИСТАННЮ СИСТЕМИ

4.1 Локальне розгортання системи

4.1.1 Попередні умови

Для побудови системи та локального розгортання необхідні:

– PostgreSQL версії 14 або вище;
– .NET Sdk версії 6;
– node версії 18 або вище;
– npm версії 8 або вище.

Перед запуском серверної частини застосунку треба створити пусту базу даних в середовищі PostgreSQL.

4.1.2 Заповнення бази даних

Щоб заповнити базу даних тестовим набором записів, необхідно:

– перейти до директорії Fwsh.MockData;
– створити файл .env та заповнити його за зразком .env.example, вказавши IP-адресу, порт, логін та пароль користувача бази даних;
– запустити застосунок командою dotnet run -- seed log-credentials > credentials.log [5].

4.1.3 Запуск сервера WebAPI

Для запуску сервера необхідно:

– перейти до директорії Fwsh.WebApi;
– створити файл .env та заповнити його за зразком .env.example, вказавши IP-адресу, порт, логін та пароль користувача бази даних;
– запустити застосунок командою dotnet run [5].

4.1.4 Запуск клієнтської частини

Для запуску клієнтської частини для менеджера, робітника чи покупця необхідно:

– нехай <usertype> = Manager для менеджера, Worker для робітника, Customer для покупця;

– перейти до директорії Frontend;

– в директорії <usertype> створити файл .env та заповнити його за зразком .env.example, вказавши URL сервера WebAPI та URL контент-сервера (для середовища Development ці значення URL будуть однакові);

– встановити необхідні пакети командою npm i -w <usertype> [25];

– запустити dev-сервер командою npm run dev -w <usertype> [25].

4.2 Розгортання на Production-сервері

4.2.1 Попередні умови

На Production-сервері необхідно встановити:

– PostgreSQL версії 14 або вище;

– Apache2 версії 2.4 або вище;

– apache2-proxy, apache2-ssl;

– rsync.

До комп’ютера, з якого здійснюється побудова компонентів системи, висуваються такі ж вимоги, як в п.4.1.1.

4.2.2 Побудова та розгортання

Сюди включаються такі етапи:

– конфігурація і побудова сервера WebAPI в форматі «self-contained»;

– налаштування і побудова модулів клієнтської частини;

– створення віртуальних хостів Apache2;

– створення SSL-сертифікатів;
– створення сервісу для запуску WebAPI у фоновому режимі;
– копіювання всіх артефактів на Production-сервер;
– створення бази даних у середовищі PostgreSQL;
– включення віртуальних хостів Apache2;
– запуск сервісу Fwsh.WebApi.

Інтерактивний скрипт для розгортання на сервері з ОС Alpine Linux наведено в репозиторії [20].

Оскільки SSL-сертифікати були створені та підписані власноруч, необхідно зареєструвати сертифікат rootCA.crt у браузері [15]. Якщо немає сервера DNS, то доведеться також внести імена хостів та відповідні IP-адреси в файл hosts [16].

Після цього можна переглядати:

– вебсайт покупця за адресою https://www.fwsh.example.com/;
– панель менеджера за адресою https://manager.fwsh.example.com/;
– панель робітника за адресою https://worker.fwsh.example.com/.

4.3 Робота з обліковими записами

Реєстрація всіх типів користувачів у системі відбувається за схожим алгоритмом. Необхідно:

– перейти на сторінку реєстрації;
– заповнити ПІБ, контактні дані та пароль (прізвище, ім’я, по батькові має містити не менше 2 літер, телефон має бути в форматі +380000000000, електронна пошта має відповідати загальноприйнятому формату, пароль має містити від 8 до 64 символів) (див. рис. Е.1);
– якщо покупець є представником організації, можна поставити відмітку «Я представляю організацію» та вказати її назву у відповідному полі (див. рис. Е.2);
– при реєстрації робітника необхідно вказати тип робіт, які він може виконувати (див. рис. Е.3);
– натиснути кнопку «Зареєструватися».

Якщо дані не відповідають формату або такий номер телефону вже існує, буде виведено повідомлення про помилку (див. рис. Е.4, Е.5). Якщо введені дані є коректними, буде здійснено перенаправлення на сторінку входу (див. рис. Е.6).

Після входу в систему буде здійнено перенаправлення на сторінку «Мій профіль» (див. рис. Е.7).

Для виходу з облікового запису будь-якого типу користувача необхідно натиснути кнопку «Вийти» на сторінці «Мій профіль» (див. рис. Е.7).

4.4 Використання панелі менеджера

Головна сторінка панелі менеджера зображена на рисунку Е.8. Для повноцінного доступу до панелі менеджера необхідно зареєструватися (див. рис. Е.1) або увійти в обліковий запис (див. рис. Е.6).

Для перегляду облікових записів користувачів необхідно на головній сторінці обрати пункт «Люди» та у відкритій сторінці обрати «Покупці», «Робітники» або «Постачальники». Буде здійснено перенаправлення на сторінку зі списком облікових записів (див. рис. Е.9).

Для перегляду ресурсів необхідно на головній сторінці обрати «Ресурси». Буде здійснено перенаправлення на таку сторінку (див. рис. Е.10). Далі можна перейти до списку деталей (див. рис. Е.11), матеріалів або тканин.

Для створення нової деталі необхідно натиснути на кнопку «+ Деталь» в правому верхньому куті сторінки (див. рис. Е.11). Буде здійснено перенаправлення на сторінку редагування ресурсу (див. рис. Е.12).

Для збереження необхідно натиснути кнопку «Зберегти» в правому нижньому куті сторінки. Після збереження деталь можна буде знайти у списку деталей (див. рис. Е.13). Таким же чином відбувається перегляд та редагування тканин і матеріалів.

Для перегляду списку кольорів необхідно на сторінці «Ресурси» (див. рис. Е.10) обрати пункт «Кольори». Буде здійснено перенаправлення на сторінку зі списком кольорів (див. рис. Е.14).

Для додавання нового кольору необхідно натиснути на кнопку «+ Колір» у правому верхньому куті сторінки. Для редагування вже існуючого кольору потрібно натиснути на його картку. Буде здійснено перенаправлення на сторінку редагування кольору (див. рис. Е.15).

За таким же алгоритмом відбувається перегляд списку типів тканин (див. рис. Е.16) та редагування типів тканин (див. рис. Е.17).

Для закупки деталей, матеріалів або тканин необхідно перейти до детального опису ресурсу та натиснути кнопку «Закупки ресурсів» у правому верхньому куті сторінки (див. рис. Е.18). Буде здійснено перенаправлення на сторінку зі списком закупок (див. рис. Е.19).

На даному етапі система не виконує прямої взаємодії з постачальниками ресурсів, але дозволяє вести облік замовлень на поставку ресурсів. Щоб записати нове замовлення, потрібно натиснути на кнопку «+ Створити» у правому верхньому куті сторінки (див. рис. Е.19). Буде здійснено перенаправлення на сторінку редагування замовлення (див. рис. Е.20).

Після збереження, замовлення буде в невизначеному стані. Його можна відмітити як відправлене, натиснувши на кнопку «Підтвердити» в правому нижньому куті сторінки (див. рис. Е.21).

Після того, як замовлення буде отримане, можна вказати реальну ціну та кількість поставленого ресурсу, та закрити замовлення (див. рис. Е.22).

Після того, як замовлення буде закрито, кількість та ціну за одиницю ресурсу буде автоматично перераховано (див. рис. Е.23). Закрите замовлення можна буде переглянути в архіві замовлень (див. рис. Е.24).

Для перегляду каталогу дизайнів необхідно на головній сторінці обрати пункт «Прототипи» та у відкритій сторінці обрати «Дизайни меблів». Буде здійснено перенаправлення на сторінку зі списком дизайнів (див. рис. Е.25).

Для створення нового дизайну необхідно натиснути кнопку «+Дизайн меблів» в правому верхньому куті сторінки. Буде здійснено перенаправлення на сторінку редагування (див. рис. Е.26). Там необхідно ввести характеристики дизайну меблів, як-от назву, опис, розміри в складеному та розгорнутому стані, прикріпити фото.

Після збереження новий дизайн можна буде переглянути в каталозі (див. рис. Е.27).

Для перегляду шаблонів задач, що стосуються одного дизайну меблів, потрібно на сторінці детального опису дизайну натиснути кнопку «Задачі». Буде здійснено перенаправлення на сторінку такого вигляду (див. рис. Е.28).

Для перегляду та редагування існуючого шаблону задачі потрібно натиснути на його картку. Для додавання нового шаблону задачі потрібно натиснути на кнопку «+ Створити» у верхньому правому куті сторінки. Буде здійснено перенаправлення на сторінку редагування задачі (див. рис. Е.29).

Після редагування списку задач варто оновити базову ціну пов’язаного дизайну меблів (див. рис. Е.30).

Для перегляду виробничих замовлень потрібно перейти на сторінку «Замовлення» та обрати список виробничих замовлень. В результаті відкриється сторінка такого вигляду (див. рис. Е.31). Аналогічно відбувається перегляд списку ремонтних замовлень.

Для детального перегляду замовлення необхідно натиснути на його картку. Буде здійснено неренаправлення на сторінку з детальним описом замовлення (див. рис. Е.32).

Для того, щоб сповістити покупця, потрібно натиснути на кнопку в вигляді дзвоника у верхньому правому куті сторінки (див. рис. Е.32), написати текст сповіщення та натиснути кнопку «Надіслати» (див. рис. Е.33). Аналогічно можна переглядати та додавати сповіщення до ремонтного замовлення.

Для того, щоб створити задачі за шаблоном, потрібно натиснути на кнопку «Створити» в правому нижньому куті сторінки (див. рис. Е.32). Буде здійснено перенаправлення на сторінку зі списком задач, що відносяться до даного замовлення (див. рис. Е.34).

Переглянути окрему задачу можна, натиснувши на її номер. Буде зійснено перенаправлення на сторінку з детальним описом задачі (див. рис. Е.35). Під детальним описом задачі подається використання ресурсів (див. рис. Е.36).

Щоб призначити робітника, необхідно натиснути на пункт «Призначити робітника» на сторінці опису задачі (див. рис. Е.35). Далі необхідно обрати робітника з запропонованого переліку та натиснути кнопку «Підтвердити» (див. рис. Е.37).

Для перегляду ремонтних замовлень необхідно на сторінці «Замовлення» обрати список ремонтних замовлень. Буде здійснено перенаправлення на сторінку зі списком замовлень (див. рис. Е.38).

Для перегляду детального опису ремонтного замовлення необхідно натиснути на його картку. Буде здійснено перенаправлення на сторінку з детальним описом замовлення (див. рис. Е.39).

Перегляд та додавання сповіщень відбувається так само, як у виробничому замовленні (див. рис. Е.33).

Щоб переглянути список задач, пов’язаних із замовленням, потрібно натиснути «Докладніше» у правому нижьому куті сторінки (див. рис. Е.39). Буде здійснено перенаправлення на сторінку зі списком задач (див. рис. Е.40).

Для перегляду або редагування існуючої задачі необхідно натиснути на її картку (див. рис. Е.40). Для створення нової задачі необхідно натиснути кнопу «+ Задача» у правому верхньому куті сторінки (див. рис. Е.40). Буде здійснено перенаправлення на сторінку такого вигляду (див. рис. Е.41).

Після задання ролі, оплати та текстового опису, можна додати ресурси (див. рис. Е.42). Для збереження потрібно натиснути кнопку «Зберегти» у правому нижньому куті сторінки, для скасування змін потрібно натиснути на кнопку «Скасувати зміни» у лівому нижньому куті сторінки (див. рис. Е.42).

Позначення можливості виконання задачі є виключно рекомендаційним. Якщо кількість деякого ресурсу на складі недостатня для виконання робіт, ресурс буде підсвічено червоним кольором (див. рис. Е.43).

Після редагування задач, необхідно повторно обчислити вартість замовлення. Для цього потрібно перейти на сторінку ремонтного замовлення та натиснути кнопку «Оновити» навпроти пункту «Вартість робіт» (див. рис. Е.44).

4.5 Використання панелі робітника

Для використання панелі робітника потрібно спочатку ввійти в обліковий запис або зареєструватися (див. рис. Е.1, Е.6).

Для того, щоб переглянути ресурси, потрібно на головній сторінці обрати пункт «Ресурси» та у відкритій сторінці обрати тип ресурсу (деталі, матеріали або тканини). Буде здійснено перенаправлення на сторінку зі списком ресурсів обраного типу (див. рис. Е.45).

Для того, щоб записати актуальну кількість ресурсу на складі, потрібно натиснути на кнопку «Оновити» на картці ресурсу (див. рис. Е.45). В результаті з’явиться форма для введення кількості ресурсу (див. рис. Е.46). Для підтвердження потрібно натиснути кнопку «Оновити кількість».

Для того, щоб переглянути виробничі задачі, потрібно на головній сторінці обрати «Задачі» та у відкритому меню задач обрати список виробничих задач. Буде здійснено перенаправлення на сторінку такого вигляду (див. рис. Е.47).

Для перегляду окремої задачі потрібно натиснути на її картку. Буде здійснено перенаправлення на сторінку з детальним описом задачі (див. рис. Е.48).

Щоб змінити статус задачі, треба натиснути на рядок статусу та обрати потрібний статус (наприклад, «Робота почалася») та підтвердити (див. рис. Е.49).

Після виконання робіт потрібно записати кількість використаних ресурсів (див. рис. Е.50), та змінити статус задачі на «Завершено». Аналогічно відбувається взаємодія з ремонтними задачами.

При появі завершених замовлень та задач (див. рис. Е.51), робітник може подати запит на отримання заробітної плати (заробітний чек).

Для створення нового заробітного чека потрібно натиснути на кнопку «+Створити» в верхньому правому куті сторінки (див. рис. Е.52). В результаті новий чек буде доданий до списку (див. рис. Е.53).

4.6 Використання вебсайту покупця

Для того, щоб переглянути повний каталог меблів, потрібно на головній сторінці прокрутити вниз по списку меблів та натиснути на посилання «Каталог», або в навігаційному меню обрати «Каталог» та у відкритій сторінці обрати «Каталог меблів». В результаті має відкритися така сторінка (див. рис. Е.54).

Для того, щоб відфільтрувати меблі по типу, треба в полі «Тип» обрати необхідне значення (див. рис. Е.55). Буде здійснено автоматичне перенаправлення на сторінку каталогу з меблями заданого типу (див. рис. Е.56).

Для перегляду детальної інформації про дизайн меблів необхідно натиснути на його картку. Буде відкрита сторінка з детальним описом та кнопкою «Зробити замовлення» (див. рис. Е.57). При натисканні на кнопку, якщо покупець ще не увійшов до свого облікового запису, буде здійснено перенаправлення на сторінку «Вхід» (див. рис. Е.6). Якщо покупець авторизований, буде здійснено перенаправлення на сторінку замовлення.

На сторінці замовлення треба:

– обрати кількість одиниць меблів (див. рис. Е.58);
– обрати колір або тип тканини (див. рис. Е.59);
– обрати тканину (див. рис. Е.60);
– обрати декораційний матеріал, якщо є (див. рис. Е.61).

По завершенню вибору всіх складових замовлення, покупця буде перенаправлено на сторінку інформації про замовлення (див. рис. Е.62). Його можна підтвердити або видалити натиснувши відповідну кнопку внизу сторінки.

Щоб переглянути сповіщення (див. рис. Е.63), треба натиснути на кнопку у вигляді дзвоника в верхньому правому куті сторінки детального опису замовлення.

Щоб подати замовлення на ремонт меблів, треба перейти на сторінку ремонтних замовлень з профілю або через навігаційне меню та натиснути на кнопку «+Замовлення» (див. рис. Е.64). Буде перенаправлено на сторінку для вводу інформації про замовлення. Тут необхідно ввести текстовий опис того, що треба відремонтувати, та додати фотографії (див. рис. Е.65).

Щоб подати замовлення, необхідно натиснути кнопку «Готово». Буде здійснено перенаправлення на сторінку детального опису замовлення (див. рис. Е.66). Можна оглянути список опублікованих замовлень (див. рис. Е.67). Так само як для виробничих замовлень, ремонтне замовлення можна підтвердити, або видалити якщо робота ще не почалася. Так само, як на рис. Е.63, можна переглядати сповіщення.

Висновки до розділу 4

В даному розділі наведені інструкції по розгортанню системи та керівництво для трьох категорій користувачів системи:

– для менеджера;
– для робітника;
– для покупця.

З використанням наочних прикладів доведено функціонування системи та її самодостатність: можна «з нуля» створити обліковий запис, додати ресурси, додати дизайни меблів до каталогу, подати замовлення на виробництво меблів, подати замовлення на ремонт меблів, створити задачі, призначити ресурси та виконавців, відмітити стан задачі, використання ресурсів, сповістити покупця про стан замовлення, обчислити заробітну плату для робітника.

ВИСНОВКИ

В ході роботи на основі інтерв’ю працівників меблевої майстерні та особистих спостережень були описані виробничі та облікові процеси, що відбуваються у майстерні.

На основі моделей процесів був запропонований варіант автоматизації у вигляді вебзастосунку. При розробці вебзастосунку були використані такі технології:

– PostgreSQL для бази даних;

– мова C Sharp та фреймворк ASP.NET Core для сервера WebAPI;

– мова Javascript та фреймворк Vue.js для клієнської частини.

В результаті отримано:

– сервер WebAPI, що забезпечує функціонування системи;

– вебсайт для покупців, що дозволяє переглядати каталог меблів, подавати замовлення на виробництво та ремонт меблів, відстежувати стан замовлень, переглядати сповіщення;

– панель робітника, що дозволяє переглядати власні задачі, звітувати про виконання задач, відмічати кількість ресурсів на складі, подавати запит на отримання заробітної плати;

– панель менеджера, що дозволяє управляти каталогом меблів та шаблонами задач, вести облік ресурсів, відстежувати замовлення на поставку ресурсів, приймати замовлення та видавати задачі робітникам, сповіщати покупців про стан замовлень, вести облік заробітної плати.

Результати роботи можуть бути використані при подальшій автоматизації виробничого обліку, прийому замовлень та управління задачами на інших підприємствах.

ПЕРЕЛІК ПОСИЛАНЬ

1. PostgreSQL 14 Documentation. URL: https://www.postgresql.org/docs/14/index.html (дата звернення: 20.02.2023).

2. Richardson L., Amundsen M., Ruby S. RESTful Web APIs: Services for a Changing World. USA : O'Reilly Media, 2013. 406 p.

3. Overview of ASP.NET Core. Microsoft Learn. URL: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core (дата звернення: 21.02.2023).

4. Overview of Entity Framework Core. Microsoft Learn. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 21.02.2023).

5. .NET CLI Overview. Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/core/tools/ (дата звернення: 21.02.2023).

6. Developer Guides. Mozilla developer network. URL: https://developer.mozilla.org/en-US/docs/Web/Guide (дата звернення: 22.02.2023).

7. Introduction. Vue.js. URL: https://vuejs.org/guide/introduction.html (дата звернення: 25.02.2023).

8. Programmatic navigation. Vue Router. URL: https://router.vuejs.org/guide/essentials/navigation.html (дата звернення: 26.02.2023).

9. Minimal Example. Axios Docs. URL: https://axios-http.com/docs/example (дата звернення: 26.02.2023).

10. ljharb/qs: A querystring parser with nesting support. GitHub. URL: https://github.com/ljharb/qs (дата звернення: 27.02.2023).

11. .NET (and .NET Core) – introduction and overview. Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/core/introduction (дата звернення: 10.03.2023).

12. Npgsql – .NET Access to PostgreSQL. Npgsql Documentation. URL: https://www.npgsql.org/index.html (дата звернення: 10.03.2023).

13. Getting Started. Vite. URL: https://vitejs.dev/guide/ (дата звернення: 14.03.2023).

14. Apache Virtual Host Documentation. URL: https://httpd.apache.org/docs/current/vhosts/ (дата звернення: 15.03.2023).

15. How To Create Self-Signed Certificates Using OpenSSL. DevOpsCube. URL: https://devopscube.com/create-self-signed-certificates-openssl/ (дата звернення: 14.03.2023).

16. Hosts(5) – Linux Manual Page. URL: https://man7.org/linux/man-pages/man5/hosts.5.html (дата звернення: 14.03.2023).

17. Creating and configuring a model – EF Core. Microsoft Learn. URL: https://learn.microsoft.com/en-us/ef/core/modeling/ (дата звернення: 25.03.2023).

18. DbContext Lifetime, Configuration, and Initialization – EF Core. Microsoft Learn. URL: https://learn.microsoft.com/en-us/ef/core/dbcontext-configuration/ (дата звернення: 26.03.2023).

19. Create and Drop APIs – EF Core. Microsoft Learn. URL: https://learn.microsoft.com/en-us/ef/core/managing-schemas/ensure-created (дата звернення: 27.02.2023).

20. Create Web APIs with ASP.NET Core. Microsoft Learn. URL: https://learn.microsoft.com/en-us/aspnet/core/web-api/ (дата звернення: 28.03.2023).

21. Dependency injection in ASP.NET Core. Microsoft Learn. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection (дата звернення: 28.03.2023).

22. App Startup in ASP.NET Core. Microsoft Learn. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/startup?view=aspnetcore-3.1 (дата звернення: 28.03.2023).

23. dmitriynaumov1024/fwsh: Production management and order processing system for a furniture workshop. GitHub. URL: https://github.com/dmitriynaumov1024/fwsh (дата звернення: 07.05.2023).

24. dmitriynaumov1024/apinb: interactive-notebook-like in-browser web api client. GitHub. URL: https://github.com/dmitriynaumov1024/apinb (дата звернення: 06.04.2023).

25. CLI Commands. Npm Docs. URL: https://docs.npmjs.com/cli/v8/commands (дата звернення: 06.04.2023).

ДОДАТОК А

Інтерв’ю зацікавлених осіб

А.1 Інтерв’ю: Андрій В., менеджер майстерні

– Чим займається це підприємство?

– Ми займаємося виробництвом меблів під замовлення.

– Чи надаєте ще які-небудь послуги?

– Також ми виконуємо ремонт меблів.

– Хто є клієнтами підприємства?

– Переважно це представники меблевих магазинів, що роблять оптові замовлення, але також є індивідуальні клієнти.

– Чи здійснюється доставка меблів?

– Ні, ми відвантажуємо меблі з цеху. У випадку, якщо клієнт бажає надіслати представника служби доставки, ми можемо віддати йому замовлення.

– Чи є Вас фіксований каталог дизайнів або Ви можете виконати будь-яке замовлення?

– Звісно, в нас є фіксований каталог дизайнів. Але якщо буде значний попит на виробництво нової моделі, то ми можемо додати її до каталогу.

– Які типи меблів виготовляєте?

– Ми займаємось виключно м'якими меблями. Можу назвати такі категорії: тахта, диван, міні-диван, кутовий диван, крісло, пуф.

– Опишіть, як зараз відбувається прийом замовлень.

– Покупець обирає дизайн меблів, колір та тип тканини, матеріал для декорацій, і найголовніше – кількість одиниць меблів. Для ідентифікації покупця ми записуємо його ПІБ, назву оргацізації якщо є, та контактні дані – номер телефону, електронну пошту.

– Опишіть, як відбувається прийом замовлень на ремонт меблів.

– Клієнт має надати фото меблів та текстовий опис робіт, які необхідно виконати. Ми разом з робітниками проводимо оцінку складності робіт та повідомляємо клієнта про можливість або неможливість виконати замовлення.

– Чи є система знижок для постійних покупців?

– Так. Знижка обчислюється на основі кількості отриманих та оплачених замовлень. Для індивідуальних покупців знижка 2% за замовлення, для організацій знижка 0.25% за замовлення. Максимальна знижка – 30%.

– Скільки замовлень на місяць та скільки одиниць меблів виготовляєте?

– В середньому ми маємо порядку 20 замовлень на місяць та виготовляємо порядку 350 одиниць меблів.

– Скільки робітників зараз працює у майстерні?

– Зараз маємо 9 робітників.

– Які типи робіт виконуються в ході створення меблів?

– Виготовлення дерев'яної рами, виготовлення корпусу, пошив чохла, оббивка, збирання.

– Який є розподіл робітників за ролями?

– У нас є 2 столяра, 2 швачки, 2 збиральника, 3 оббивщика.

– Чи виконують робітники лише ту роботу, що призначена для їх ролі, чи вони можуть виконувати інші типи завдань?

– У нас оббивщики є досить вмілими людьми та можуть виконувати задачі столярів та збиральників. А збиральники, у свою чергу, можуть виконувати роботу оббивщиків. Тобто, більшість робітників можуть виконувати декілька видів завдань.

– Розкажіть, як виконується виробниче замовлення.

– Після прийому замовлення обговорюємо з робітниками можливість його виконання. Якщо замовлення неможливо виконати з суб'єктивних причин або немає можливості добути всі необхідні деталі та матеріали – повідомляємо клієнта про неможливість виконання. Якщо в цеху залишилися меблі підходящого типу та кольору зі скасованих замовлень, то закріплюємо їх за цим замовленням. Далі, якщо потрібно робити нові меблі, задачі розподіляються між робітниками. Робітники виконують задачі по пріоритетам, тобто зазвичай спочатку виготовляється рама, потім корпус, потім чохол, потім частини корпусу обклеюються поролоном чи іншими матеріалами та закріплюється чохол, наприкінці предмет меблів збирається в єдине ціле. Коли всі задачі по замовленню виконані, ми повідомляємо

клієнта. Клієнт прибуває до цеху, оплачує замовлення та отримує меблі. Або він може відмовитися від замовлення, тоді вже готові меблі будуть по можливості переводитися на інші замовлення.

– **Розкажіть, як виконується ремонтне замовлення.**

– Після прийому замовлення визначаємо перелік задач, які треба виконати. Оцінюємо тривалість робіт, вартість та використання ресурсів. Робітники виконують задачі по пріоритетам. Коли всі задачі по замовленню виконані, або якщо замовлення неможливо виконати, ми повідомляємо клієнта. Клієнт прибуває до цеху, оплачує замовлення та забирає меблі.

– **Розкажіть, які є категорії ресурсів та як відбувається їх облік.**

– В нас на складі є такі категорії ресурсів: деталі, як-от гвинти, гайки, скоби, привідні механізми; матеріали, як-от деревина, ДСП, фанера, поролон, флізелін; тканина для обшивки. Облік відбувається на папері.

– **Як відбувається замовлення ресурсів?**

– Якщо я або робітники виявляють, що кількість деталей або матеріалу на складі значно нижче нормальної кількості (зазвичай це 30% від нормальної кількості або 60% якщо запаси ресурсу давно не поповнювалися), то необхідно замовити відповідний ресурс у постачальника. Може бути таке, що постачальник перестає існувати або ресурсу немає у нього в наявності, тоді треба шукати нового постачальника.

– **Як обчислюється ціна меблів?**

– Ми знаємо приблизне використання ресурсів для кожного найменування меблів. Але є індивідуальні для кожного замовлення тканини та матеріали для декораційних накладок. Тому ми можемо називати лише приблизну ціну для кожного дизайну меблів. Також окрім собівартості ресурсів до ціни додається заробітна плата та маржа (зазвичай 40%). При подачі замовлення від ціни віднімається знижка для постійних покупців.

– **Якою є мета розробки сайту для вашої майстерні?**

– Ми хочемо привабити більше клієнтів та автоматизувати облікові процеси, наскільки це можливо. Бо зараз на розподіл завдань та перевірки ресурсів на складі витрачається досить багато часу, до години кожного дня.

– **Хто буде користуватися майбутньою системою?**

– Менеджер, робітники та покупці.

– **Чи є ще які-небудь категорії осіб, з якими ви взаємодієте?**

– Так, це постачальники ресурсів.

А.2 Інтерв'ю: Павло М., робітник майстерні

– **Розкажіть, як відбувається облік виробництва меблів.**

– Це відбувається на папері. Робітники записують, наприклад, у зошиті, що необхідно виробити, які тканини та матеріал для декорацій використати. Після виконання робіт ми повідомляємо менеджера про завершення робіт, він проводить оцінку якості, якщо є якісь зауваження – меблі можуть бути відправлені на переробку. Якщо все добре, то повідомляємо клієнта про готовність меблів. Клієнт або його представник прибуває до нашої майстерні, оплачує замовлення та забирає меблі.

– **Розкажіть, як відбувається обчислення заробітної плати.**

– Відповідальність за це покладається на робітників. Ми самі підраховуємо кількість виконаних завдань та на основі прайс-листа робіт обчислюємо скільки треба просити у менеджера. Все тримається на чесному слові, але в менеджера є перелік замовлень, тому він не видасть більше ніж треба.

– **Розкажіть, як відбувається облік ресурсів на складі.**

– Робітники майстерні в ході виконання задач та походів на склад мають відмічати стан ресурсів. Якщо кількість якої-небудь деталі або матеріалу менше деякої порогової кількісті, або ресурс давно не поповнювався, то необхідно замовити цей ресурс у постачальника.

– **Чи потрібна саме Вам автоматизація обліку?**

– Я вважаю, так, тому що це дозволить зменшити кількість папірців та спростити обмін інформацією між нами та керівництвом. А чим швидше і надійніше передається інформація, тим менше імовірність того, що робота зупиниться через відсутність яких-небудь ресурсів.

А.3 Реверс-інтерв'ю: Дмитро Н., розробник системи

– Чи вважаєте Ви доречною автоматизацію нашого підприємства?

– Так, тому що це дозволить показати свою присутність в мережі Інтернет, привабити нових клієнтів, пришвидшити обмін інформацією між клієнтами та виконавцями робіт, спростити видачу завдань, забезпечити наглядність виконуваних робіт та збереження історії виробництва.

– Що Ви пропонуєте розробити для досягнення поставленої мети?

– Систему управління виробництвом та прийому замовлень, з клієнтським вебсайтом та панелями управління для менеджерів та для робітників. Система зв'яже воєдино процес перегляду каталогу, прийому замовлення, розподілу задач, виконання задач, отримання меблів.

– Які функції будуть доступні для клієнта?

– Вхід в обліковий запис, реєстрація, перегляд каталогу меблів, вибір кольору та тканини, вибір матеріалу декорацій, подача замовлення на виробництво меблів, подача замовлення на ремонт меблів, перегляд стану замовлення, перегляд сповіщень.

– Які функції будуть доступні для робітника?

– Вхід в обліковий запис, реєстрація, перегляд ресурсів на складі, оновлення кількості ресурсу, перегляд виробничих задач, перегляд ремонтних задач, зміна статусу задачі, обчислення заробітної плати.

– Які функції будуть доступні для менеджера?

– Всі можливості робітника, а також перегляд списку користувачів системи, додавання до каталогу нових дизайнів, кольорів, типів тканин, додавання нових ресурсів, створення виробничих задач, створення ремонтних задач, розподіл задач між робітниками, замовлення ресурсів у постачальника.

– Які технології будуть використані?

– Оскільки зараз немає ніякої автоматизації та все буде розроблено з нуля, пропоную використати PostgreSQL в якості бази даних, ASP.NET Core для серверного застосунку та Vue.js для інтерфейсів користувача.

ДОДАТОК Б

Модель WebAPI

Таблиця Б.1 – Опис зони /auth

№ в/п	Метод	Відносний шлях	Структура запиту	Структура відповіді

1.1	POST	/customer/signup	body: {
			"surname": String,
			"name": String,
			"patronym": String,
			"isOrganization":
			Boolean,
			"orgName": String,
			"phone": String,
			"email": String,
			"password": String
			}
				{
				"id": Number,
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

1.2	POST	/customer/login	body: {
			"phone": String,
			"password": String
			}
				{
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

1.3	POST	/customer/logout	-	{
				"message": String
				}

1.4	POST	/worker/signup	body: {
			"surname": String,
			"name": String,
			"patronym": String,
			"roles": String[],
			"phone": String,
			"email": String,
			"password": String
			}
				{
				"id": Number,
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

1.5	POST	/worker/login	body: {
			"phone": String,
			"password": String
			}
				{
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

1.6	POST	/worker/logout	-	{ "message": String }

1.7	POST	/manager/signup	body: {
			"surname": String,
			"name": String,
			"patronym": String,
			"phone": String,
			"email": String,
			"password": String
			}
				{
				"id": Number,
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

1.8	POST	/manager/login	body: {
			"phone": String,
			"password": String
			}
				{
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

1.9	POST	/manager/logout	-	{ "message": String }

Таблиця Б.2 – Опис зони /catalog

№ в/п	Метод	Відносний шлях	Структура запиту	Структура відповіді

2.1	GET	/colors/list	query: {
			"page": Number,
			"slot": String,
			"reverse": Boolean
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": Color[]
				}

2.2	GET	/fabrictypes/list	query: {
			"page": Number,
			"reverse": Boolean
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": FabricType[]
				}

2.3	GET	/fabrics/list	query: {
			"page": Number,
			"color": Number,
			"fabrictype": Number,
			"reverse": Boolean
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": Resource[]
				}

2.4	GET	/materials/list	query: {
			"page": Number,
			"reverse": Boolean
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": Resource[]
				}

2.5	GET	/designs/list	query: {
			"page": Number,
			"type": String,
			"reverse": Boolean
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": Design[]
				}

2.6	GET	/designs/view/:id	path: {
			"id": Number
			}
				Design

2.7	GET	/designs/view/:name	path: {
			"name": String
			}
				Design

Таблиця Б.3 – Опис зони /customer

№ в/п	Метод	Відносний шлях	Структура запиту	Структура відповіді
3.1	GET	/profile/view	-	Customer

3.2	POST	/profile/update	body: {
			"surname": String,
			"name": String,
			"patronym": String,
			"isOrganization":
			Boolean,
			"orgName": String,
			"oldPassword":String,
			"password": String
			}
				{
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

Продовження таблиці Б.3

№ в/п	Метод	Відносний шлях	Структура запиту	Структура відповіді

3.3	GET	/orders/production /list	query: {
			"page": Number
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": ProdOrder[]
				}

3.4	GET	/orders/production /archive	query: {
			"page": Number
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": ProdOrder[]
				}

3.5	GET	/orders/production /view/:id	path: {
			"id": Number
			}
				ProdOrder

3.6	POST	/orders/production /create	body: {
			"quantity": Number,
			"designId": Number,
			"fabricId": Number,
			"decorId": Number
			}
				{
				"id": Number,
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

3.7	POST	/orders/production /update/:id	path: {
			"id": Number
			}
			
			body: {
			"quantity": Number,
			"designId": Number,
			"fabricId": Number,
			"decorId": Number
			}
				{
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

3.8	POST	/orders/production /confirm-submit/:id	path: {
			"id": Number
			}
				{
				"success": Boolean,
				"message": String
				}

3.9	DELETE	/orders/production /delete/:id	path: {
			"id": Number
			}
				{
				"success": Boolean,
				"message": String
				}

3.10	POST	/orders/production /read-notifications	query: {
			"order": Number,
			"id": Number,
			"last": Number
			}
				{
				"success": Boolean,
				"message": String
				}

3.11	GET	/orders/repair /list	query: {
			"page": Number
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": RepairOrder[]
				}

3.12	GET	/orders/repair /archive	query: {
			"page": Number
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": RepairOrder[]
				}

Продовження таблиці Б.3

№ в/п	Метод	Відносний шлях	Структура запиту	Структура відповіді
3.13	GET	/orders/repair /view/:id	path: { "id": Number }	RepairOrder
3.14	POST	/orders/repair /create	body: { "description": String }	{ "id": Number, "success": Boolean, "message": String, "badFields": String[] }
3.15	POST	/orders/repair /update/:id	path: { "id": Number } body: { "description": String }	{ "success": Boolean, "message": String, "badFields": String[] }
3.16	POST	/orders/repair /attach-photos/:id	path: { "id": Number } body: FormData	{ "success": Boolean, "message": String }
3.17	POST	/orders/repair /confirm-submit/:id	path: { "id": Number }	{ "success": Boolean, "message": String }
3.18	DELETE	/orders/repair /delete/:id	path: { "id": Number }	{ "success": Boolean, "message": String }
3.19	POST	/orders/repair /read-notifications	query: { "order": Number, "id": Number, "last": Number }	{ "success": Boolean, "message": String }

Таблиця Б.4 – Опис зони /worker

№ в/п	Метод	Відносний шлях	Структура запиту	Структура відповіді
4.1	GET	/profile/view	-	Worker
4.2	POST	/profile/update	body: { "surname": String, "name": String, "patronym": String, "roles": String[], "oldPassword":String, "password": String }	{ "success": Boolean, "message": String, "badFields": String[] }

Продовження таблиці Б.4

№ в/п	Метод	Відносний шлях	Структура запиту	Структура відповіді

4.3	GET	/tasks/production /list	query: {
			"page": Number
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": ProdTask[]
				}

4.4	GET	/tasks/production /archive	query: {
			"page": Number
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": ProdTask[]
				}

4.5	GET	/tasks/production /view/:id	path {
			"id": Number
			}
				ProdTask

4.6	POST	/tasks/production /set-status/:id	path: {
			"id": Number
			}
			query: {
			"status": String
			}
				{
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

4.7	POST	/tasks/production /set-usage/:id	path: {
			"id": Number
			}
			
			body:
			ResourceQuantity[]
				{
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

4.8	GET	/tasks/repair /list	query: {
			"page": Number
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": RepairTask[]
				}

4.9	GET	/tasks/repair /archive	query: {
			"page": Number
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": RepairTask[]
				}

4.10	GET	/tasks/repair /view/:id	path: {
			"id": Number
			}
				RepairTask

4.11	POST	/tasks/repair /set-status/:id	path: {
			"id": Number
			}
			query {
			"status": String
			}
				{
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

4.12	POST	/tasks/repair /set-usage/:id	path {
			"id": Number
			}
			
			body:
			ResourceQuantity[]
				{
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

Продовження таблиці Б.4

№ в/п	Метод	Відносний шлях	Структура запиту	Структура відповіді

4.13	GET	/paychecks/list	query: {
			"page": Number
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": ProdTask[]
				}

4.14	GET	/paychecks/archive	query: {
			"page": Number
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": ProdTask[]
				}

4.15	POST	/paychecks/create	-	{
				"success": Boolean,
				"message": String,
				"tryLaterAt": String
				}

Таблиця Б.5 – Опис зони /manager

№ в/п	Метод	Відносний шлях	Структура запиту	Структура відповіді
5.1	GET	/profile/view	-	Worker

5.2	POST	/profile/update	body: {
			"surname": String,
			"name": String,
			"patronym": String,
			"roles": String[],
			"oldPassword":String,
			"password": String
			}
				{
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

5.3	GET	/customers/list	query: {
			"page": Number
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": Customer[]
				}

5.4	GET	/customers/view/:id	path: {
			"id": Number
			}
				Customer

5.5	GET	/workers/list	query: {
			"page": Number,
			"role": String
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": Worker[]
				}

5.6	GET	/workers/search	query: {
			"query": String
			}
				{
				"items": Worker[]
				}

5.7	GET	/workers/view/:id	path: {
			"id": Number
			}
				Worker

Продовження таблиці Б.5

№ в/п	Метод	Відносний шлях	Структура запиту	Структура відповіді

5.8	GET	/suppliers/list	query: {
			"page": Number,
			"role": String
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": Supplier[]
				}

5.9	GET	/suppliers/search	query: {
			"query": String
			}
				{
				"items": Supplier[]
				}

5.10	GET	/suppliers/view/:id	path: {
			"id": Number
			}
				Supplier

5.11	POST	/suppliers/create	body: {
			"surname": String,
			"name": String,
			"patronym": String,
			"orgName": String,
			"phone": String,
			"email": String
			}
				{
				"id": Number,
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

5.12	POST	/suppliers/update/:id	path: {
			"id": Number
			}
			body: {
			"surname": String,
			"name": String,
			"patronym": String,
			"orgName": String,
			"phone": String,
			"email": String
			}
				{
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

5.13	GET	/designs/list	query: {
			"page": Number,
			"type": String
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": Design[]
				}

5.14	GET	/designs/view/:id	path: {
			"id": Number
			}
				Design

5.15	POST	/designs/create	body: Design	{
				"id": Number,
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

5.16	POST	/designs/update/:id	path: {
			"id": Number
			}
			
			body: Design
				{
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

5.17	POST	/designs
		/attach-photos/:id
			path: {
			"id": Number
			}
			
			body:
			FormData
				{
				"success": Boolean,
				"message": String
				}

Продовження таблиці Б.5

№ в/п	Метод	Відносний шлях	Структура запиту	Структура відповіді
5.18	POST	/designs /set-visible/:id	path: { "id": Number } query: { "visible": Boolean }	{ "success": Boolean, "message": String }
5.19	POST	/designs/recalculate	query: { "id": Number }	{ "success": Boolean, "message": String }
5.20	DELETE	/designs/delete/:id	path: { "id": Number }	{ "success": Boolean, "message": String }
5.21	GET	/taskprototypes/list	query: { "design": Number }	{ "items": TaskPrototype[] }
5.22	GET	/taskprototypes /view/:id	path: { "id": Number }	TaskPrototype
5.23	POST	/taskprototypes /create	body: TaskPrototype	{ "id": Number, "success": Boolean, "message": String, "badFields": String[] }
5.24	POST	/taskprototypes /update/:id	path: { "id": Number } body: TaskPrototype	{ "success": Boolean, "message": String, "badFields": String[] }
5.25	DELETE	/taskprototypes /delete/:id	path: { "id": Number }	{ "success": Boolean, "message": String }
5.26	POST	/priceformation /update-default-prices	-	{ "success": Boolean, "message": String }
5.27	GET	/orders/production /list	query: { "page": Number, "design": Number }	{ "page": Number, "previous": Number, "next": Number, "items": ProdOrder[] }
5.28	GET	/orders/production /archive	query: { "page": Number, "design": Number }	{ "page": Number, "previous": Number, "next": Number, "items": ProdOrder[] }

Продовження таблиці Б.5

№ в/п	Метод	Відносний шлях	Структура запиту	Структура відповіді

5.29	GET	/orders/production /view/:id	path: {
			  "id": Number
			}
				ProdOrder

5.30	POST	/orders/production /set-active/:id	path: {
			  "id": Number
			}
			query: {
			  "active": Boolean
			}
				{
				  "success": Boolean,
				  "message": String
				}

5.31	POST	/orders/production /set-status/:id	path: {
			  "id": Number
			}
			query: {
			  "status": String
			}
				{
				  "success": Boolean,
				  "message": String
				}

5.32	POST	/orders/production /notify/:id	path: {
			  "id": Number
			}
			
			body: String
				{
				  "success": Boolean,
				  "message": String
				}

5.33	GET	/orders/repair /list	query: {
			  "page": Number,
			  "design": Number
			}
				{
				  "page": Number,
				  "previous": Number,
				  "next": Number,
				  "items": RepairOrder[]
				}

5.34	GET	/orders/repair /archive	query: {
			  "page": Number,
			  "design": Number
			}
				{
				  "page": Number,
				  "previous": Number,
				  "next": Number,
				  "items": RepairOrder[]
				}

5.35	GET	/orders/repair /view/:id	path: {
			  "id": Number
			}
				RepairOrder

5.36	POST	/orders/repair /set-active/:id	path: {
			  "id": Number
			}
			query: {
			  "active": Boolean
			}
				{
				  "success": Boolean,
				  "message": String
				}

5.37	POST	/orders/repair /set-status/:id	path: {
			  "id": Number
			}
			query: {
			  "status": String
			}
				{
				  "success": Boolean,
				  "message": String
				}

5.38	POST	/orders/repair /notify/:id	path: {
			  "id": Number
			}
			
			body: String
				{
				  "success": Boolean,
				  "message": String
				}

Продовження таблиці Б.5

№ в/п	Метод	Відносний шлях	Структура запиту	Структура відповіді

5.39	GET	/tasks/production /list	query: {
			"page": Number,
			"design": Number,
			"order": Number,
			"worker": Number
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": ProdTask[]
				}

5.40	GET	/task/production /archive	query: {
			"page": Number,
			"design": Number,
			"order": Number,
			"worker": Number
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": ProdTask[]
				}

5.41	GET	/tasks/production /view/:id	path: {
			"id": Number
			}
				ProdTask

5.42	GET	/tasks/production /preview	query: {
			"order": Number,
			"reuse": Boolean
			}
				{
				"items": ProdTask[],
				"success": Boolean,
				"message": String
				}

5.43	POST	/tasks/production /create	query: {
			"order": Number,
			"reuse": Boolean
			}
				{
				"id": Number[],
				"success": Boolean,
				"message": String
				}

5.44	POST	/tasks/production /assign/:id	path: {
			"id": Number
			}
			query: {
			"worker": Number
			}
				{
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

5.45	POST	/tasks/production /set-active/:id	path: {
			"id": Number
			}
			query: {
			"active": Boolean
			}
				{
				"success": Boolean,
				"message": String
				}

5.46	POST	/tasks/production /set-status/:id	path: {
			"id": Number
			}
			query: {
			"status": String
			}
				{
				"success": Boolean,
				"message": String
				}

5.47	GET	/tasks/repair /list	query: {
			"page": Number,
			"order": Number,
			"worker": Number
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": RepairTask[]
				}

5.48	GET	/task/repair /archive	query: {
			"page": Number,
			"order": Number,
			"worker": Number
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": RepairTask[]
				}

Продовження таблиці Б.5

№ в/п	Метод	Відносний шлях	Структура запиту	Структура відповіді

5.49	GET	/tasks/repair /view/:id	path: {
			"id": Number
			}
				ProdTask

5.50	POST	/tasks/repair /create	body:
			RepairTask
				{
				"id": Number,
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

5.51	POST	/tasks/repair /update/:id	path: {
			"id": Number
			}
			body:
			RepairTask
				{
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

5.52	POST	/tasks/repair /assign/:id	path: {
			"id": Number
			}
			query: {
			"worker": Number
			}
				{
				"success": Boolean,
				"message": String,
				"badFields": String[]
				}

5.53	POST	/tasks/repair /set-active/:id	path: {
			"id": Number
			}
			query: {
			"active": Boolean
			}
				{
				"success": Boolean,
				"message": String
				}

5.54	POST	/tasks/repair /set-status/:id	path: {
			"id": Number
			}
			query: {
			"status": String
			}
				{
				"success": Boolean,
				"message": String
				}

5.55	GET	/paychecks/list	query: {
			"page": Number
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items": Worker[]
				}

5.56	GET	/paychecks/archive	query: {
			"page": Number
			}
				{
				"page": Number,
				"previous": Number,
				"next": Number,
				"items":
				WorkerPaycheck[]
				}

5.57	POST	/paychecks/create	query: {
			"workerId": Number
			}
				{
				"success": Boolean,
				"message": String,
				"tryLaterAt": String
				}

5.58	POST	/paychecks /create-bonus	query: {
			"workerId": Number,
			"amount": Number
			}
				{
				"success": Boolean,
				"message": String,
				"tryLaterAt": String
				}

Продовження таблиці Б.5

№ в/п	Метод	Відносний шлях	Структура запиту	Структура відповіді
5.59	POST	/paychecks /confirm-payment	path: { "workerId": Number, "paycheckId": Number }	ProdTask
5.60	GET	/orders/supply /list	query: { "page": Number, "resource": Number }	{ "page": Number, "previous": Number, "next": Number, "items": SupplyOrder[] }
5.61	GET	/orders/supply /archive	query: { "page": Number, "resource": Number }	{ "page": Number, "previous": Number, "next": Number, "items": SupplyOrder[] }
5.62	GET	/orders/supply /view/:id	path: { "id": Number }	SupplyOrder
5.63	POST	/orders/supply /create	body: SupplyOrder	{ "id": Number, "success": Boolean, "message": String, "badFields": String[] }
5.64	POST	/orders/supply /update/:id	path: { "id": Number } body: SupplyOrder	{ "success": Boolean, "message": String, "badFields": String[] }
5.65	POST	/orders/supply /confirm-submit/:id	path: { "id": Number }	{ "success": Boolean, "message": String }
5.66	POST	/orders/supply /confirm-receive/:id	path: { "id": Number }	{ "success": Boolean, "message": String }

Таблиця Б.6 – Опис зони /resources

№ в/п	Метод	Відносний шлях	Структура запиту	Структура відповіді
6.1	GET	/colors/list	query: { "page": Number, "reverse": Boolean, "query": String }	{ "page": Number, "previous": Number, "next": Number, "items": Color[] }
6.2	GET	/colors/view/:id	path: { "id": Number }	Color

Продовження таблиці Б.6

№ в/п	Метод	Відносний шлях	Структура запиту	Структура відповіді

6.3	POST	/colors/create	body:
			  Color
				{
				  "id": Number,
				  "success": Boolean,
				  "message": String,
				  "badFields": String[]
				}

6.4	POST	/colors/update/:id	path: {
			  "id": Number
			}
			
			body:
			  Color
				{
				  "success": Boolean,
				  "message": String,
				  "badFields": String[]
				}

6.5	DELETE	/colors/delete/:id	path: {
			  "id": Number
			}
				{
				  "success": Boolean,
				  "message": String
				}

6.6	GET	/fabrictypes/list	query: {
			  "page": Number,
			  "reverse": Boolean,
			  "query": String
			}
				{
				  "page": Number,
				  "previous": Number,
				  "next": Number,
				  "items": FabricType[]
				}

6.7	GET	/fabrictypes/view/:id	path: {
			  "id": Number
			}
				FabricType

6.8	POST	/fabrictypes/create	body:
			  FabricType
				{
				  "id": Number,
				  "success": Boolean,
				  "message": String,
				  "badFields": String[]
				}

6.9	POST	/fabrictypes
		/update/:id
			path: {
			  "id": Number
			}
			
			body:
			  FabricType
				{
				  "success": Boolean,
				  "message": String,
				  "badFields": String[]
				}

6.10	DELETE	/fabrictypes
		/delete/:id
			path: {
			  "id": Number
			}
				{
				  "success": Boolean,
				  "message": String
				}

6.11	GET	/resources/any/list	query: {
			  "page": Number
			}
				{
				  "page": Number,
				  "previous": Number,
				  "next": Number,
				  "items": Resource[]
				}

6.12	GET	/resources/any
		/view/:id
			path: {
			  "id": Number
			}
				Resource

Продовження таблиці Б.6

№ в/п	Метод	Відносний шлях	Структура запиту	Структура відповіді

6.13	GET	/parts/list	query: {
			  "page": Number
			}
				{
				  "page": Number,
				  "previous": Number,
				  "next": Number,
				  "items": Resource[]
				}

6.14	GET	/parts/view/:id	path: {
			  "id": Number
			}
				Resource

6.15	POST	/parts/create	body:
			  Resource
				{
				  "id": Number,
				  "success": Boolean,
				  "message": String,
				  "badFields": String[]
				}

6.16	POST	/parts/update/:id	path: {
			  "id": Number
			}
			
			body: Resource
				{
				  "success": Boolean,
				  "message": String,
				  "badFields": String[]
				}

6.17	POST	/parts/set-quantity	path: {
			  "id": Number
			}
			query: {
			  "quantity": Number
			}
				{
				  "success": Boolean,
				  "message": String,
				  "badFields": String[]
				}

6.18	GET	/materials/list	query: {
			  "page": Number
			}
				{
				  "page": Number,
				  "previous": Number,
				  "next": Number,
				  "items": Resource[]
				}

6.19	GET	/materials/view/:id	path: {
			  "id": Number
			}
				Resource

6.20	POST	/materials/create	body:
			  Resource
				{
				  "id": Number,
				  "success": Boolean,
				  "message": String,
				  "badFields": String[]
				}

6.21	POST	/materials/update/:id	path: {
			  "id": Number
			}
			
			body: Resource
				{
				  "success": Boolean,
				  "message": String,
				  "badFields": String[]
				}

6.22	POST	/materials
		/attach-photo/:id
			path: {
			  "id": Number
			}
			
			body: FormaData
				{
				  "success": Boolean,
				  "message": String
				}

6.23	POST	/materials
		/set-quantity
			path: {
			  "id": Number
			}
			query: {
			  "quantity": Number
			}
				{
				  "success": Boolean,
				  "message": String,
				  "badFields": String[]
				}

Продовження таблиці Б.6

№ в/п	Метод	Відносний шлях	Структура запиту	Структура відповіді

6.24	GET	/fabrics/list	query: {
			  "page": Number,
			  "reverse": Boolean
			}
				{
				  "page": Number,
				  "previous": Number,
				  "next": Number,
				  "items": Resource[]
				}

6.25	GET	/fabrics/view/:id	path: {
			  "id": Number
			}
				Resource

6.26	POST	/fabrics/create	body:
			  Resource
				{
				  "id": Number,
				  "success": Boolean,
				  "message": String,
				  "badFields": String[]
				}

6.27	POST	/fabrics/update/:id	path: {
			  "id": Number
			}
			
			body: Resource
				{
				  "success": Boolean,
				  "message": String,
				  "badFields": String[]
				}

6.28	POST	/fabrics
		/attach-photo/:id
			path: {
			  "id": Number
			}
			
			body: FormaData
				{
				  "success": Boolean,
				  "message": String
				}

6.29	POST	/fabrics
		/set-quantity
			path: {
			  "id": Number
			}
			query: {
			  "quantity": Number
			}
				{
				  "success": Boolean,
				  "message": String,
				  "badFields": String[]
				}

ДОДАТОК В

Реалізація бази даних

В.1 Приклад класу сутності Design

```
namespace Fwsh.Common;

using System;
using System.Collections.Generic;
using System.Linq;

public class Design
{
    public int Id { get; set; }
    public string NameKey { get; set; }
    public string DisplayName { get; set; }
    public string Type { get; set; } = FurnitureTypes.Unknown;
    public bool IsVisible { get; set; }
    public bool IsTransformable { get; set; }
    public Dimensions DimCompact { get; set; }
    public Dimensions DimExpanded { get; set; }
    public double FabricUsage { get; set; }
    public double DecorUsage { get; set; }
    public string Description { get; set; }
    public int Price { get; set; }
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
    public DateTime RecalculatedAt { get; set; } = DateTime.UtcNow;

    public virtual ICollection<TaskPrototype> Tasks { get; set; }
    public virtual IReadOnlyList<string> PhotoUrls { get; set; }

    public Design()
    {
        this.Tasks = new HashSet<TaskPrototype>();
        this.PhotoUrls = new List<string>();
    }

    public void UpdatePrice()
    {
        this.Price = (this.CalculateResourcePrice()
                     + this.CalculatePayment())
                     .WithMargin();
        this.RecalculatedAt = DateTime.UtcNow;
    }

    public void UpdateResourceQuantities()
    {
        this.FabricUsage = this.Tasks.Sum ( task => task.Fabrics
            .Where(f => f.SlotName == SlotNames.Fabric)
            .Sum(f => f.ExpectQuantity)
        );

        this.DecorUsage = this.Tasks.Sum ( task => task.Materials
            .Where(m => m.SlotName == SlotNames.Decor)
            .Sum(m => m.ExpectQuantity)
        );
    }
```

```
    public int CalculateResourcePrice()
    {
        return this.Tasks.Sum(task => task.CalculateResourcePrice());
    }

    public int CalculatePayment()
    {
        return this.Tasks.Sum(task => task.Payment);
    }

    public ProdFurniture CreateFurniture (ProdOrder order)
    {
        return new ProdFurniture() {
            Order = order,
            Design = this,
            Fabric = order.Fabric,
            Decor = order.Decor,
            Status = OrderStatus.Delayed,
            Tasks = new HashSet<ProdTask> (
                this.Tasks.Select(t => t.CreateProdTask(order))
            )
        };
    }
}
```

В.2 Фасад бази даних

```
namespace Fwsh.Database;

using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.Extensions.Logging;

using Fwsh.Common;
using Fwsh.Logging;
using Fwsh.Utils;


public class FwshDataContext : DbContext
{
    private Action<DbContextOptionsBuilder> Configure;

    public DbSet<Customer> Customers { get; set; }
    public DbSet<Worker> Workers { get; set; }
    public DbSet<Supplier> Suppliers { get; set; }

    public DbSet<Design> Designs { get; set; }
    public DbSet<TaskPrototype> TaskPrototypes { get; set; }

    public DbSet<Color> Colors { get; set; }
    public DbSet<FabricType> FabricTypes { get; set; }

    public DbSet<Resource> Resources { get; set; }
    public DbSet<SupplyOrder> SupplyOrders { get; set; }

    public DbSet<ProdOrder> ProdOrders { get; set; }
    public DbSet<ProdFurniture> ProdFurniture { get; set; }
```

```
public DbSet<ProdTask> ProdTasks { get; set; }

public DbSet<RepairOrder> RepairOrders { get; set; }
public DbSet<RepairTask> RepairTasks { get; set; }

public DbSet<ProdNotification> ProdNotifications { get; set; }
public DbSet<RepairNotification> RepairNotifications { get; set; }

public FwshDataContext(Action<DbContextOptionsBuilder> configure = null) : base()
{
    this.Configure = configure;
}

protected override void OnConfiguring (DbContextOptionsBuilder optionsBuilder)
{
    if (this.Configure != null) this.Configure(optionsBuilder);

    this.ConfigureLogging(optionsBuilder);
}

void ConfigureLogging (DbContextOptionsBuilder optionsBuilder)
{
    if (env.get("DB_LOGFILE") == null) return;

    var loggingCategories = new List<string>();
    if (env.isTrue("DB_LOG_QUERIES"))
        loggingCategories.Add(DbLoggerCategory.Query.Name);
    if (env.isTrue("DB_LOG_QUERIES"))
        loggingCategories.Add(DbLoggerCategory.Database.Command.Name);
    if (env.isTrue("DB_LOG_UPDATES"))
        loggingCategories.Add(DbLoggerCategory.Update.Name);

    if (loggingCategories.Count > 0) {
        Logger logger = env.get("DB_LOGFILE")?.ToLower() switch {
            null => new ConsoleLogger(),
            "console" => new ConsoleLogger(),
            string filename => FileLogger.To(filename)
        };

        if (env.isDevelopment)
            optionsBuilder.EnableSensitiveDataLogging();
        optionsBuilder.LogTo(message => logger.Log(message),
                             loggingCategories, LogLevel.Information);
    }
}

protected override void OnModelCreating (ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Resource>(resource => {
        // HasForeignKey<TDependentEntity>
        // Type param is necessary to know who is the dependent entity here
        resource.HasOne(r => r.Stored).WithOne()
                .HasForeignKey<StoredResource>(stored => stored.Id);
    });

    modelBuilder.Entity<Design>(design => {
        design.HasKey(d => d.Id);
        design.HasIndex(d => d.NameKey).IsUnique();
        design.Property(d => d.DimCompact).HasConversion<DimensionsConverter>();
        design.Property(d => d.DimExpanded).HasConversion<DimensionsConverter>();
        design.Property(d => d.PhotoUrls).HasConversion<StringListConverter>();
    });

    modelBuilder.Entity<TaskPrototype>(task => {
        task.Ignore(t => t.Parts);
```

```
            task.Ignore(t => t.Materials);
            task.Ignore(t => t.Fabrics);
        });

        modelBuilder.Entity<Customer>(customer => {
            customer.HasIndex(c => c.Phone).IsUnique();
        });

        modelBuilder.Entity<Worker>(worker => {
            worker.HasIndex(w => w.Phone).IsUnique();
            worker.Ignore(w => w.Roles);
        });

        modelBuilder.Entity<RepairOrder>(order => {
            order.Property(o => o.PhotoUrls).HasConversion<StringListConverter>();
        });
    }
}
```

B.3 Конкретна реалізація фасаду для PostgreSQL

```
namespace Fwsh.Database;
using System;
using Microsoft.EntityFrameworkCore;
using System.Linq;
using Fwsh.Utils;

public class FwshDataContextPostgres : FwshDataContext
{
    public FwshDataContextPostgres() : base(null) { }

    protected override void OnConfiguring (DbContextOptionsBuilder optionsBuilder)
    {
#warning FwshDataContextPostgres relies on DB_HOST, DB_PORT, DB_DATABASE, DB_USER,
DB_PASSWORD variables from either environment or .env file in working directory. An
Exception will be thrown if one or more variables are not found.

        base.OnConfiguring(optionsBuilder);
        string[] requiredEnv = new[] {
            "DB_HOST", "DB_PORT", "DB_DATABASE", "DB_USER", "DB_PASSWORD"
        };
        var missingEnv = requiredEnv.Where(key => env.get(key) == null).ToList();
        if (missingEnv.Count > 0) {
            throw new Exception (
                "One or more environment variables were not found: " +
                String.Join(", ", missingEnv)
            );
        }
        optionsBuilder.UseSnakeCaseNamingConvention();
        optionsBuilder.UseNpgsql ( String.Format (
            "Host={0};Port={1};Database={2};Username={3};Password={4}",
            env.get("DB_HOST"), env.get("DB_PORT"), env.get("DB_DATABASE"),
            env.get("DB_USER"), env.get("DB_PASSWORD")
        ));
    }

    protected override void OnModelCreating (ModelBuilder modelBuilder) {
        base.OnModelCreating(modelBuilder);
    }
}
```

B.4 SQL для створення бази даних

```
CREATE TABLE colors (
    id integer GENERATED BY DEFAULT AS IDENTITY,
    name text NULL,
    rgb_code text NULL,
    created_at timestamp with time zone NOT NULL,
    CONSTRAINT pk_colors PRIMARY KEY (id)
);

CREATE TABLE customers (
    id integer GENERATED BY DEFAULT AS IDENTITY,
    is_organization boolean NOT NULL,
    org_name text NULL,
    discount_percent integer NOT NULL,
    surname text NULL,
    name text NULL,
    patronym text NULL,
    phone text NULL,
    email text NULL,
    password text NULL,
    created_at timestamp with time zone NOT NULL,
    CONSTRAINT pk_customers PRIMARY KEY (id)
);

CREATE TABLE designs (
    id integer GENERATED BY DEFAULT AS IDENTITY,
    name_key text NULL,
    display_name text NULL,
    type text NULL,
    is_visible boolean NOT NULL,
    is_transformable boolean NOT NULL,
    dim_compact text NULL,
    dim_expanded text NULL,
    fabric_usage double precision NOT NULL,
    decor_usage double precision NOT NULL,
    description text NULL,
    price integer NOT NULL,
    created_at timestamp with time zone NOT NULL,
    recalculated_at timestamp with time zone NOT NULL,
    photo_urls text NULL,
    CONSTRAINT pk_designs PRIMARY KEY (id)
);

CREATE TABLE fabric_types (
    id integer GENERATED BY DEFAULT AS IDENTITY,
    name text NULL,
    description text NULL,
    created_at timestamp with time zone NOT NULL,
    CONSTRAINT pk_fabric_types PRIMARY KEY (id)
);
CREATE TABLE suppliers (
    id integer GENERATED BY DEFAULT AS IDENTITY,
    org_name text NULL,
    surname text NULL,
    name text NULL,
    patronym text NULL,
    phone text NULL,
    email text NULL,
    password text NULL,
    created_at timestamp with time zone NOT NULL,
    CONSTRAINT pk_suppliers PRIMARY KEY (id)
);
```

```
CREATE TABLE workers (
    id integer GENERATED BY DEFAULT AS IDENTITY,
    roles_string text NULL,
    surname text NULL,
    name text NULL,
    patronym text NULL,
    phone text NULL,
    email text NULL,
    password text NULL,
    created_at timestamp with time zone NOT NULL,
    CONSTRAINT pk_workers PRIMARY KEY (id)
);

CREATE TABLE repair_orders (
    id integer GENERATED BY DEFAULT AS IDENTITY,
    description text NULL,
    photo_urls text NULL,
    customer_id integer NOT NULL,
    price integer NOT NULL,
    payment integer NOT NULL,
    status text NULL,
    is_active boolean NOT NULL,
    created_at timestamp with time zone NOT NULL,
    started_at timestamp with time zone NULL,
    finished_at timestamp with time zone NULL,
    received_at timestamp with time zone NULL,
    CONSTRAINT pk_repair_orders PRIMARY KEY (id),
    CONSTRAINT fk_repair_orders_customers_customer_id FOREIGN KEY (customer_id)
    REFERENCES customers (id) ON DELETE CASCADE
);

CREATE TABLE task_prototypes (
    id integer GENERATED BY DEFAULT AS IDENTITY,
    design_id integer NOT NULL,
    precedence integer NOT NULL,
    role text NULL,
    payment integer NOT NULL,
    description text NULL,
    CONSTRAINT pk_task_prototypes PRIMARY KEY (id),
    CONSTRAINT fk_task_prototypes_designs_design_id FOREIGN KEY (design_id)
    REFERENCES designs (id) ON DELETE CASCADE
);

CREATE TABLE resources (
    id integer GENERATED BY DEFAULT AS IDENTITY,
    color_id integer NULL,
    fabric_type_id integer NULL,
    type text NULL,
    slot_name text NULL,
    name text NULL,
    description text NULL,
    photo_url text NULL,
    precision integer NOT NULL,
    measure_unit text NULL,
    price_per_unit double precision NOT NULL,
    created_at timestamp with time zone NOT NULL,
    CONSTRAINT pk_resources PRIMARY KEY (id),
    CONSTRAINT fk_resources_colors_color_id FOREIGN KEY (color_id)
    REFERENCES colors (id),
    CONSTRAINT fk_resources_fabric_types_fabric_type_id FOREIGN KEY (fabric_type_id)
    REFERENCES fabric_types (id)
);
```

```
CREATE TABLE worker_paycheck (
    id integer GENERATED BY DEFAULT AS IDENTITY,
    worker_id integer NOT NULL,
    interval_start timestamp with time zone NOT NULL,
    interval_end timestamp with time zone NOT NULL,
    amount integer NOT NULL,
    is_received boolean NOT NULL,
    CONSTRAINT pk_worker_paycheck PRIMARY KEY (id),
    CONSTRAINT fk_worker_paycheck_workers_worker_id FOREIGN KEY (worker_id) REFERENCES
workers (id) ON DELETE CASCADE
);

CREATE TABLE repair_notifications (
    id integer GENERATED BY DEFAULT AS IDENTITY,
    repair_order_id integer NOT NULL,
    text text NULL,
    is_read boolean NOT NULL,
    created_at timestamp with time zone NOT NULL,
    CONSTRAINT pk_repair_notifications PRIMARY KEY (id),
    CONSTRAINT fk_repair_notifications_repair_orders_repair_order_id FOREIGN KEY
(repair_order_id) REFERENCES repair_orders (id) ON DELETE CASCADE
);

CREATE TABLE repair_tasks (
    id integer GENERATED BY DEFAULT AS IDENTITY,
    order_id integer NOT NULL,
    description text NULL,
    role text NULL,
    worker_id integer NULL,
    payment integer NOT NULL,
    status text NULL,
    is_active boolean NOT NULL,
    created_at timestamp with time zone NOT NULL,
    started_at timestamp with time zone NULL,
    finished_at timestamp with time zone NULL,
    CONSTRAINT pk_repair_tasks PRIMARY KEY (id),
    CONSTRAINT fk_repair_tasks_repair_orders_order_id FOREIGN KEY (order_id)
REFERENCES repair_orders (id) ON DELETE CASCADE,
    CONSTRAINT fk_repair_tasks_workers_worker_id FOREIGN KEY (worker_id) REFERENCES
workers (id)
);

CREATE TABLE prod_orders (
    id integer GENERATED BY DEFAULT AS IDENTITY,
    design_id integer NOT NULL,
    fabric_id integer NOT NULL,
    decor_id integer NULL,
    price_per_one integer NOT NULL,
    quantity integer NOT NULL,
    prepayment integer NOT NULL,
    customer_id integer NOT NULL,
    price integer NOT NULL,
    payment integer NOT NULL,
    status text NULL,
    is_active boolean NOT NULL,
    created_at timestamp with time zone NOT NULL,
    started_at timestamp with time zone NULL,
    finished_at timestamp with time zone NULL,
    received_at timestamp with time zone NULL,
    CONSTRAINT pk_prod_orders PRIMARY KEY (id),
    CONSTRAINT fk_prod_orders_customers_customer_id FOREIGN KEY (customer_id)
REFERENCES customers (id) ON DELETE CASCADE,
    CONSTRAINT fk_prod_orders_designs_design_id FOREIGN KEY (design_id) REFERENCES
designs (id) ON DELETE CASCADE,
```

```
    CONSTRAINT fk_prod_orders_resources_decor_id FOREIGN KEY (decor_id) REFERENCES
resources (id),
    CONSTRAINT fk_prod_orders_resources_fabric_id FOREIGN KEY (fabric_id) REFERENCES
resources (id) ON DELETE CASCADE
);

CREATE TABLE stored_resource (
    id integer NOT NULL,
    supplier_id integer NULL,
    external_id text NULL,
    in_stock double precision NOT NULL,
    normal_stock double precision NOT NULL,
    supply_order_count integer NOT NULL,
    refill_period_days integer NOT NULL,
    last_refilled_at timestamp with time zone NOT NULL,
    last_checked_at timestamp with time zone NOT NULL,
    CONSTRAINT pk_stored_resource PRIMARY KEY (id),
    CONSTRAINT fk_stored_resource_resources_id FOREIGN KEY (id) REFERENCES resources
(id) ON DELETE CASCADE,
    CONSTRAINT fk_stored_resource_suppliers_supplier_id FOREIGN KEY (supplier_id)
REFERENCES suppliers (id)
);

CREATE TABLE supply_orders (
    id integer GENERATED BY DEFAULT AS IDENTITY,
    supplier_id integer NOT NULL,
    item_id integer NOT NULL,
    external_id text NULL,
    expect_quantity double precision NOT NULL,
    expect_price_per_unit double precision NOT NULL,
    actual_quantity double precision NOT NULL,
    actual_price_per_unit double precision NOT NULL,
    status text NULL,
    is_active boolean NOT NULL,
    created_at timestamp with time zone NOT NULL,
    received_at timestamp with time zone NULL,
    CONSTRAINT pk_supply_orders PRIMARY KEY (id),
    CONSTRAINT fk_supply_orders_resources_item_id FOREIGN KEY (item_id) REFERENCES
resources (id) ON DELETE CASCADE,
    CONSTRAINT fk_supply_orders_suppliers_supplier_id FOREIGN KEY (supplier_id)
REFERENCES suppliers (id) ON DELETE CASCADE
);

CREATE TABLE prod_furniture (
    id integer GENERATED BY DEFAULT AS IDENTITY,
    order_id integer NULL,
    design_id integer NOT NULL,
    fabric_id integer NULL,
    decor_id integer NULL,
    status text NULL,
    created_at timestamp with time zone NOT NULL,
    started_at timestamp with time zone NULL,
    finished_at timestamp with time zone NULL,
    CONSTRAINT pk_prod_furniture PRIMARY KEY (id),
    CONSTRAINT fk_prod_furniture_designs_design_id FOREIGN KEY (design_id) REFERENCES
designs (id) ON DELETE CASCADE,
    CONSTRAINT fk_prod_furniture_prod_orders_order_id FOREIGN KEY (order_id)
REFERENCES prod_orders (id),
    CONSTRAINT fk_prod_furniture_resources_decor_id FOREIGN KEY (decor_id) REFERENCES
resources (id),
    CONSTRAINT fk_prod_furniture_resources_fabric_id FOREIGN KEY (fabric_id)
REFERENCES resources (id)
);
```

```
CREATE TABLE prod_notifications (
    id integer GENERATED BY DEFAULT AS IDENTITY,
    prod_order_id integer NOT NULL,
    text text NULL,
    is_read boolean NOT NULL,
    created_at timestamp with time zone NOT NULL,
    CONSTRAINT pk_prod_notifications PRIMARY KEY (id),
    CONSTRAINT fk_prod_notifications_prod_orders_prod_order_id FOREIGN KEY
(prod_order_id) REFERENCES prod_orders (id) ON DELETE CASCADE
);

CREATE TABLE prod_tasks (
    id integer GENERATED BY DEFAULT AS IDENTITY,
    prototype_id integer NOT NULL,
    furniture_id integer NULL,
    worker_id integer NULL,
    payment integer NOT NULL,
    status text NULL,
    is_active boolean NOT NULL,
    created_at timestamp with time zone NOT NULL,
    started_at timestamp with time zone NULL,
    finished_at timestamp with time zone NULL,
    CONSTRAINT pk_prod_tasks PRIMARY KEY (id),
    CONSTRAINT fk_prod_tasks_prod_furniture_furniture_id FOREIGN KEY (furniture_id)
REFERENCES prod_furniture (id),
    CONSTRAINT fk_prod_tasks_task_prototypes_prototype_id FOREIGN KEY (prototype_id)
REFERENCES task_prototypes (id) ON DELETE CASCADE,
    CONSTRAINT fk_prod_tasks_workers_worker_id FOREIGN KEY (worker_id) REFERENCES
workers (id)
);

CREATE TABLE resource_quantity (
    id integer GENERATED BY DEFAULT AS IDENTITY,
    item_id integer NULL,
    slot_name text NULL,
    item_name text NULL,
    expect_quantity double precision NOT NULL,
    actual_quantity double precision NOT NULL,
    prod_task_id integer NULL,
    repair_task_id integer NULL,
    task_prototype_id1 integer NULL,
    CONSTRAINT pk_resource_quantity PRIMARY KEY (id),
    CONSTRAINT fk_resource_quantity_prod_tasks_prod_task_id FOREIGN KEY (prod_task_id)
REFERENCES prod_tasks (id),
    CONSTRAINT fk_resource_quantity_repair_tasks_repair_task_id FOREIGN KEY
(repair_task_id) REFERENCES repair_tasks (id),
    CONSTRAINT fk_resource_quantity_resources_item_id FOREIGN KEY (item_id) REFERENCES
resources (id),
    CONSTRAINT fk_resource_quantity_task_prototypes_task_prototype_id3 FOREIGN KEY
(task_prototype_id1) REFERENCES task_prototypes (id)
);

CREATE UNIQUE INDEX ix_customers_phone ON customers (phone);

CREATE UNIQUE INDEX ix_designs_name_key ON designs (name_key);

CREATE INDEX ix_prod_furniture_decor_id ON prod_furniture (decor_id);
CREATE INDEX ix_prod_furniture_design_id ON prod_furniture (design_id);
CREATE INDEX ix_prod_furniture_fabric_id ON prod_furniture (fabric_id);
CREATE INDEX ix_prod_furniture_order_id ON prod_furniture (order_id);

CREATE INDEX ix_prod_notifications_prod_order_id ON prod_notifications
(prod_order_id);

CREATE INDEX ix_prod_orders_customer_id ON prod_orders (customer_id);
```

```
CREATE INDEX ix_prod_orders_decor_id ON prod_orders (decor_id);
CREATE INDEX ix_prod_orders_design_id ON prod_orders (design_id);
CREATE INDEX ix_prod_orders_fabric_id ON prod_orders (fabric_id);

CREATE INDEX ix_prod_tasks_furniture_id ON prod_tasks (furniture_id);
CREATE INDEX ix_prod_tasks_prototype_id ON prod_tasks (prototype_id);
CREATE INDEX ix_prod_tasks_worker_id ON prod_tasks (worker_id);

CREATE INDEX ix_repair_notifications_repair_order_id ON repair_notifications
(repair_order_id);

CREATE INDEX ix_repair_orders_customer_id ON repair_orders (customer_id);

CREATE INDEX ix_repair_tasks_order_id ON repair_tasks (order_id);
CREATE INDEX ix_repair_tasks_worker_id ON repair_tasks (worker_id);

CREATE INDEX ix_resource_quantity_item_id ON resource_quantity (item_id);
CREATE INDEX ix_resource_quantity_prod_task_id ON resource_quantity (prod_task_id);
CREATE INDEX ix_resource_quantity_repair_task_id ON resource_quantity
(repair_task_id);
CREATE INDEX ix_resource_quantity_task_prototype_id1 ON resource_quantity
(task_prototype_id1);

CREATE INDEX ix_resources_color_id ON resources (color_id);
CREATE INDEX ix_resources_fabric_type_id ON resources (fabric_type_id);

CREATE INDEX ix_stored_resource_supplier_id ON stored_resource (supplier_id);

CREATE INDEX ix_supply_orders_item_id ON supply_orders (item_id);
CREATE INDEX ix_supply_orders_supplier_id ON supply_orders (supplier_id);

CREATE INDEX ix_task_prototypes_design_id ON task_prototypes (design_id);

CREATE INDEX ix_worker_paycheck_worker_id ON worker_paycheck (worker_id);

CREATE UNIQUE INDEX ix_workers_phone ON workers (phone);
```

В.5 Заповнення бази даних

```
namespace Fwsh.MockData;

using System;
using System.Collections.Generic;
using System.Linq;
using Fwsh.Common;
using Fwsh.Database;
using Fwsh.Logging;
using Fwsh.Utils;

public class DesignFactory : Factory<Design>
{
    Random random = new Random();

    static string cm = MeasureUnits.Centimeters;

    static Design[] data = new[] {
        new Design {
            NameKey = "prague",
            DisplayName = "Prague",
            Type = FurnitureTypes.Ottoman,
```

```
            Description = "Compact transformable ottoman couch",
            IsVisible = true,
            IsTransformable = true,
            DimCompact = new Dimensions (170, 100, 90, cm),
            DimExpanded = new Dimensions (170, 190, 90, cm)
        },
        new Design {
            NameKey = "lyra",
            DisplayName = "Lyra",
            Type = FurnitureTypes.Ottoman,
            Description = "Compact transformable couch",
            IsVisible = true,
            IsTransformable = true,
            DimCompact = new Dimensions (165, 105, 60, cm),
            DimExpanded = new Dimensions (165, 201, 60, cm)
        },
        new Design {
            NameKey = "barras",
            DisplayName = "Barras",
            Type = FurnitureTypes.Corner,
            Description = "Transformable corner couch",
            IsVisible = true,
            IsTransformable = true,
            DimCompact = new Dimensions (170, 140, 90, cm),
            DimExpanded = new Dimensions (170, 210, 90, cm)
        },
        new Design {
            NameKey = "londonese",
            DisplayName = "Londonese",
            Type = FurnitureTypes.Armchair,
            Description = "Comfortable armchair",
            IsVisible = true,
            IsTransformable = false,
            DimCompact = new Dimensions (85, 95, 92, cm)
        },
        new Design {
            NameKey = "chester",
            DisplayName = "Chester",
            Type = FurnitureTypes.Pouffe,
            Description = "Compact round pouffe",
            IsVisible = true,
            IsTransformable = false,
            DimCompact = new Dimensions (45, 45, 48, cm)
        }
    };

    public override int? FixedSize => data.Length;

    public override Design Next() => random.Choice(data);

    public override Design[] All() => data.ToArray();
}

public class BlueprintSeeder : Seeder
{
    Random random = new Random();

    Factory<Design> Designs = new DesignFactory();

    void SeedDesigns (FwshDataContext context)
    {
        var designs = this.Designs.All();

        int count = 0;
```

```
        foreach (var design in designs) {
            count++;
            design.PhotoUrls = Enumerable.Range(0, random.Next(2, 5))
                .Select(i => $"design-{count}-stub.jpg").ToList();
        }

        context.Designs.AddRange(designs);
    }

    void SeedTaskPrototypes (FwshDataContext context)
    {
        var res = context.Resources.Local;

        var designs = context.Designs.Local.ToList();

        var screwPart = res.Where(p => p.Name.Contains("Screw
55mm")).FirstOrDefault();

        var wood = res.Where(m => m.Name == "Wood").FirstOrDefault();

        var woodenSlab = res.Where(m => m.Name.Contains("slab")).FirstOrDefault();

        var pvaGlue = res.Where(m => m.Name.Contains("PVA")).FirstOrDefault();

        var foam50 = res.Where(m => m.Name.Contains("Foam 50mm")).FirstOrDefault();

        var syntp = res.Where(m => m.Name == "Synthepone").FirstOrDefault();

        var interlin = res.Where(m =>
m.Name.Contains("Interlining")).FirstOrDefault();

        var foamGlue = res.Where(m => m.Name.Contains("Foam glue")).FirstOrDefault();

        var joints = res.Where(p => p.Name.Contains("mechanism")).ToArray();

        var leg = res.Where(p => p.Name.Contains("leg")).FirstOrDefault();

        foreach (var design in designs)
        {
            var dimensions = design.DimExpanded ?? design.DimCompact
                        ?? new Dimensions { Length = 100, Width = 100 };
            double approxArea = (double)dimensions.Length
                          * (double)dimensions.Width / 10000;

            design.Tasks = new HashSet<TaskPrototype> {
                new TaskPrototype {
                    Precedence = 0,
                    Role = WorkerRoles.Carpentry,
                    Payment = 300,
                    Description = "Wooden frame manufacture",
                    Resources = new HashSet<ResourceQuantity> {
                        new ResourceQuantity {
                            Item = wood,
                            ExpectQuantity = approxArea * 0.01
                        },
                        new ResourceQuantity {
                            Item = pvaGlue,
                            ExpectQuantity = 0.075
                        },
                        new ResourceQuantity {
                            Item = screwPart,
                            ExpectQuantity = (int)approxArea * 10 + random.Next(0, 5)
                        }
                    }
                },
```

```
new TaskPrototype {
    Precedence = 1,
    Role = WorkerRoles.Carpentry,
    Payment = 300,
    Description = "Compound frame manufacture",
    Resources = new HashSet<ResourceQuantity> {
        new ResourceQuantity {
            Item = wood,
            ExpectQuantity = approxArea * 0.005
        },
        new ResourceQuantity {
            Item = woodenSlab,
            ExpectQuantity = approxArea
        },
        new ResourceQuantity {
            Item = screwPart,
            ExpectQuantity = (int)approxArea * 12 + random.Next(0, 5)
        }
    }
},
new TaskPrototype {
    Precedence = 1,
    Role = WorkerRoles.Sewing,
    Payment = 300,
    Description = "Cover manufacture",
    Resources = new HashSet<ResourceQuantity> {
        new ResourceQuantity {
            SlotName = SlotNames.Fabric,
            ExpectQuantity = approxArea * 2 + 1.5
        }
    }
},
new TaskPrototype {
    Precedence = 2,
    Role = WorkerRoles.Upholstery,
    Payment = 300,
    Description = "Upholstery",
    Resources = new HashSet<ResourceQuantity> {
        new ResourceQuantity {
            Item = foamGlue,
            ExpectQuantity = 0.025 * approxArea
        },
        new ResourceQuantity {
            Item = foam50,
            ExpectQuantity = approxArea * 1.8
        },
        new ResourceQuantity {
            Item = syntp,
            ExpectQuantity = approxArea * 1.5
        },
        new ResourceQuantity {
            Item = interlin,
            ExpectQuantity = approxArea * 2
        }
    }
},
new TaskPrototype {
    Precedence = 3,
    Role = WorkerRoles.Assembly,
    Payment = 250,
    Description = "Final assembly",
    Resources = new HashSet<ResourceQuantity> {
        new ResourceQuantity {
            Item = leg,
            ExpectQuantity = random.Next(4, 6)
```

```
                        },
                        new ResourceQuantity {
                            Item = random.Choice(joints),
                            ExpectQuantity = 2
                        }
                    }
                }
            };

            if (design.Type == FurnitureTypes.Ottoman) {
                design.Tasks.Add ( new TaskPrototype {
                    Precedence = 2,
                    Role = WorkerRoles.Carpentry,
                    Payment = 100,
                    Description = "Decorative panels manufacture",
                    Resources = new HashSet<ResourceQuantity> {
                        new ResourceQuantity {
                            SlotName = SlotNames.Decor,
                            ExpectQuantity = approxArea * 0.25
                        }
                    }
                });
            }

            design.UpdateResourceQuantities();
            design.UpdatePrice();
        }
    }

    public override void Seed (FwshDataContext context)
    {
        this.SeedDesigns(context);
        this.SeedTaskPrototypes(context);
    }
}
```

ДОДАТОК Г

Реалізація сервера WebAPI

Г.1 Приклад класу контролера

```
namespace Fwsh.WebApi.Controllers.Manager;

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using System.Text.Json;

using Fwsh.Utils;
using Fwsh.Common;
using Fwsh.Database;
using Fwsh.Logging;
using Fwsh.WebApi.FileStorage;
using Fwsh.WebApi.Requests;
using Fwsh.WebApi.Requests.Manager;
using Fwsh.WebApi.Results;
using Fwsh.WebApi.SillyAuth;
using Fwsh.WebApi.Utils;

[ApiController]
[Route("manager/designs")]
public class DesignController : FwshController
{
    const int PAGESIZE = 10;
    const int MAX_PHOTOS = 10;

    protected FileStorageProvider storage;

    public DesignController (FwshDataContext dataContext, Logger logger, FwshUser
user, FileStorageProvider storage)
    {
        this.dataContext = dataContext;
        this.logger = logger;
        this.user = user;
        this.storage = storage;
    }

    [HttpGet("list")]
    public IActionResult List (int? page = null, string type = null)
    {
        IQueryable<Design> designs = dataContext.Designs;

        if (type != null) {
            designs = designs.Where(d => d.Type == type);
        }
        if (page is int pagenumber) {
            return Ok ( designs.OrderBy(d => d.Id).Paginate (
                pagenumber, PAGESIZE, design => new DesignResult(design).Mini()
            ));
        }
```

```
        else {
            return BadRequest(new BadFieldResult("page"));
        }
    }

    [HttpGet("view/{id}")]
    public IActionResult View (int id)
    {
        Design design = dataContext.Designs
            .FirstOrDefault(d => d.Id == id);

        if (design == null) {
            return NotFound(new BadFieldResult("id"));
        }

        return Ok (new DesignResult(design).ForManager());
    }

    [HttpPost("create")]
    public IActionResult Create (DesignRequest request)
    {
        if (request.Validate().State.HasBadFields) {
            return BadRequest (new BadFieldResult(request.State.BadFields));
        }
        if (! request.State.IsValid) {
            return BadRequest (new FailResult(request.State.Message ?? "Something went
wrong"));
        }

        if (dataContext.Designs.FirstOrDefault(d => d.NameKey == request.NameKey) !=
null) {
            return BadRequest (new BadFieldResult("nameKey"));
        }

        try {
            Design design = request.Create();
            dataContext.Designs.Add(design);
            dataContext.SaveChanges();
            return Ok (new CreationResult(design.Id, $"Successfully created Design
{design.Id}"));
        }
        catch (Exception ex) {
            logger.Error(ex.ToString());
            return ServerError (new FailResult("Something went wrong while trying to
create new Design"));
        }
    }

    [HttpPost("update/{id}")]
    public IActionResult Update (int id, DesignRequest request)
    {
        if (request.Validate().State.HasBadFields) {
            return BadRequest (new BadFieldResult(request.State.BadFields));
        }
        if (! request.State.IsValid) {
            return BadRequest (new FailResult(request.State.Message ?? "Something went
wrong"));
        }

        Design design = dataContext.Designs.FirstOrDefault(d => d.Id == id);

        if (design == null) {
            return NotFound (new BadFieldResult("id"));
        }
```

```
        try {
            request.ApplyTo(design);
            dataContext.Designs.Update(design);
            dataContext.SaveChanges();
            return Ok (new CreationResult(design.Id, $"Successfully created Design
{design.Id}"));
        }
        catch (Exception ex) {
            logger.Error(ex.ToString());
            return ServerError (new FailResult("Something went wrong while trying to
create new Design"));
        }
    }

    [HttpPost("recalculate")]
    public IActionResult Recalculate (int? id = null)
    {
        IQueryable<Design> designs = dataContext.Designs
            .Include(d => d.Tasks)
            .ThenInclude(t => t.Resources)
            .ThenInclude(p => p.Item);

        if (id != null)
            designs = designs.Where(design => design.Id == id);

        int count = 0;
        foreach (var design in designs.ToList()) {
            design.UpdatePrice();
            design.UpdateResourceQuantities();
            dataContext.Designs.Update(design);
            count += 1;
        }

        try {
            dataContext.SaveChanges();
            logger.Log("Successfully recalculated {0} designs", count);
            if (id != null) {
                return Ok (new SuccessResult($"Successfully recalculated Design
{id}"));
            }
            else {
                return Ok (new SuccessResult("Successfully recalculated all
designs"));
            }
        }
        catch (Exception ex) {
            logger.Error(ex.ToString());
            return ServerError("Something went wrong while trying to recalculate
designs");
        }
    }

    [HttpPost("set-visible/{id}")]
    public IActionResult SetVisible (int id, string visible)
    {
        Design design = dataContext.Designs
            .FirstOrDefault(d => d.Id == id);

        design.IsVisible = visible?.ToLower() switch { "true" => true, _ => false };

        try {
            dataContext.Designs.Update(design);
            dataContext.SaveChanges();
            return Ok (new SuccessResult($"Successfully set visible={design.IsVisible}
for Design {id}"));
```

```
        }
        catch (Exception ex) {
            logger.Error(ex.ToString());
            return ServerError (new FailResult("Something went wrong while trying to
set visibility of Design"));
        }
    }

    [HttpPost("attach-photos/{designId}")]
    public IActionResult AttachPhotos (int designId)
    {
        Design design = dataContext.Designs
            .FirstOrDefault(d => d.Id == designId);

        if (design == null) {
            return BadRequest (new BadFieldResult("id"));
        }

        var requestPhotos = this.Request.Form.Files.ToList();
        var photoUrls = design.PhotoUrls.ToList();

        int count = 0, pos = design.PhotoUrls.Count + 1;

        foreach (var photo in requestPhotos) {
            if (photoUrls.Count >= MAX_PHOTOS) break;
            string ext = photo.FileName.Split('.').LastOrDefault();
            string url = $"design-{design.Id}-{pos}-{Guid.NewGuid()}.{ext}";
            if (storage.TrySave(photo.OpenReadStream(), url)) {
                photoUrls.Add(url);
                count += 1;
                pos += 1;
            }
        }

        try {
            design.PhotoUrls = photoUrls;
            dataContext.Designs.Update(design);
            dataContext.SaveChanges();
            return Ok(new SuccessResult($"Successfully attached {count} photos to
Design {designId}"));
        }
        catch (Exception ex) {
            logger.Error(ex.ToString());
            return ServerError(new FailResult("Something went wrong while trying to
attach photos"));
        }
    }

    [HttpDelete("delete/{id}")]
    public IActionResult Delete (int id)
    {
        Design design = dataContext.Designs
            .Include(design => design.Tasks)
            .ThenInclude(task => task.Resources)
            .FirstOrDefault(design => design.Id == id);

        if (design == null) {
            return NotFound(new BadFieldResult("id"));
        }

        bool canDelete = dataContext.ProdOrders
            .Where(order => order.IsActive == true)
            .Where(order => order.DesignId == design.Id)
            .Count() == 0;
```

```
        if (! canDelete) {
            return BadRequest(new FailResult($"Can not delete Design {id} because of
dependent production orders"));
        }

        foreach (var photoUrl in design.PhotoUrls) {
            storage.TryDelete(photoUrl);
        }

        try {
            dataContext.Designs.Remove(design);
            dataContext.SaveChanges();
            return Ok(new DeletionResult(id, $"Successfully deleted Design {id}"));
        }
        catch (Exception ex) {
            logger.Error(ex.ToString());
            return ServerError(new FailResult("Something went wrong"));
        }
    }
}
```

Г.2 Приклад класу запиту

```
namespace Fwsh.WebApi.Requests.Manager;

using System;
using System.Text.RegularExpressions;
using Fwsh.Common;
using Fwsh.WebApi.Validation;

public class DesignRequest: Request,
    CreationRequest<Design>, UpdateRequest<Design>
{
    public string Type { get; set; }
    public string NameKey { get; set; }
    public string DisplayName { get; set; }
    public string Description { get; set; }
    public bool IsVisible { get; set; }
    public bool IsTransformable { get; set; }
    public Dimensions DimCompact { get; set; }
    public Dimensions DimExpanded { get; set; }

    protected override void OnValidation (ObjectValidator validator)
    {
        validator.Property("type", this.Type)
            .NotNull().Condition(FurnitureTypes.Contains(this.Type));

        validator.Property("nameKey", this.NameKey)
            .NotNull().Match(NameKeyRegex);

        validator.Property("displayName", this.DisplayName)
            .NotNull().Match(DisplayNameRegex);

        validator.Property("description", this.Description)
            .NotNull().LengthInRange(100, 10000);

        validator.Property("dimCompact", this.DimCompact)
            .NotNull();

        validator.Property("dimExpanded", this.DimExpanded)
```

```
            .Condition((! this.IsTransformable) || (this.DimExpanded is Dimensions));
    }

    public Design Create()
    {
        var design = new Design();
        design.NameKey = this.NameKey;
        this.ApplyTo(design);
        return design;
    }

    public void ApplyTo (Design design)
    {
        design.Type = this.Type;
        design.DisplayName = this.DisplayName;
        design.Description = this.Description;
        design.IsVisible = this.IsVisible;
        design.IsTransformable = this.IsTransformable;
        design.DimCompact = this.DimCompact;
        design.DimExpanded = this.DimExpanded;
    }

    static Regex NameKeyRegex = new Regex(@"^[a-z0-9\-]{3,24}$");
    static Regex DisplayNameRegex = new Regex(@"^[A-Za-z0-9\s\-]{3,24}$");
}
```

Г.3 Приклад класу результату

```
namespace Fwsh.WebApi.Results;

using System;
using System.Collections.Generic;
using System.Linq;
using Fwsh.Common;

public class DesignResult : Result, IResultBuilder<DesignResult>
{
    private Design design { get; set; }

    public int Id { get; set; }

    public string NameKey { get; set; }
    public string Type { get; set; }
    public string DisplayName { get; set; }
    public string Description { get; set; }
    public bool IsVisible { get; set; }
    public bool IsTransformable { get; set; }
    public Dimensions DimCompact { get; set; }
    public Dimensions DimExpanded { get; set; }
    public double FabricUsage { get; set; }
    public double DecorUsage { get; set; }
    public int Price { get; set; }
    public DateTime CreatedAt { get; set; }
    public int DaysSinceCreated => (DateTime.UtcNow - this.CreatedAt).Days;
    public DateTime? RecalculatedAt { get; set; }

    public IReadOnlyList<string> PhotoUrls { get; set; }

    public DesignResult () { }
```

```
    public DesignResult (Design design)
    {
        this.design = design;
    }


    public DesignResult Mini ()
    {
        return new DesignResult() {
            Id = design.Id,
            NameKey = design.NameKey,
            Type = design.Type,
            DisplayName = design.DisplayName,
            IsVisible = design.IsVisible,
            Price = design.Price,
            DimCompact = design.DimCompact,
            DimExpanded = design.DimExpanded,
            CreatedAt = design.CreatedAt,
            PhotoUrls = design.PhotoUrls
        };
    }


    public DesignResult ForCustomer ()
    {
        return new DesignResult() {
            Id = design.Id,
            NameKey = design.NameKey,
            Type = design.Type,
            DisplayName = design.DisplayName,
            Price = design.Price,
            CreatedAt = design.CreatedAt,
            Description = design.Description,
            IsVisible = design.IsVisible,
            IsTransformable = design.IsTransformable,
            DimCompact = design.DimCompact,
            DimExpanded = design.DimExpanded,
            FabricUsage = (design.FabricUsage > 0)? 1 : 0,
            DecorUsage = (design.DecorUsage > 0)? 1 : 0,
            PhotoUrls = design.PhotoUrls
        };
    }


    public DesignResult ForWorker ()
    {
        var result = this.ForCustomer();
        result.FabricUsage = design.FabricUsage;
        result.DecorUsage = design.DecorUsage;
        result.RecalculatedAt = design.RecalculatedAt;
        return result;
    }


    public DesignResult ForManager ()
    {
        return this.ForWorker();
    }

}
```

Г.4 Клас Startup

```
namespace Fwsh.WebApi;

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Cors.Infrastructure;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.EntityFrameworkCore;

using Fwsh.Utils;
using Fwsh.Common;
using Fwsh.Database;
using Fwsh.Logging;
using Fwsh.WebApi.FileStorage;
using Fwsh.WebApi.Logging;
using Fwsh.WebApi.SillyAuth;
using Fwsh.WebApi.RouteGuard;
using Fwsh.WebApi.Utils;

public class Startup
{
    public void ConfigureServices (IServiceCollection services)
    {
        string logfile = env.get("API_LOGFILE");

        if (logfile == null || logfile == "console")
            services.AddSingleton<Logger, ConsoleLogger>();
        else
            services.AddSingleton<Logger>(FileLogger.To(logfile));

        if (env.get("DB_HOST") == "memory")
            services.AddDbContext<FwshDataContext,
            FwshDataContextInMemory>(ServiceLifetime.Scoped);
        else
            services.AddDbContext<FwshDataContext,
            FwshDataContextPostgres>(ServiceLifetime.Scoped);

        services.AddSingleton<FileStorageProvider>
            (new PhysicalFileStorageProvider(env.get("UPLOAD_DIR")));

        services.AddSingleton<FwshUserStorage, FwshUserStorageInMemory>();

        services.AddScoped<FwshUser>();

        services.AddRouting();

        services.AddControllers().AddControllersAsServices();

        this.DoStartupRoutine(services);
    }

    public void DoStartupRoutine (IServiceCollection services)
    {
        var serviceProvider = services.BuildServiceProvider();
```

```
    var serviceScope = serviceProvider.CreateScope();
    var dataContext = serviceProvider.GetService<FwshDataContext>();
    var logger = serviceProvider.GetService<Logger>();

    // create first manager account if there is nothing yet
    bool managerExists = dataContext.Workers
        .Where(E.Worker.IsManager)
        .Count() > 0;

    if (! managerExists) {
        var manager = new Worker() {
            Name = "Default Manager",
            Phone = env.get("MGR_LOGIN") ?? "admin",
            Password = env.get("MGR_PASSWORD").QuickHash(),
            Roles = new[] { WorkerRoles.Management }
        };
        dataContext.Workers.Add(manager);
        dataContext.SaveChanges();
        logger.Log("Created default manager");
    }

    // recalculate resource prices
    var priceController = serviceProvider
        .GetService<Fwsh.WebApi.Controllers.Manager.PriceFormationController>();
    priceController.UpdateDefaultPrices();

    // recalculate design prices
    var designController = serviceProvider
        .GetService<Fwsh.WebApi.Controllers.Manager.DesignController>();
    designController.Recalculate();

    serviceScope.Dispose();
    serviceProvider.Dispose();
}

public void Configure (IApplicationBuilder app, IWebHostEnvironment environment)
{
    // Enable routing
    app.UseRouting();

    // Enable CORS
    app.UseCors (policy => {
        policy.AllowAnyOrigin()
            .AllowAnyMethod()
            .AllowAnyHeader()
            .WithExposedHeaders("Authorization");
    });

    // Console logging
    app.UseMiddleware<HttpRequestLoggerMiddleware>();

    if (env.isDevelopment) {
        app.UseStaticFiles ("/files", env.get("UPLOAD_DIR"));
    }

    // Custom silly authentication
    app.UseMiddleware<SillyAuthMiddleware>();

    // Custom route guard
    app.UseMiddleware<RouteGuardMiddleware>();

    // Configure endpoints
    app.UseEndpoints (endpoints => {
        endpoints.MapControllers();
    });
```

```
        app.Run (async (context) => {
            var request = context.Request;
            var response = context.Response;

            response.StatusCode = StatusCodes.Status404NotFound;

            if (env.isDevelopment) {
                await response.WriteAsJsonAsync (new {
                    Message = $"Nothing here on {request.Path}",
                    Host = request.Host.ToString(),
                    Method = request.Method.ToString(),
                    Path = request.Path.ToString(),
                    Query = String.Join(", ", request.Query
                                        .Select(kv => kv.Key+"="+kv.Value))
                });
            }
            else {
                await response.WriteAsJsonAsync<object>(null);
            }
        });
    }
}
```

ДОДАТОК Д

Реалізація клієнтської частини

Д.1 Компонент ImageGallery

```
<template>
<div>
    <div class="photo-gallery mar-b-1">
        <template v-for="(item, i) in items">
            <slot :active="data.current == i" :item="item"></slot>
        </template>
    </div>
    <div v-if="items?.length" class="photo-gallery-controls flex-stripe accent-gray">
        <button class="button button-secondary" @click="goToPrevious"
            :disabled="data.current<=minPos">&lt;</button>
        <span class="text-center flex-grow">{{galleryInfo}}</span>
        <button class="button button-secondary" @click="goToNext"
            :disabled="data.current>=maxPos">&gt;</button>
    </div>
</div>
</template>

<script setup>
import { ref, reactive, computed } from "vue"

const props = defineProps({
    items: Array
})

const data = reactive({
    current: 0
})

const minPos = computed(() => 0)
const maxPos = computed(() => props.items?.length - 1)

const galleryInfo = computed(() =>
       `${data.current + 1} / ${props.items?.length ?? 0}`)

function goToPrevious() {
    if (data.current > minPos.value) data.current -= 1
}

function goToNext() {
    if (data.current < maxPos.value) data.current += 1
}

</script>
```

Д.2 Компонент designs/create

```
<template>
<Bread>
    <Crumb to="/">fwsh</Crumb>
    <Crumb to="/blueprints">{{locale.blueprint.plural}}</Crumb>
    <Crumb last>{{locale.design.plural}}</Crumb>
</Bread>
<div class="width-container card pad-1 mar-b-2">
    <h2 class="mar-b-1">{{locale.design.create}}</h2>
    <div v-if="data.success">
        <p>{{locale.design.creationMessage}}</p>
        <router-link :to="`/blueprints/designs/view/${data.id}`" class="link">
            {{locale.action.details}}</router-link>
    </div>
    <DesignEdit v-else
        :design="data.design"
        :photos="data.photos"
        :badFields="data.badFields"
        :errorMessage="data.errorMessage"
        @submit="createDesign"
        @reset="resetDesign"
        @attach-photo="attachPhoto" />
</div>
</template>

<script setup>
import { useRouter } from "vue-router"
import { ref, reactive, inject } from "vue"
import { arrayToDict } from "@common/utils"
import { Bread, Crumb } from "@common/comp/layout"
import DesignEdit from "@/comp/edits/DesignEdit.vue"

const locale = inject("locale")
const axios = inject("axios")

function designTemplate() {
    return {
        dimCompact: { measureUnit: "cm" },
        dimExpanded: { measureUnit: "cm" }
    }
}

let data = reactive({
    photos: [ ],
    design: designTemplate()
})

function createDesign() {
    console.log(data.design)
    axios.post({
        url: "/manager/designs/create",
        data: data.design
    })
    .then(({ status, data: response }) => {
        if (response.success) {
            data.id = response.id
            attachPhotosToDesign()
        }
        else if (response.badFields) {
            data.badFields = arrayToDict(response.badFields)
            data.errorMessage = locale.value.formatBadFields(response.badFields,
                                                    locale => locale.design)
```

```
        }
    })
}

function attachPhotosToDesign() {
    let formData = new FormData()
    for (const photo of data.photos) formData.append("files", photo)
    axios.post({
        url: `/manager/designs/attach-photos/${data.id}`,
        contentType: "multipart/form-data",
        data: formData
    })
    .then(({ status, data: response }) => {
        if (response.success) {
            data.success = true
        }
    })
}

function resetDesign() {
    data.photos = [ ]
    data.badFields = { }
    data.design = designTemplate()
}

function attachPhoto(photos) {
    data.photos = [...data.photos, ...photos]
}

</script>
```

Д.3 Компонент DesignView

```
<template>
<div class="width-container card pad-1 mar-b-2">
    <div class="flex-stripe mar-b-1">
        <h2>{{design.displayName}}
         <span class="text-thin text-gray">#{{design.id}}</span></h2>
        <span class="flex-grow"></span>
        <router-link :to="`/blueprints/taskprototypes/list?design=${design.id}`"
            class="button button-secondary">{{locale.task.plural}}</router-link>
    </div>
    <table class="kvtable stripes mar-b-2">
        <tr>
            <td>{{locale.common.id}}</td>
            <td>#{{design.id}}</td>
        </tr>
        <tr>
            <td>{{locale.design.displayName}}</td>
            <td>{{design.displayName}}</td>
        </tr>
        <tr>
            <td>{{locale.design.type}}</td>
            <td>{{locale.furnitureTypes[design.type]}} ({{design.type}})</td>
        </tr>
        <tr>
            <td>{{locale.design.isVisible}}</td>
            <td>{{locale.yesNo[design.isVisible]}}</td>
        </tr>
        <tr>
```

```
            <td>{{locale.design.isTransformable}}</td>
            <td>{{locale.yesNo[design.isTransformable]}}</td>
        </tr>
        <tr>
            <td>{{locale.design.dimCompact}}</td>
            <td>{{design.dimCompact.width}} x {{design.dimCompact.length}} x
                {{design.dimCompact.height}} cm</td>
        </tr>
        <tr v-if="design.isTransformable">
            <td>{{locale.design.dimExpanded}}</td>
            <td>{{design.dimExpanded.width}} x {{design.dimExpanded.length}} x
                {{design.dimExpanded.height}} cm</td>
        </tr>
        <tr>
            <td>{{locale.design.price}}</td>
            <td>{{design.price}} &#8372;  
                <button class="button button-inline"
                    @click="()=> emit('click-recalculate')">
                    {{locale.action.update}}</button></td>
        </tr>
        <tr>
            <td>{{locale.design.description}}</td>
            <td>{{design.description}}</td>
        </tr>
    </table>
    <div class="fancy-group mar-b-1">
        <header>{{locale.photo.plural}}</header>
        <main>
            <ImageGallery :items="design.photoUrls">
                <template v-slot="{ active, item }">
                    <img :src="cdnResolve(item)" :class="{ 'visible': active }">
                </template>
            </ImageGallery>
        </main>
    </div>
    <div class="flex-stripe flex-pad-1">
        <button class="button button-primary" @click="()=>emit('click-edit')">
            {{locale.action.edit}}
        </button>
        <span class="flex-grow text-right">{{message}} </span>
        <button class="button button-secondary accent-gray"
            @click="()=>emit('click-toggle-visible')">
            {{locale.setVisible[design.isVisible]}}
        </button>
    </div>
</div>
</template>
<script setup>
import { cdnResolve } from "@common/utils"
import { inject } from "vue"
import { ImageGallery } from "@common/comp/layout"

const locale = inject("locale")

const props = defineProps({
    design: Object,
    message: String
})

const emit = defineEmits([
    "click-edit",
    "click-toggle-visible",
    "click-recalculate"
])
</script>
```

Д.4 Компонент DesignEdit

```
<template>
    <inputbox type="text" v-model="design.nameKey"
        :disabled="mode=='edit'" :invalid="badFields?.nameKey">
        {{locale.design.nameKey}}
    </inputbox>
    <inputbox type="text" v-model="design.displayName"
        :invalid="badFields?.displayName">
        {{locale.design.displayName}}
    </inputbox>
    <groupbox :invalid="badFields?.type">
        <template v-slot:header>{{locale.design.type}}</template>
        <radiobox v-for="type of designTypes" v-model="design.type" :value="type">
            <span>{{locale.furnitureTypes[type]}} ({{type}})</span>
        </radiobox>
    </groupbox>
    <div class="mar-b-1">
        <checkbox v-model="design.isVisible">
            {{locale.design.isVisible}}</checkbox>
    </div>
    <div class="mar-b-1">
        <checkbox v-model="design.isTransformable">
            {{locale.design.isTransformable}}</checkbox>
    </div>
    <groupbox :invalid="badFields?.dimCompact">
        <template v-slot:header>{{locale.design.dimCompact}}</template>
        <inputbox v-for="dimension of ['width', 'length', 'height']"
            type="number" v-model="design.dimCompact[dimension]">
            {{locale.design[dimension]}}, cm
        </inputbox>
    </groupbox>
    <groupbox v-if="design.isTransformable" :invalid="badFields?.dimExpanded">
        <template v-slot:header>{{locale.design.dimExpanded}}</template>
        <inputbox v-for="dimension of ['width', 'length', 'height']"
            type="number" v-model="design.dimExpanded[dimension]">
            {{locale.design[dimension]}}, cm
        </inputbox>
    </groupbox>
    <textbox v-model="design.description" :invalid="badFields?.description">
        {{locale.design.description}}
    </textbox>
    <groupbox>
        <template v-slot:header>{{locale.photo.plural}}</template>
        <div class="preview-photo-gallery">
            <div v-for="url of design.photoUrls">
                <img :src="cdnResolve(url)">
            </div>
            <div v-for="photo of photos">
                <img :src="URL.createObjectURL(photo)"
                @load="URL.revokeObjectURL(photo)">
            </div>
        </div>
        <div>
            <input type="file" multiple id="input-photos" ref="photoInput"
                @change="photoInputChanged" class="hidden">
            <label for="input-photos"
                class="button button-secondary button-block accent-gray">
                + {{locale.action.addPhotos}}
            </label>
        </div>
    </groupbox>
```

```
    <div class="mar-b-2">
        <span class="text-error">{{errorMessage}}</span>
    </div>
    <div class="flex-stripe flex-pad-1">
        <span class="flex-grow"></span>
        <button class="button button-inline accent-bad"
            @click="()=> emit('reset')">{{locale.action.clear}}</button>
        <button class="button button-primary"
            @click="()=> emit('submit')">{{locale.action.submit}}</button>
    </div>
</template>

<script setup>
import { FurnitureTypes } from "@common"
import { cdnResolve } from "@common/utils"
import { ref, reactive, computed, inject } from "vue"
import { Groupbox, Inputbox, Textbox, Checkbox, Radiobox } from "@common/comp/ctrl"

const URL = window.URL

const locale = inject("locale")

const props = defineProps({
    photos: Array,
    design: Object,
    badFields: Object,
    errorMessage: String
})

const mode = computed(()=> (!props.design.id) ? "create" : "edit")

const designTypes = Object.values(FurnitureTypes).filter(v => !!v)

function consoleLogDesign() {
    console.log(JSON.stringify(props.design))
}

const emit = defineEmits([
    "submit",
    "reset",
    "attach-photo",
    "delete-photo",
])

const photoInput = ref(null)

function photoInputChanged() {
    console.log([...photoInput.value.files])
    emit("attach-photo", [...photoInput.value.files])
}

</script>
```

ДОДАТОК Е

Скріншоти до керівництва користувача

Е.1 Робота з обліковими записами

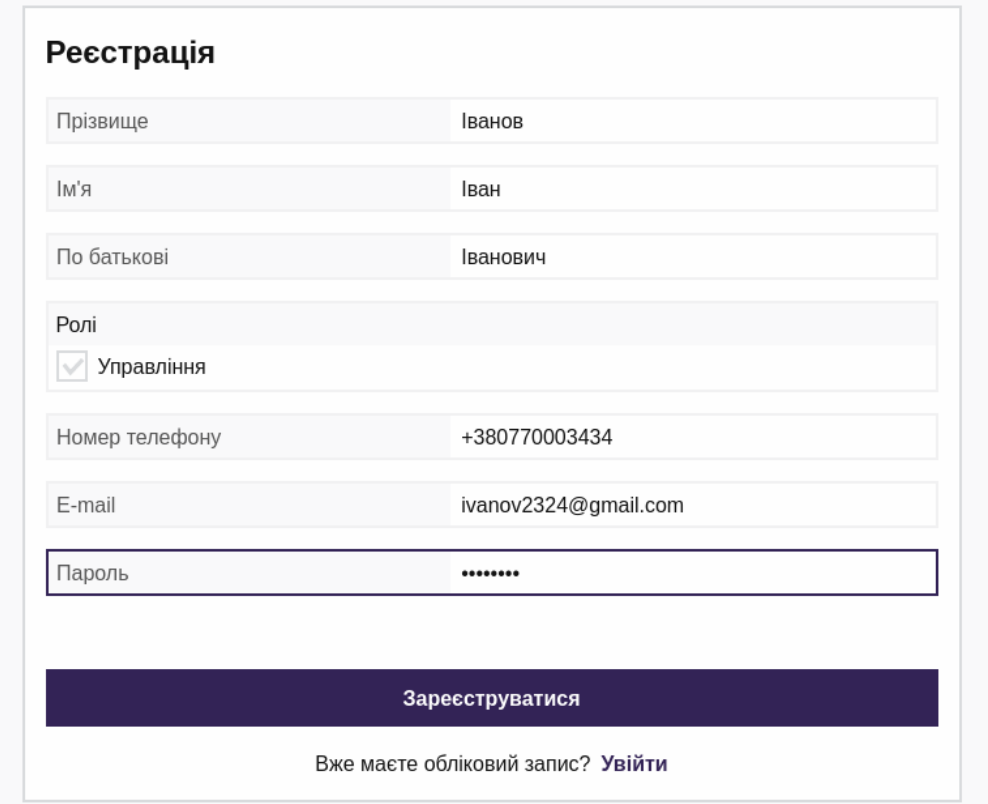

Рисунок Е.1 – Реєстрація менеджера

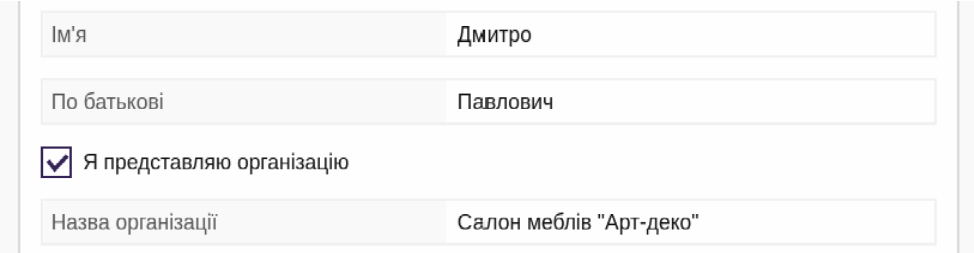

Рисунок Е.2 – Реєстрація покупця, що представляє організацію

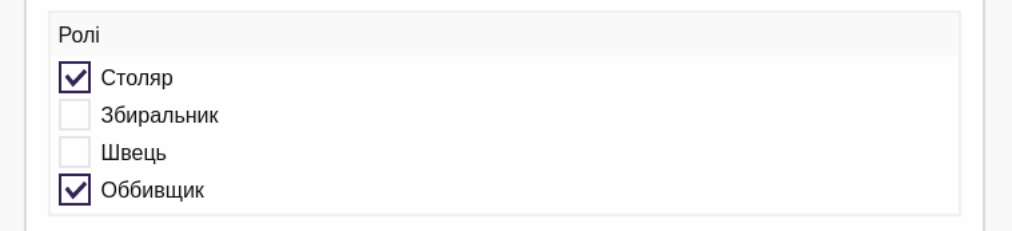

Рисунок Е.3 – Реєстрація робітника

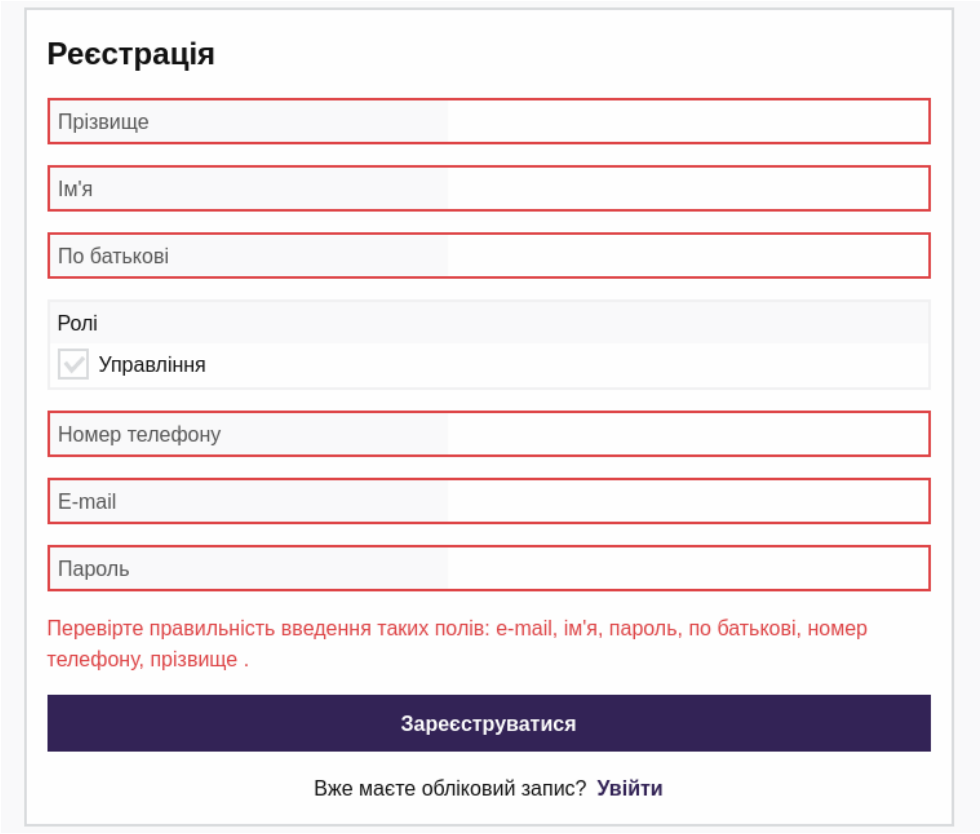

Рисунок Е.4 – Повідомлення про некоректні дані

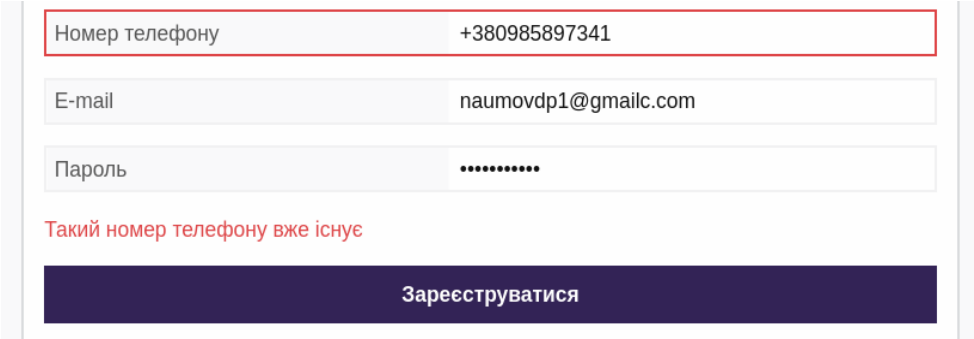

Рисунок Е.5 – Повторне використання номера телефону

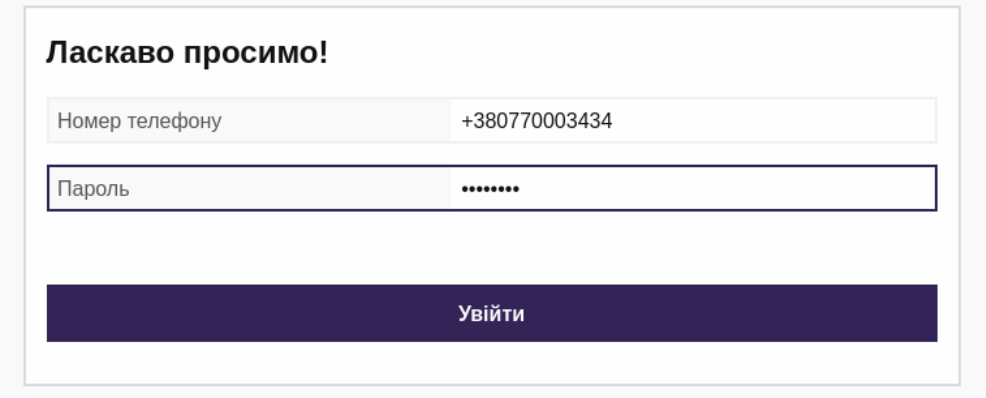

Рисунок Е.6 – Вхід в обліковий запис

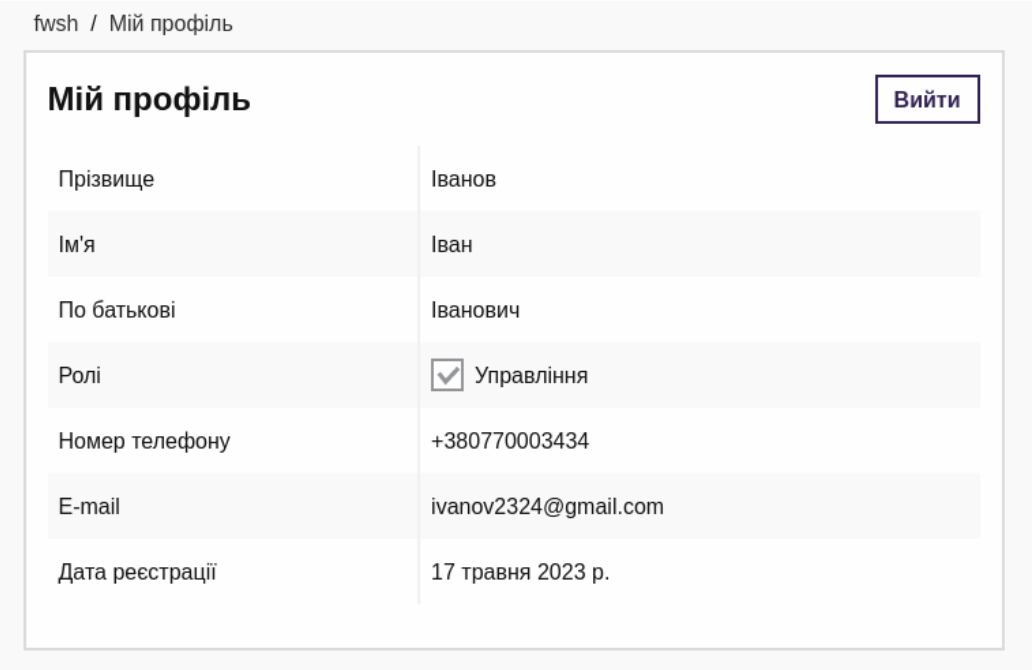

Рисунок Е.7 – Сторінка «Мій профіль»

Е.2 Використання панелі менеджера

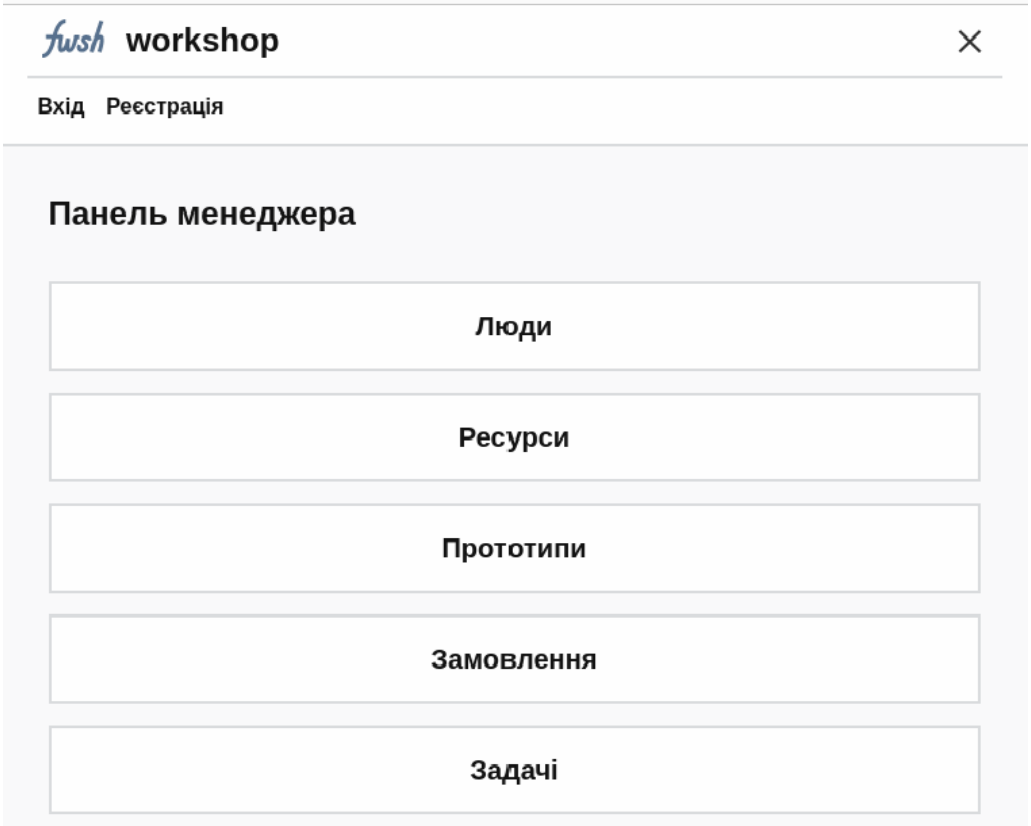

Рисунок Е.8 – Головна сторінка панелі менеджера

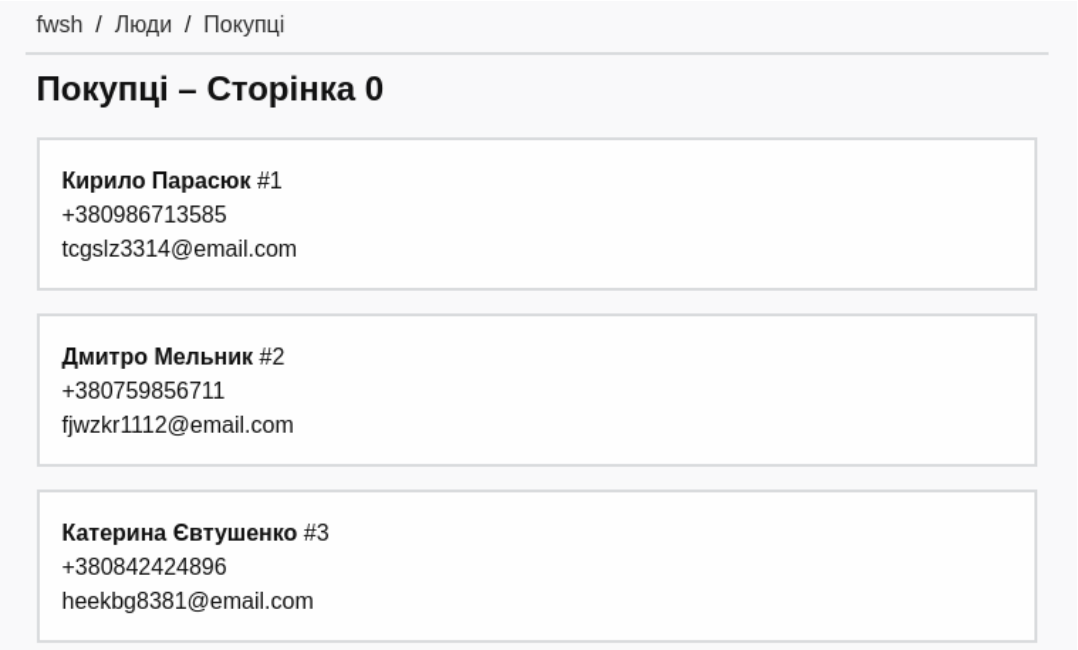

Рисунок Е.9 – Перегляд списку покупців

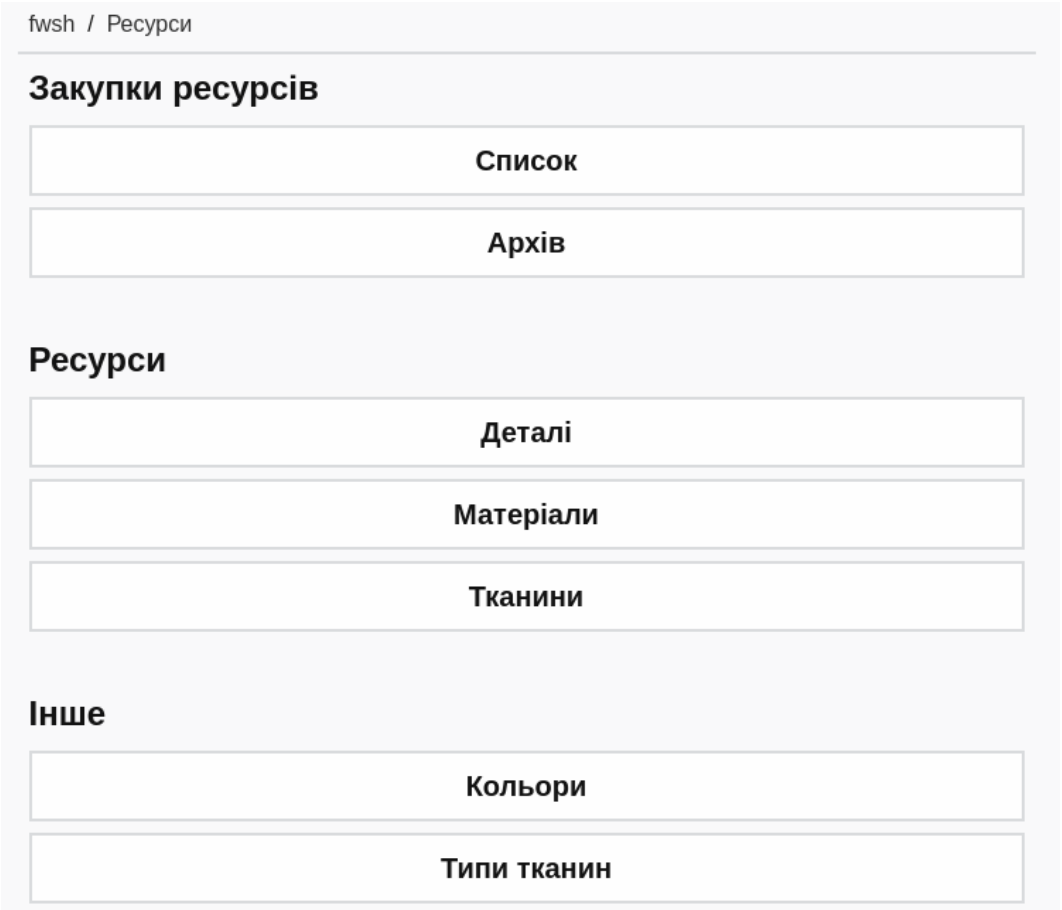

Рисунок Е.10 – Перегляд меню ресурсів

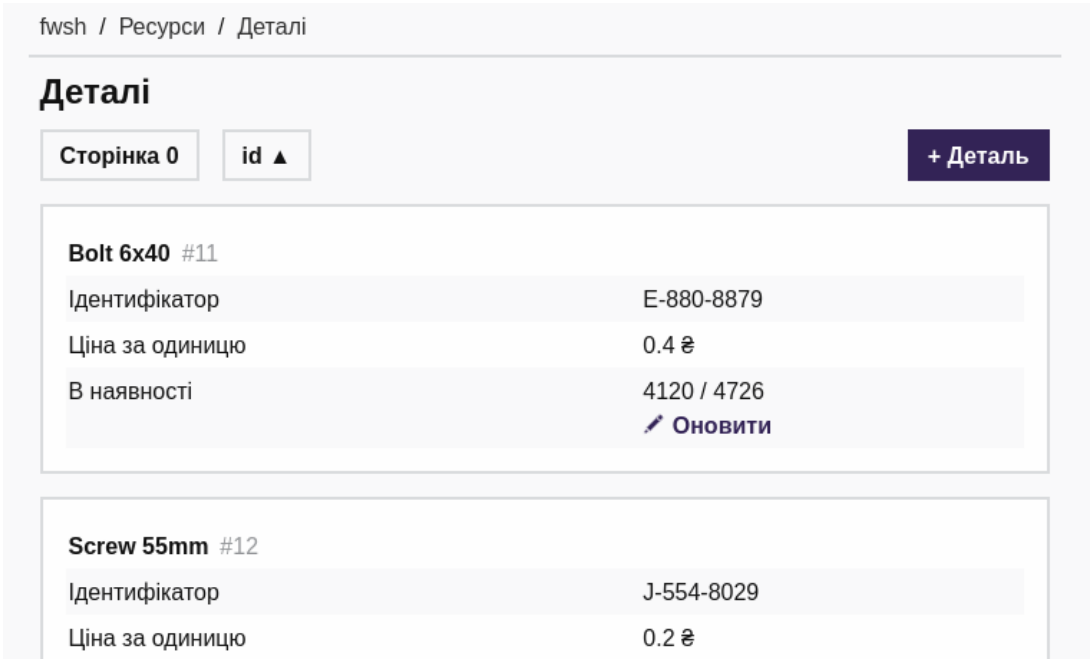

Рисунок Е.11 – Перегляд списку деталей

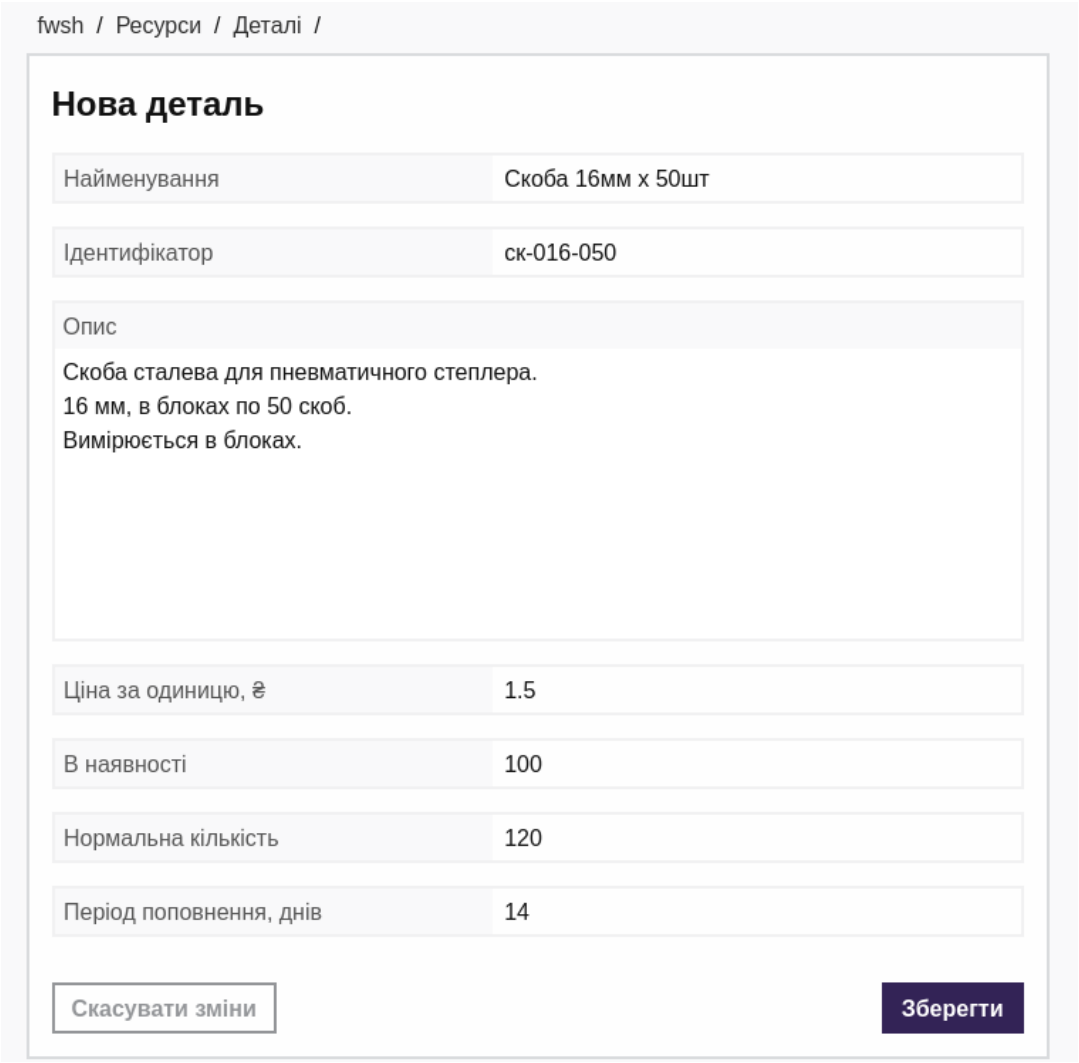

Рисунок Е.12 – Редагування ресурсу

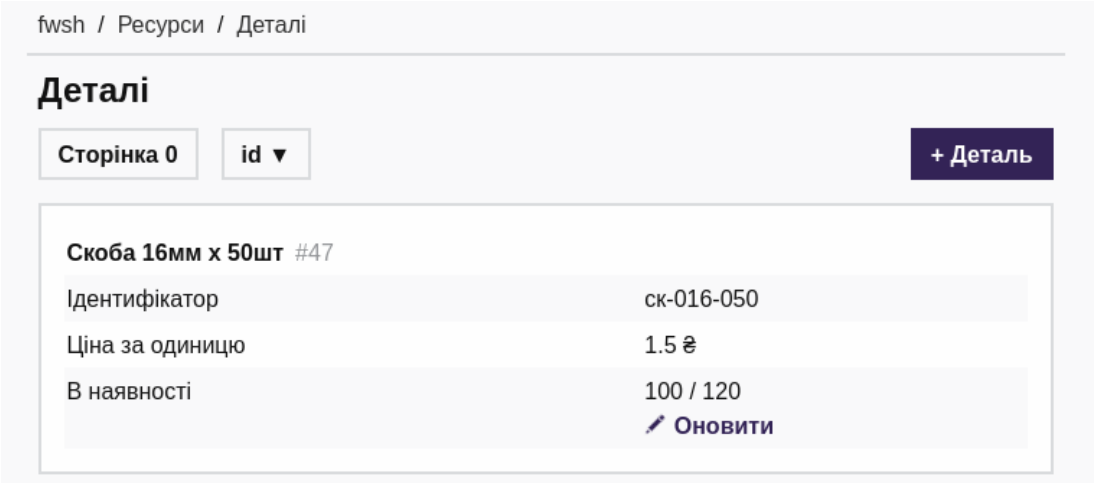

Рисунок Е.13 – Перегляд створеної деталі в списку

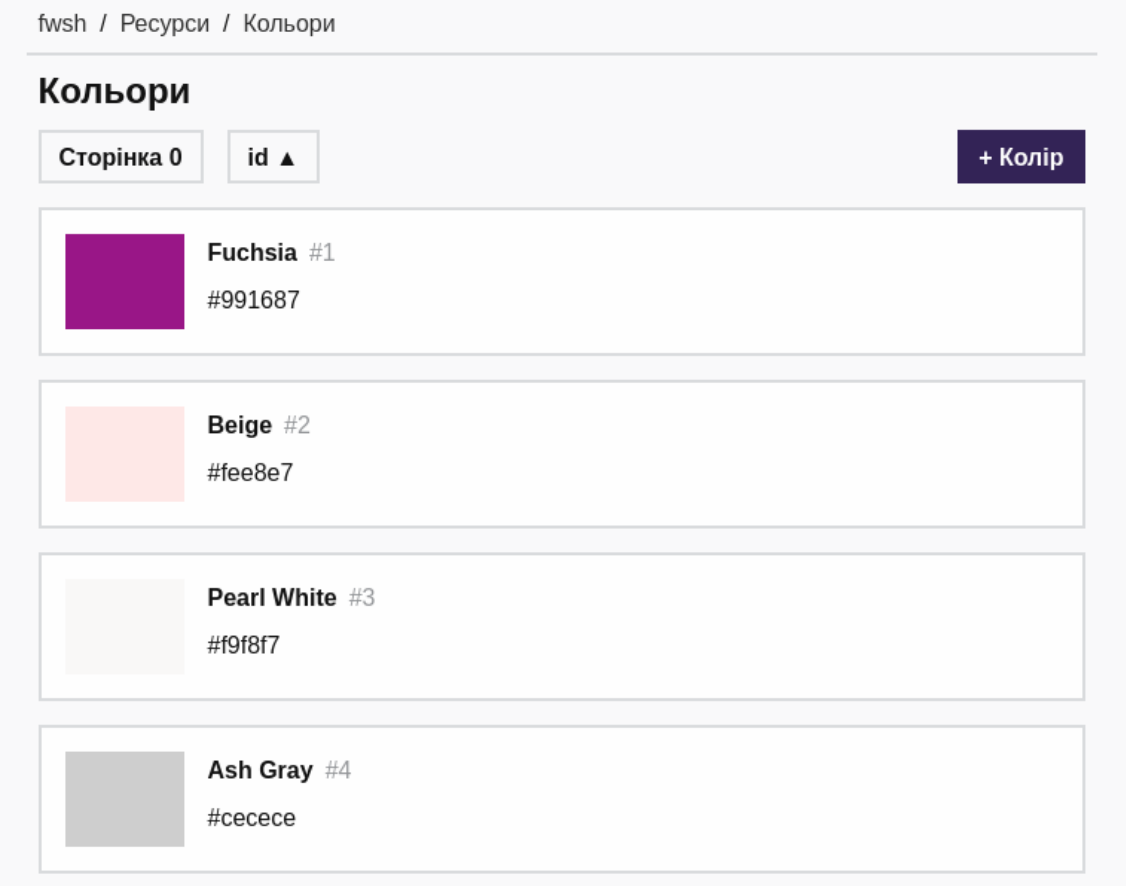

Рисунок Е.14 – Перегляд списку кольорів

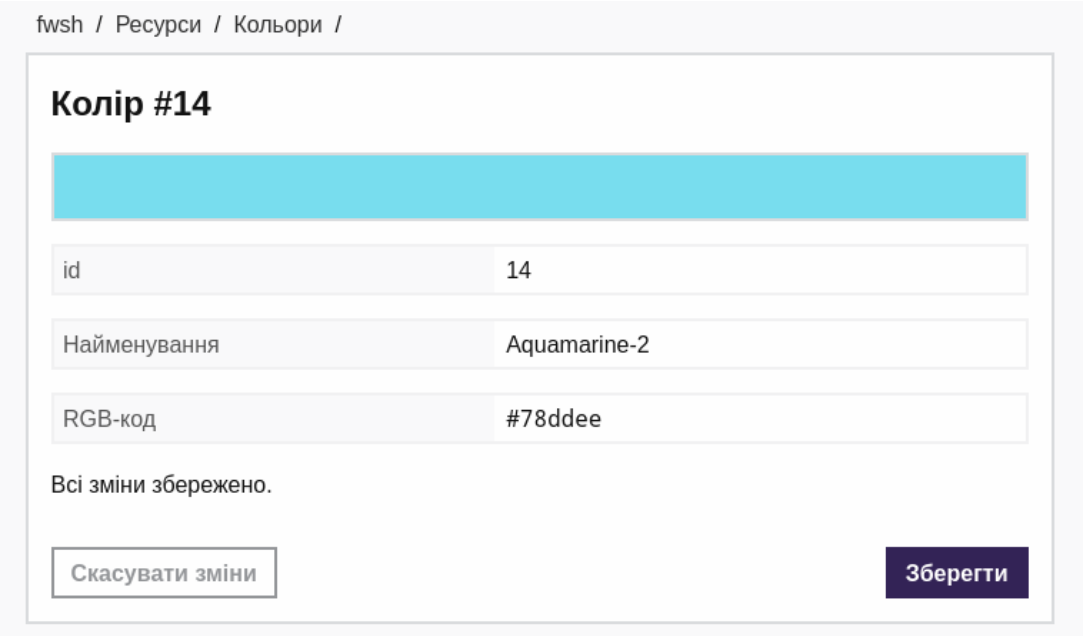

Рисунок Е.15 – Редагування кольору

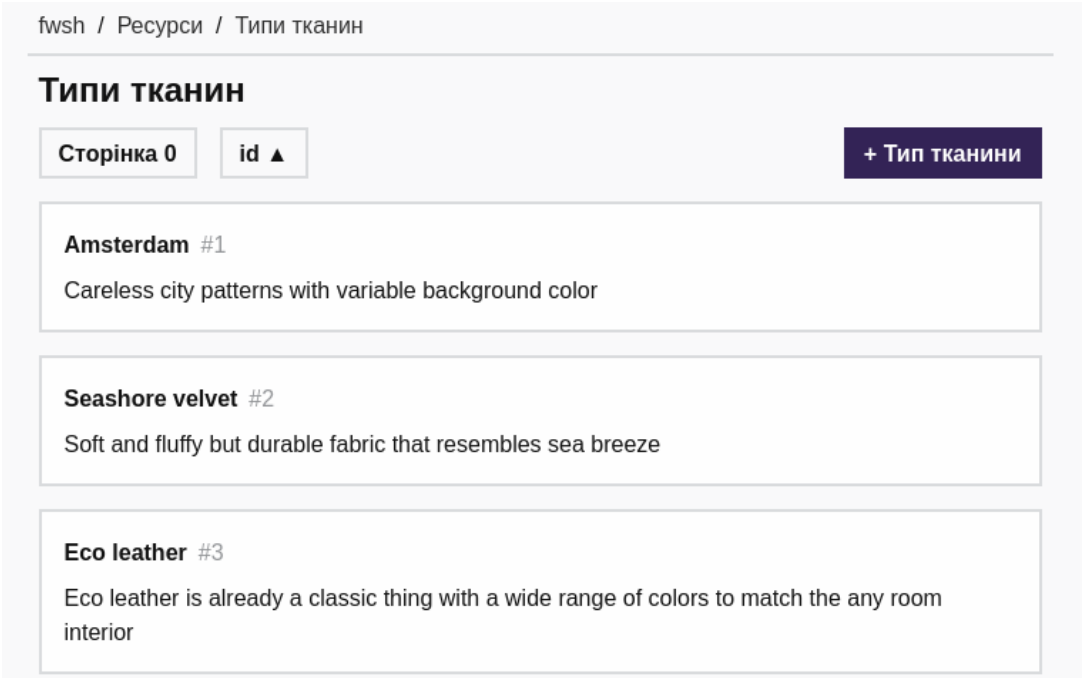

Рисунок Е.16 – Перегляд списку типів тканин

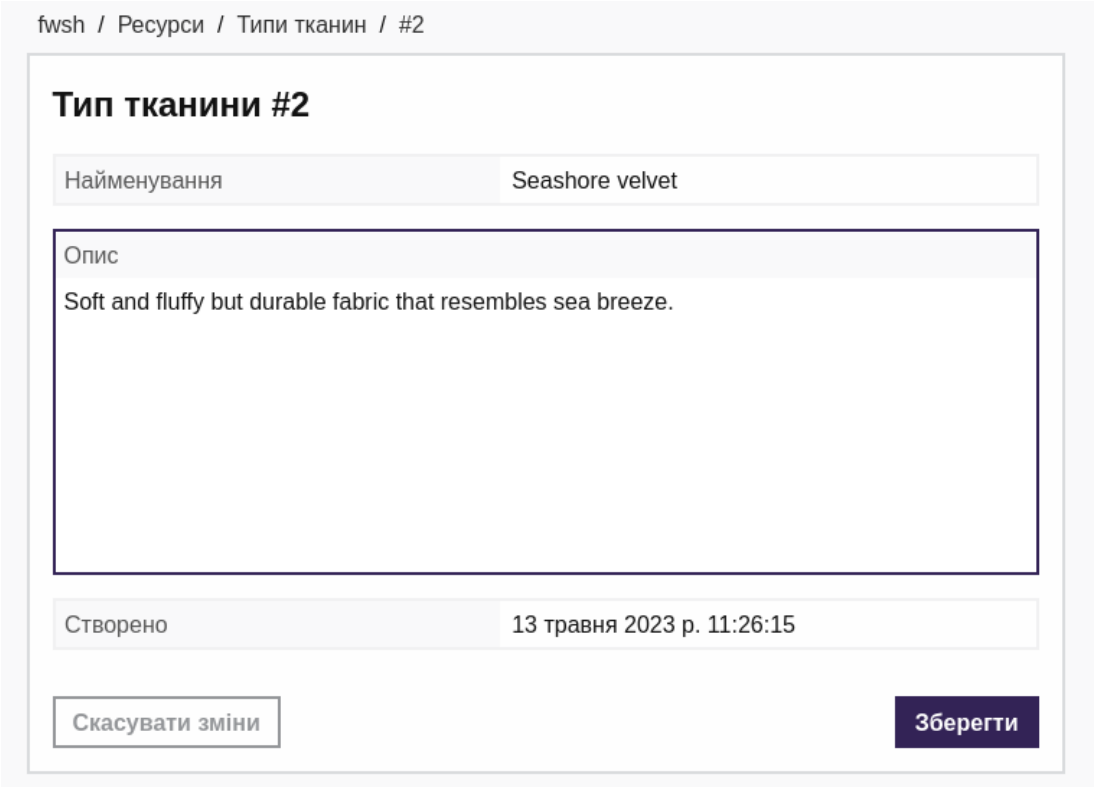

Рисунок Е.17 – Редагування типу тканини

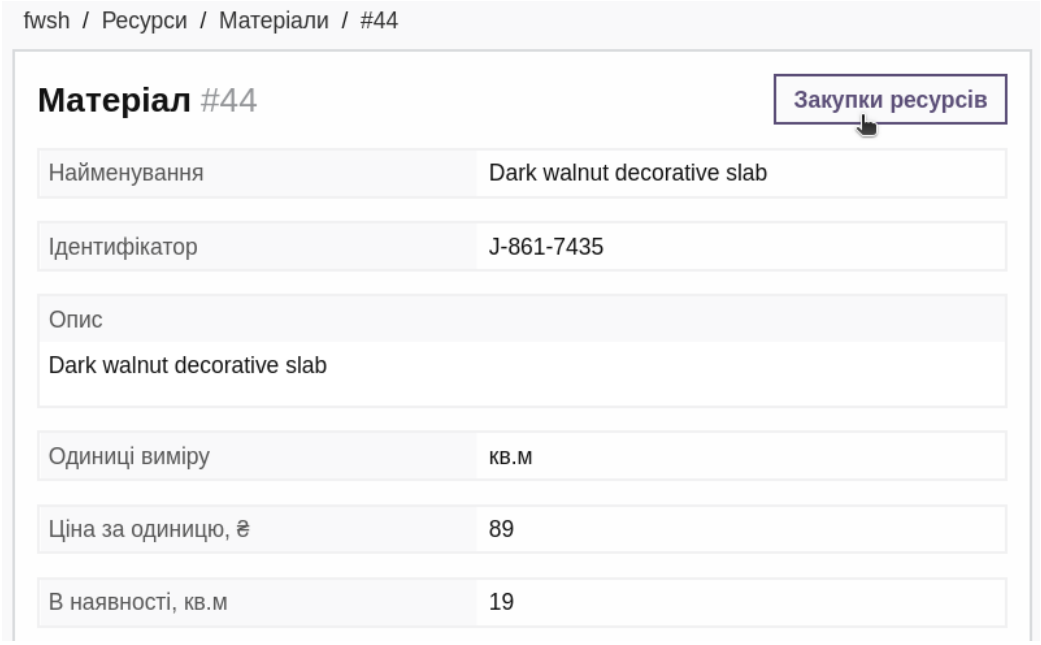

Рисунок Е.18 – Сторінка з детальним описом ресурсу

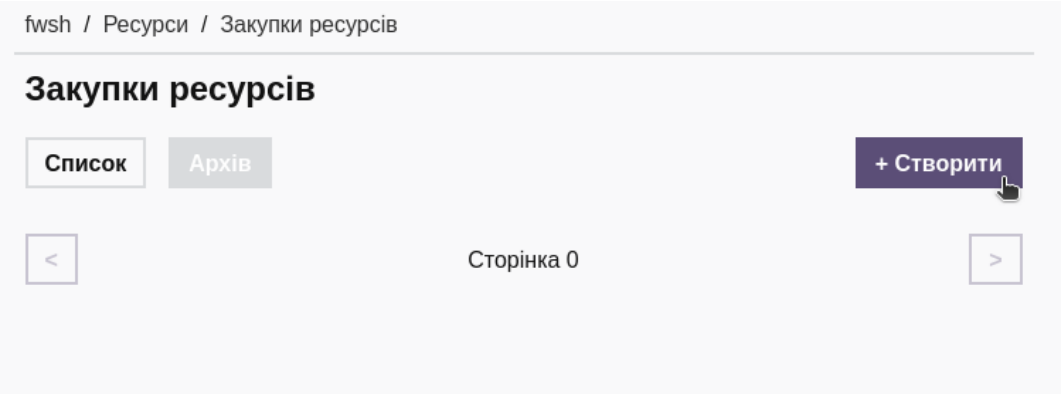

Рисунок Е.19 – Список закупок ресурсів

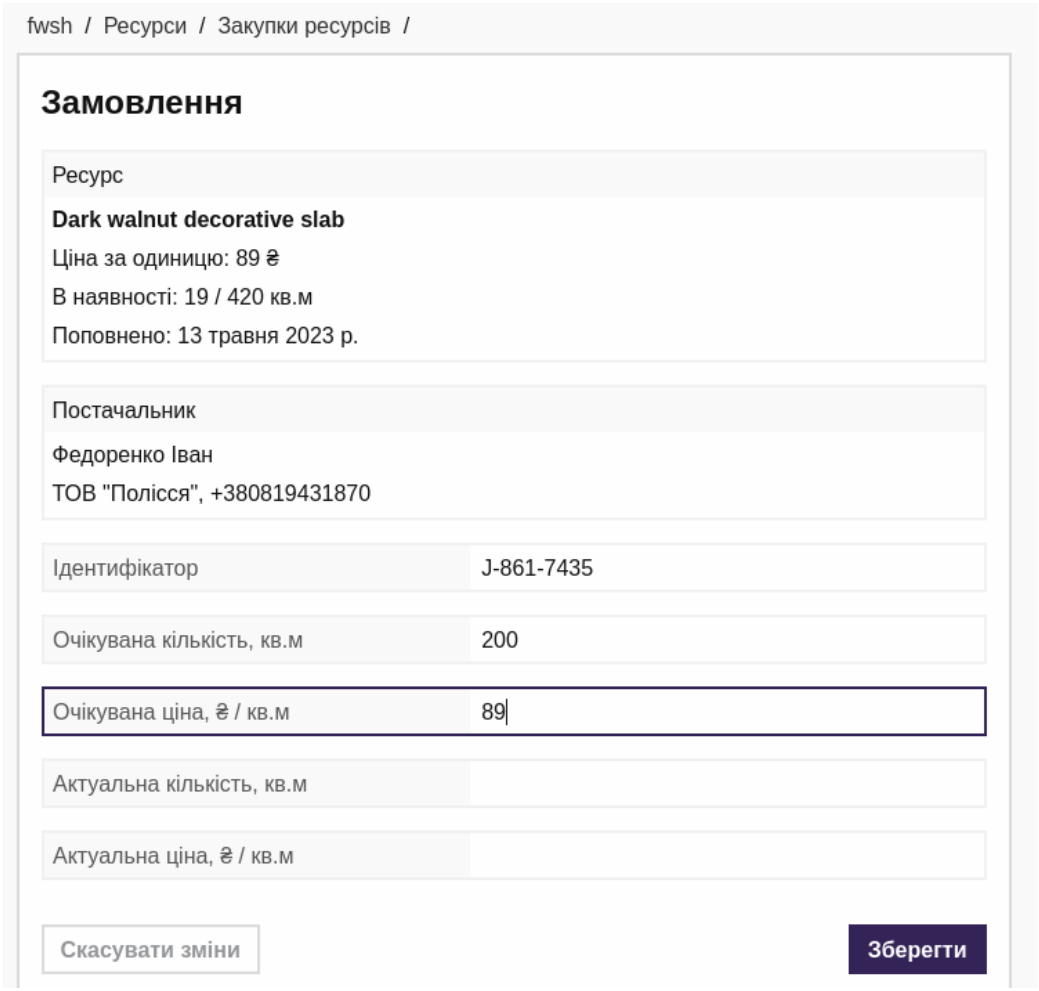

Рисунок Е.20 – Редагування замовлення на поставку ресурсів

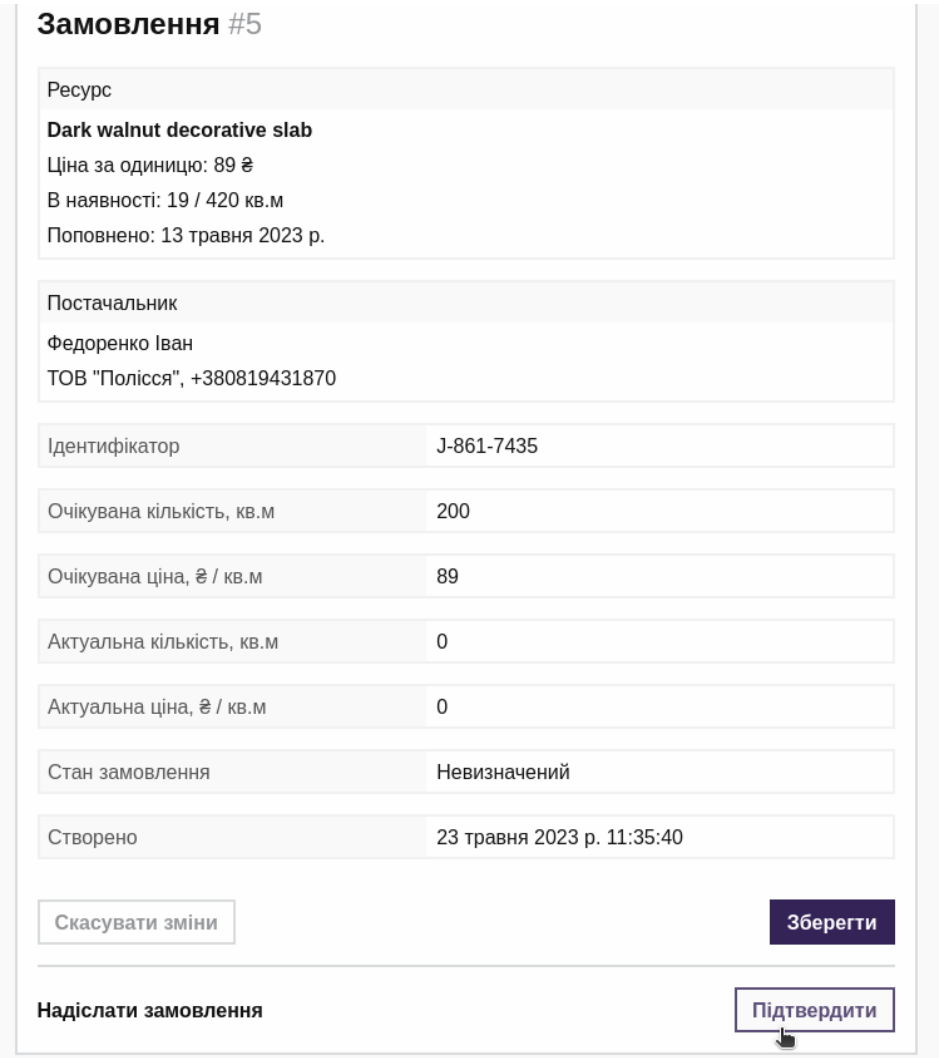

Рисунок Е.21 – Створене замовлення на поставку ресурсів

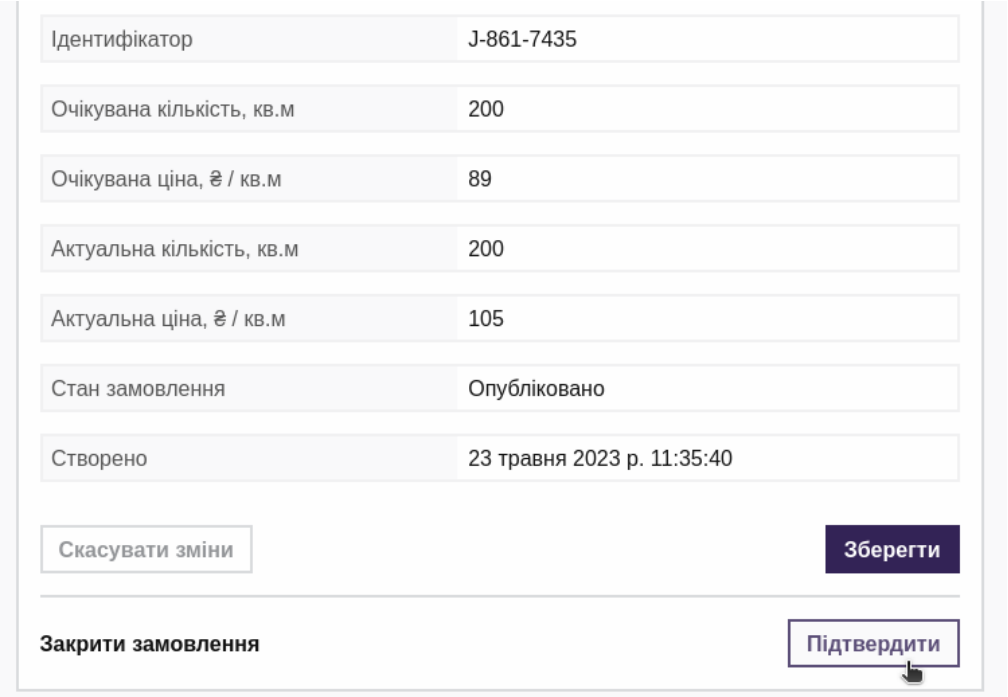

Рисунок Е.22 – Закриття отриманого замовлення

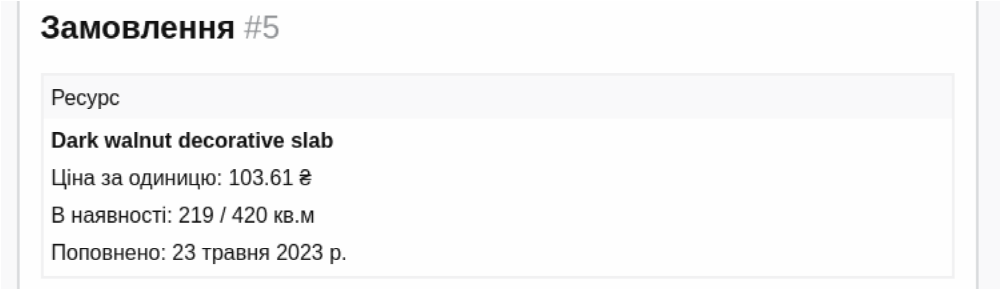

Рисунок Е.23 – Стан ресурсу після перерахування

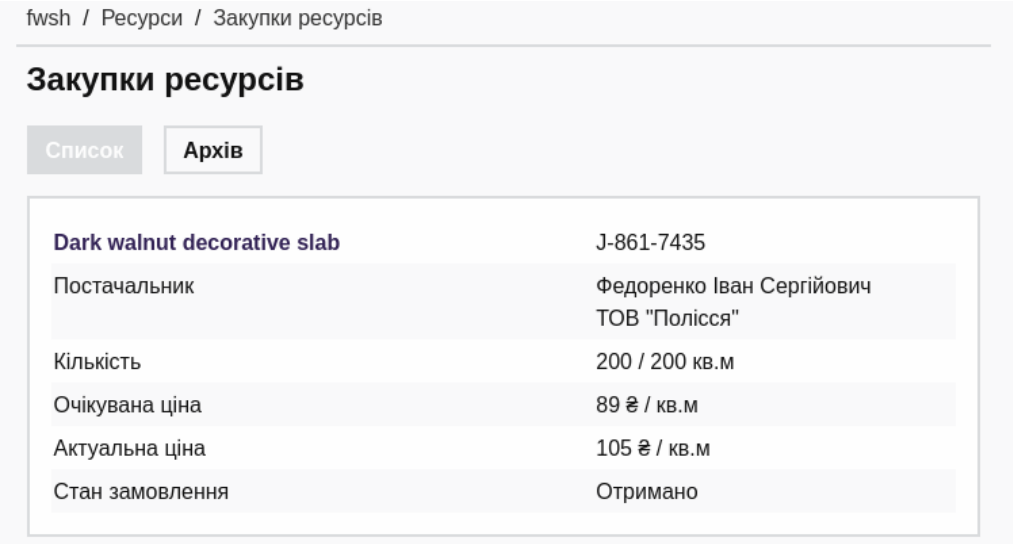

Рисунок Е.24 – Перегляд архіву замовлень

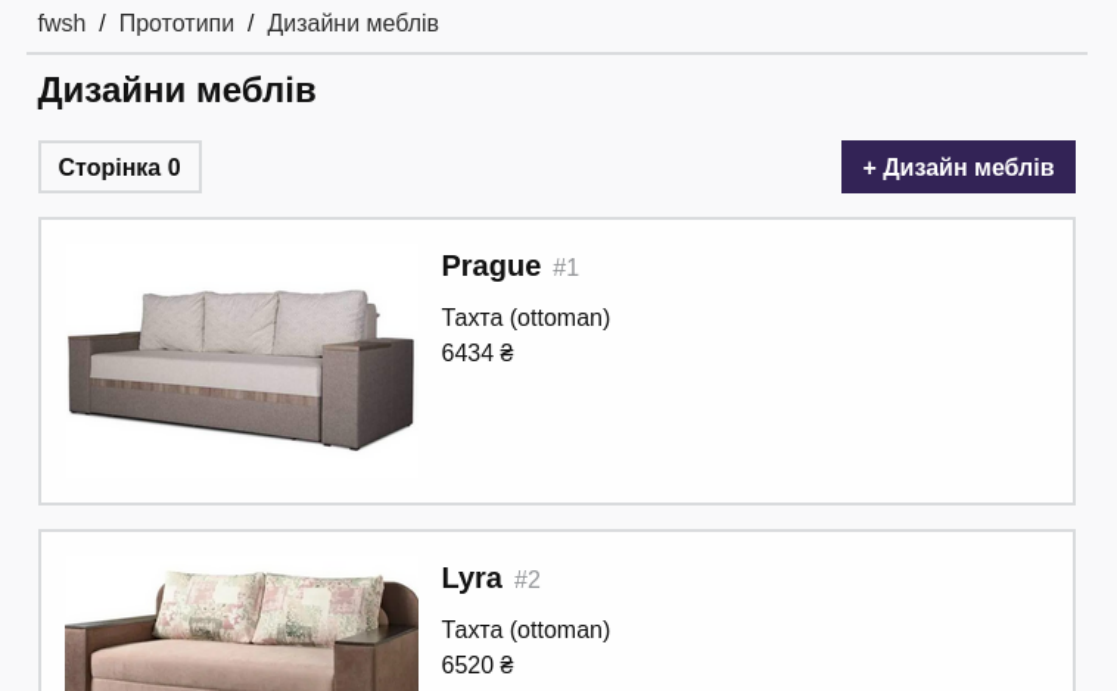

Рисунок Е.25 – Перегляд каталогу дизайнів

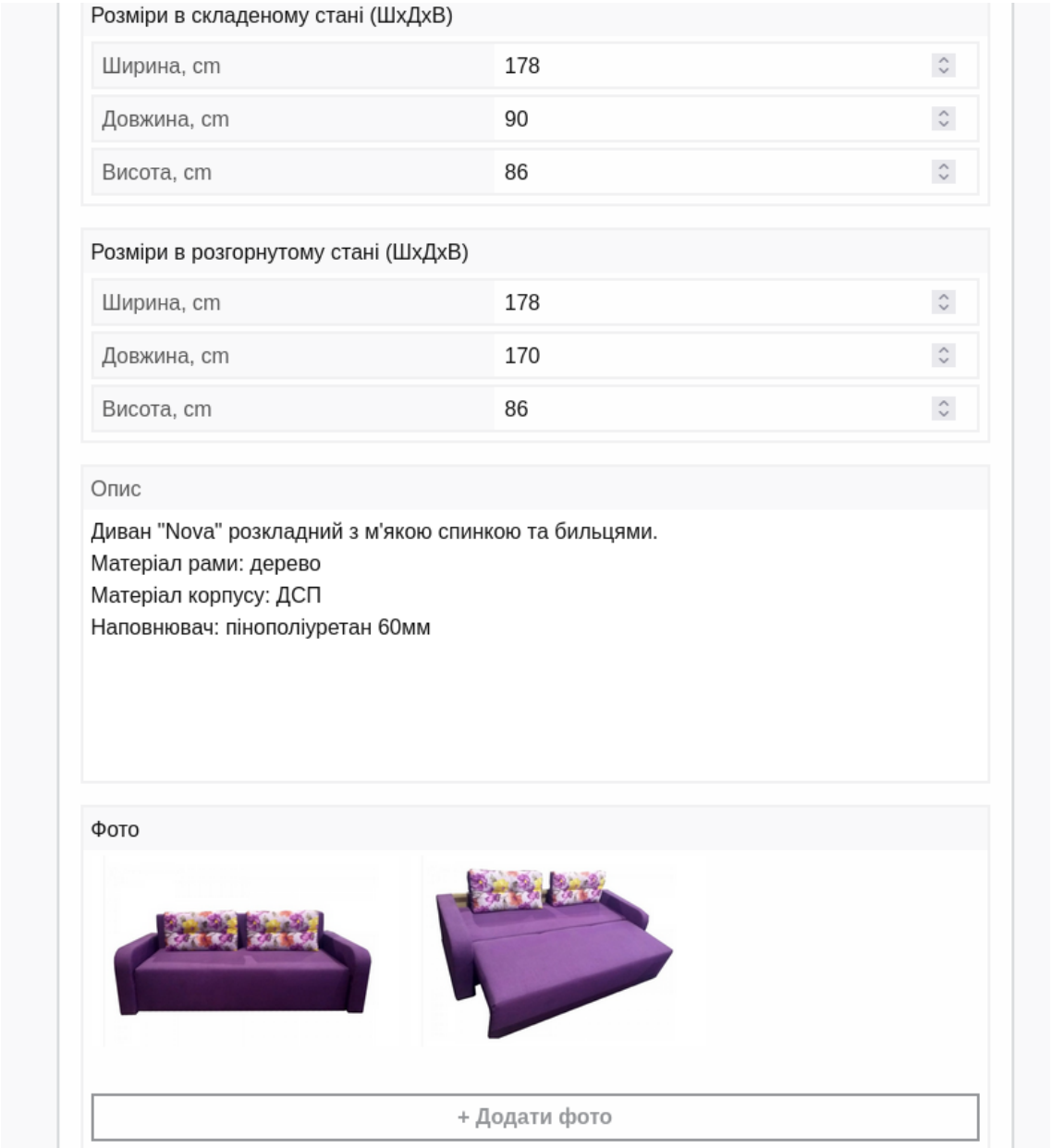

Рисунок Е.26 – Редагування дизайну

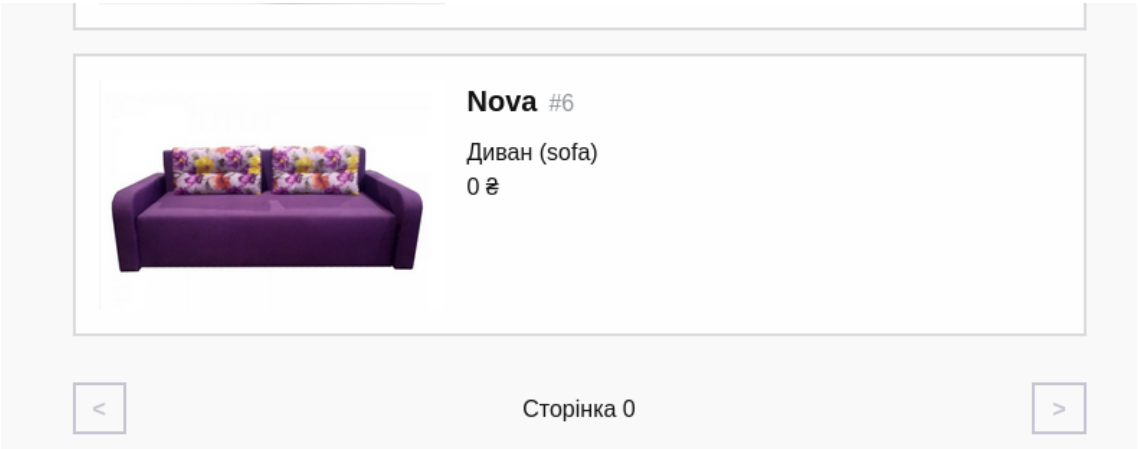

Рисунок Е.27 – Перегляд створеного дизайну в каталозі

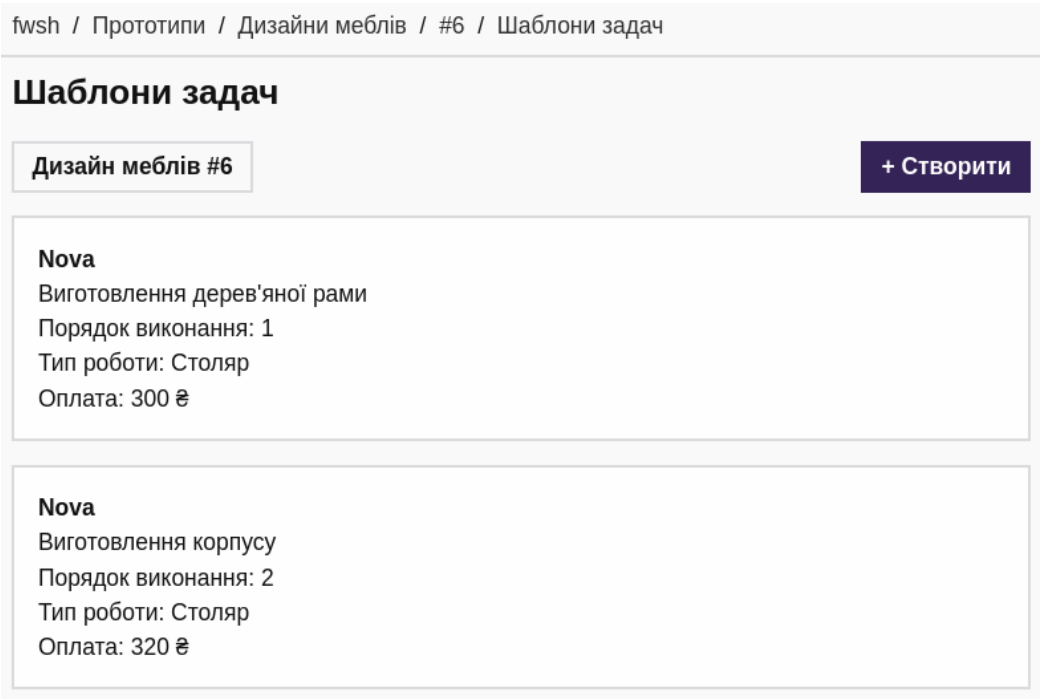

Рисунок Е.28 – Список шаблонів задач

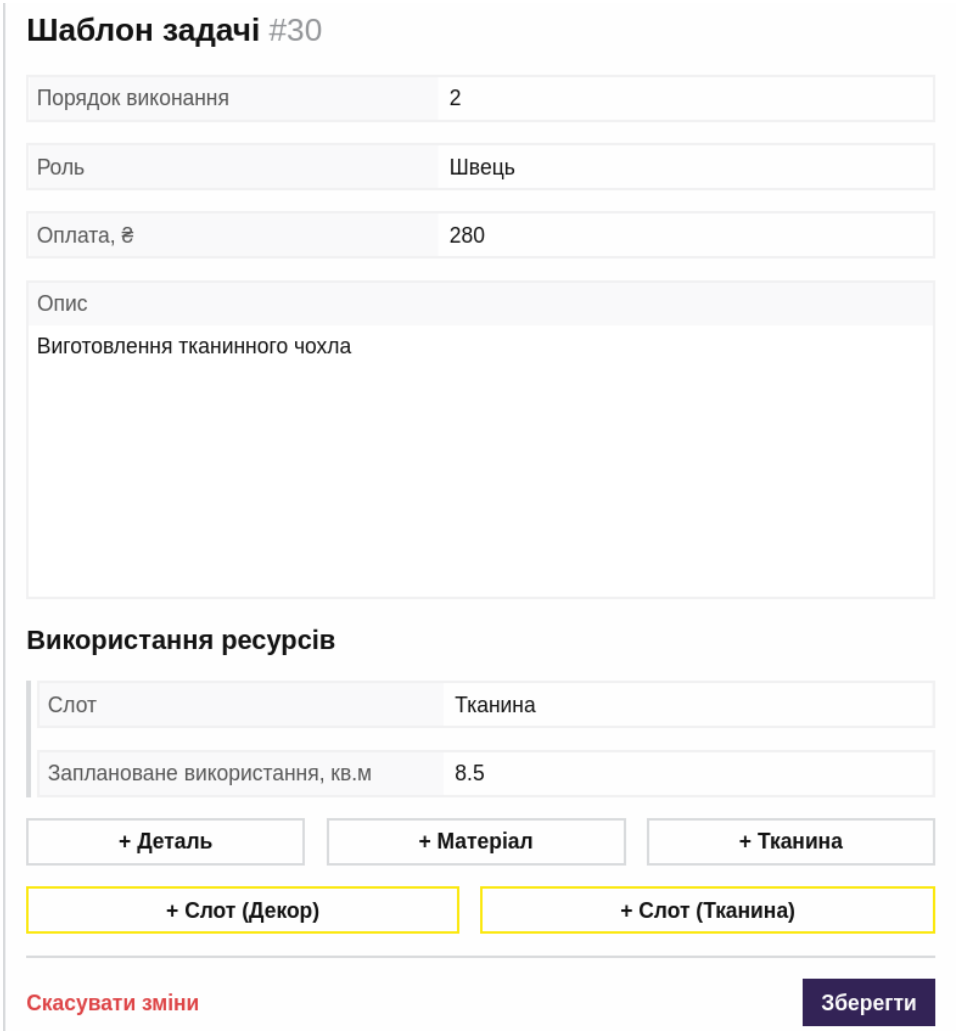

Рисунок Е.29 – Редагування шаблону задачі

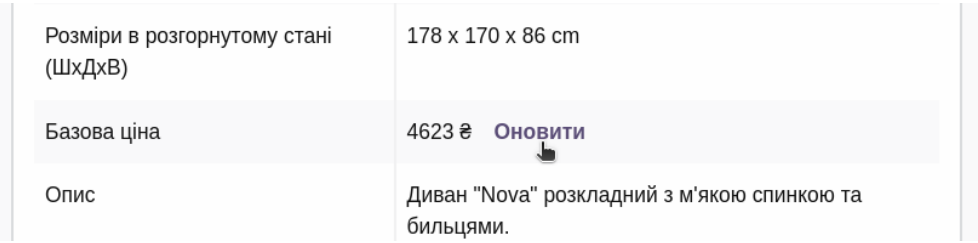

Рисунок Е.30 – Оновлення базової ціни дизайну

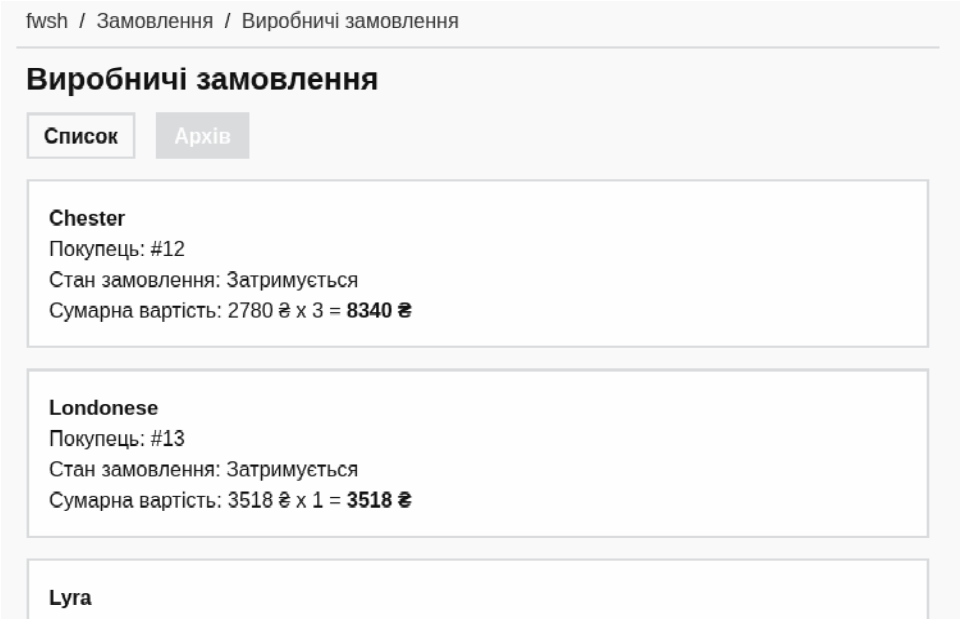

Рисунок Е.31 – Перегляд списку замовлень

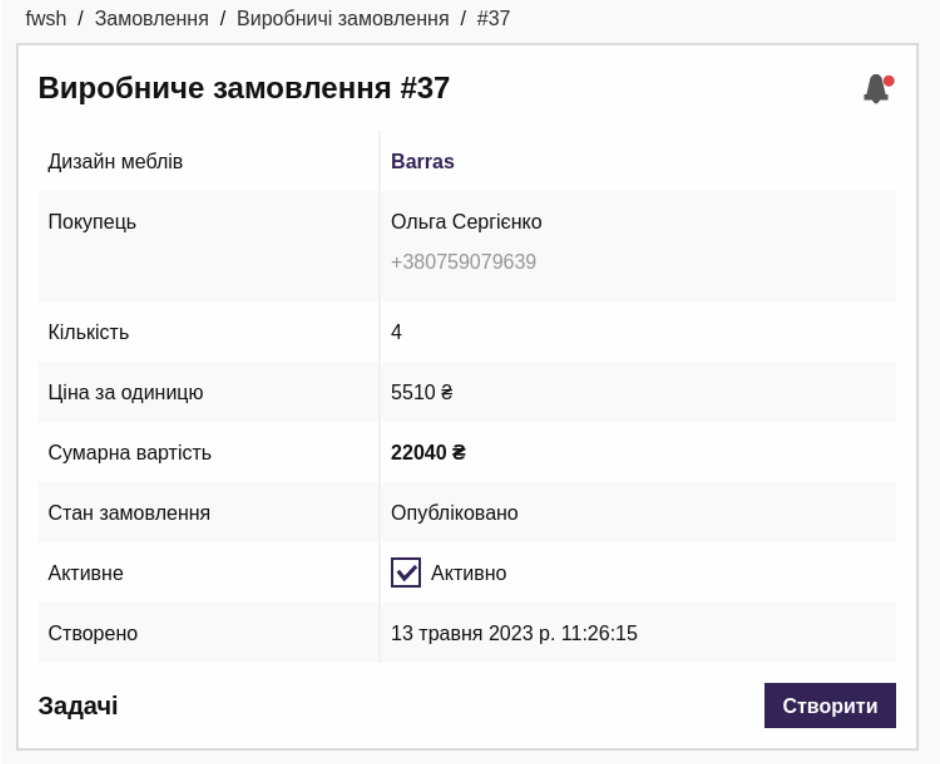

Рисунок Е.32 – Детальний опис виробничого замовлення

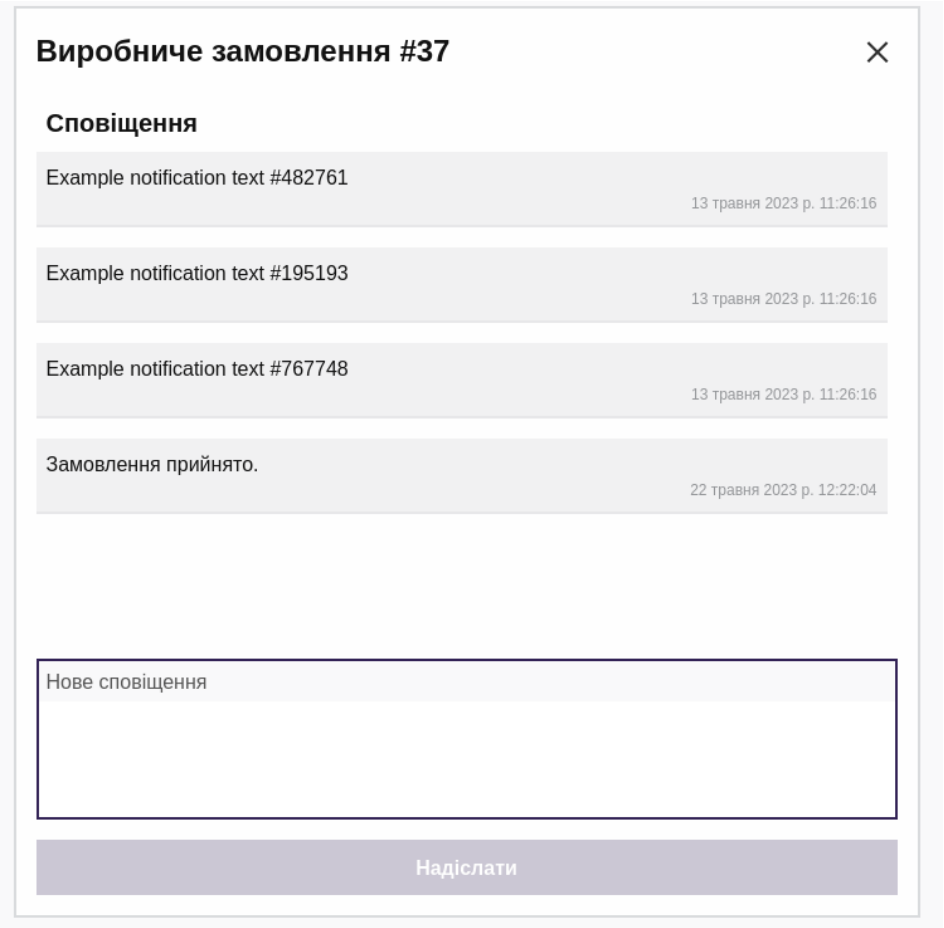

Рисунок Е.33 – Перегляд та додавання сповіщень

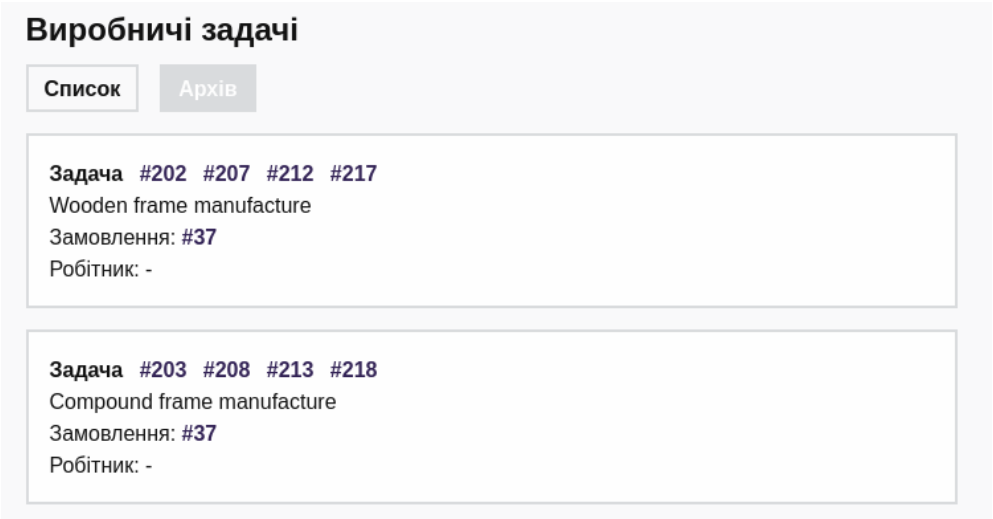

Рисунок Е.34 – Перегляд списку виробничих задач

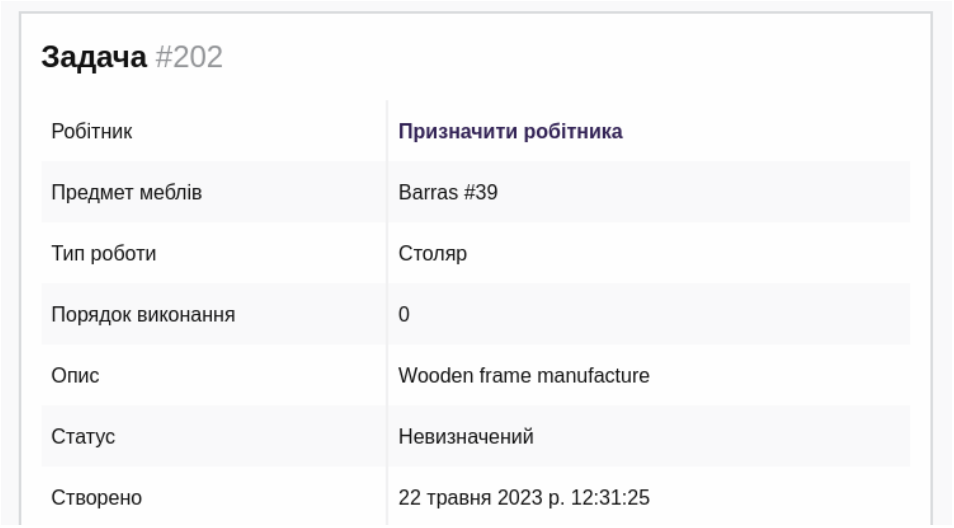

Рисунок Е.35 – Опис виробничої задачі

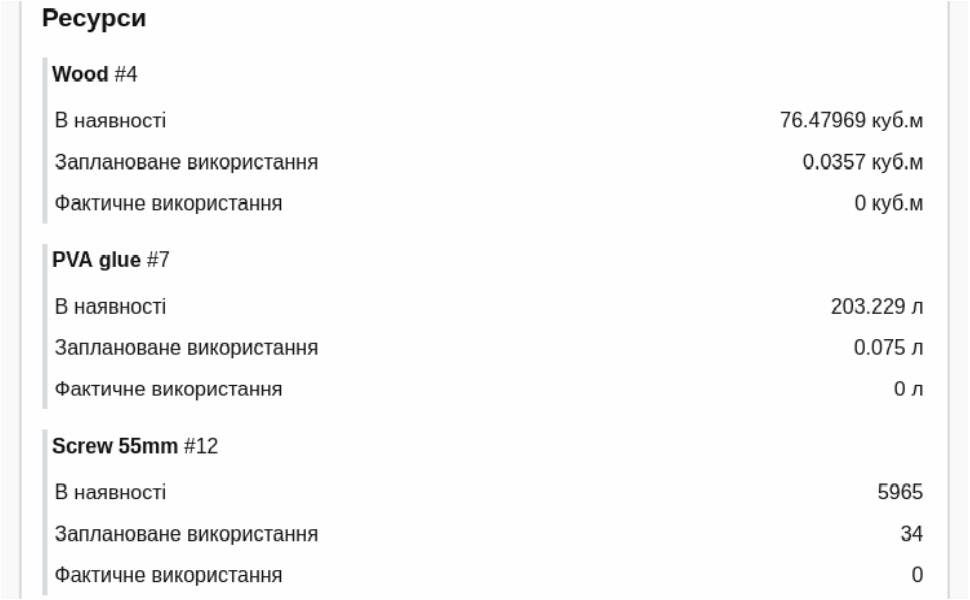

Рисунок Е.36 – Використання ресурсів

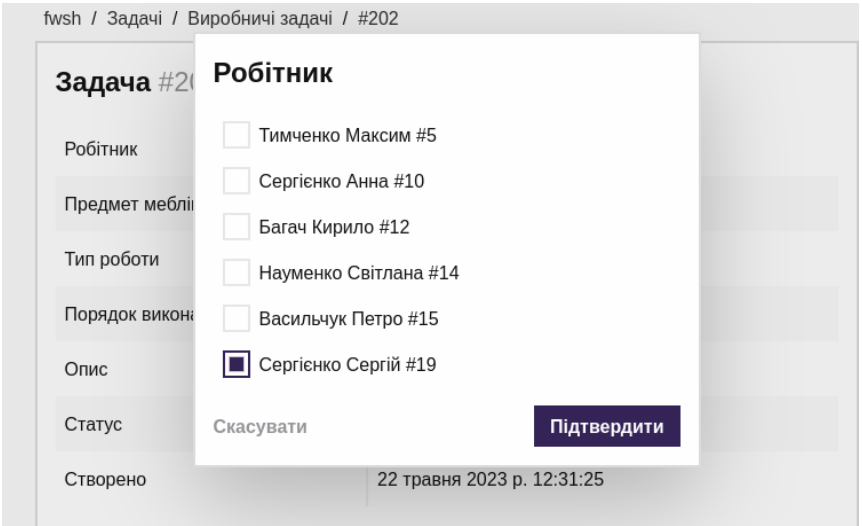

Рисунок Е.37 – Призначення робітника

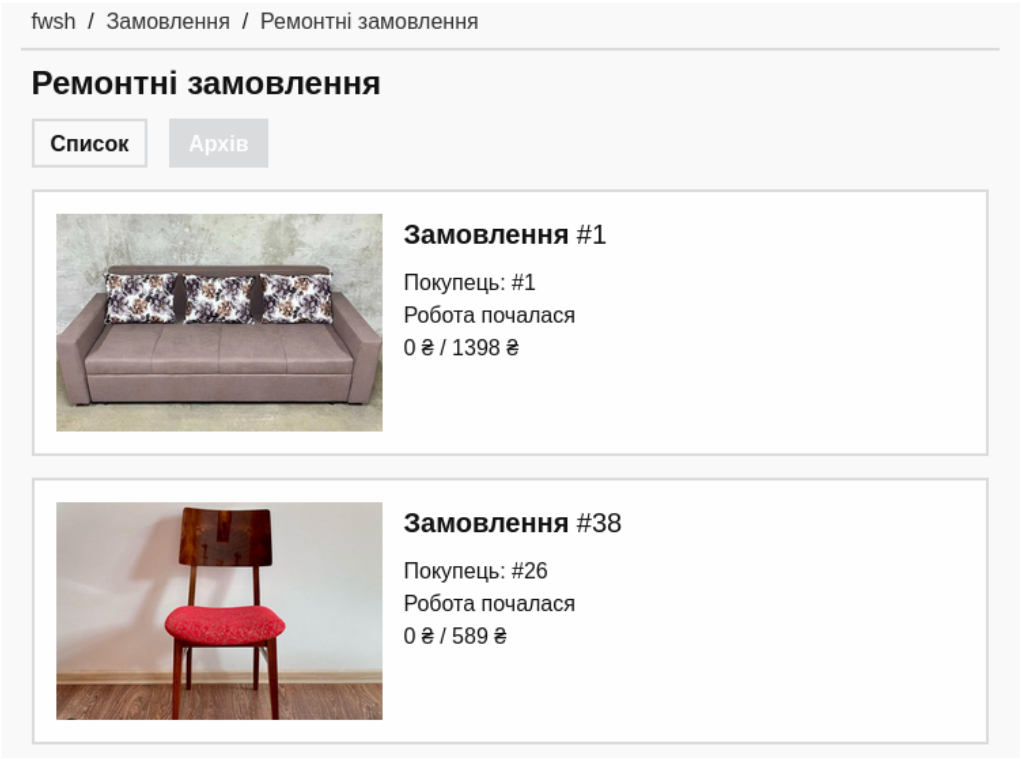

Рисунок Е.38 – Перегляд списку ремонтних замовлень

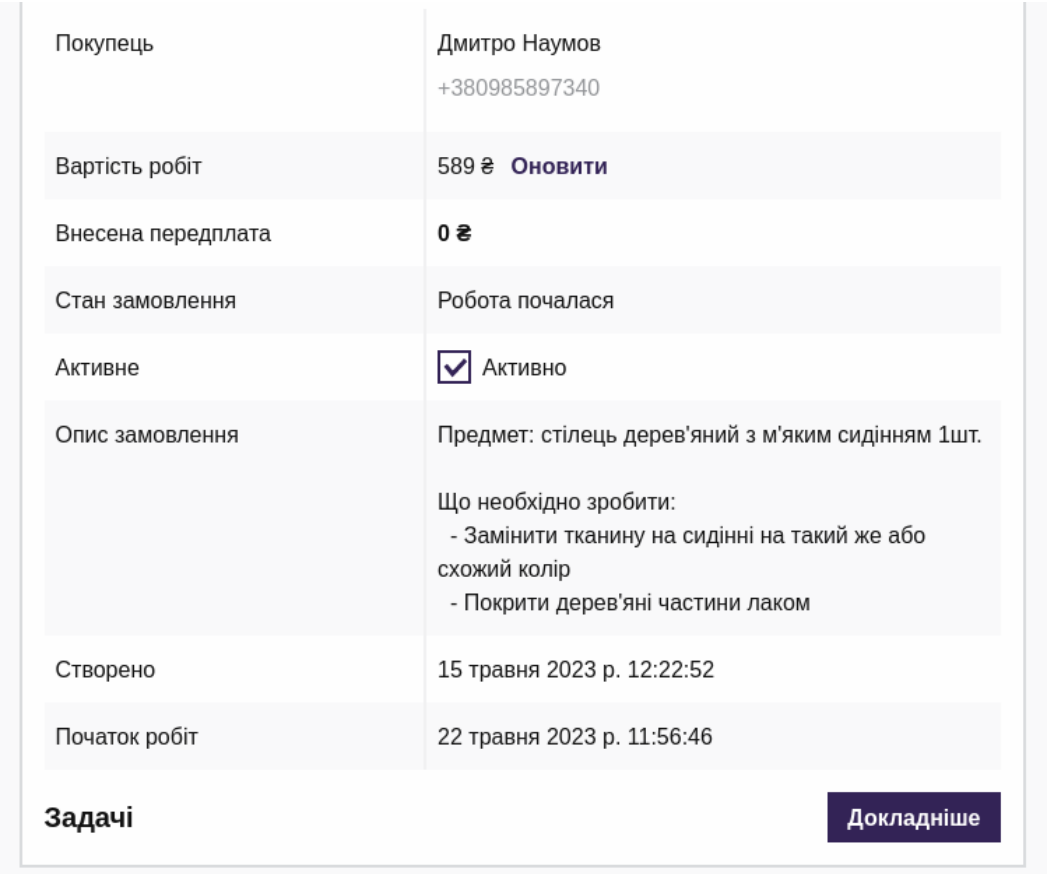

Рисунок Е.39 – Перегляд детального опису замовлення

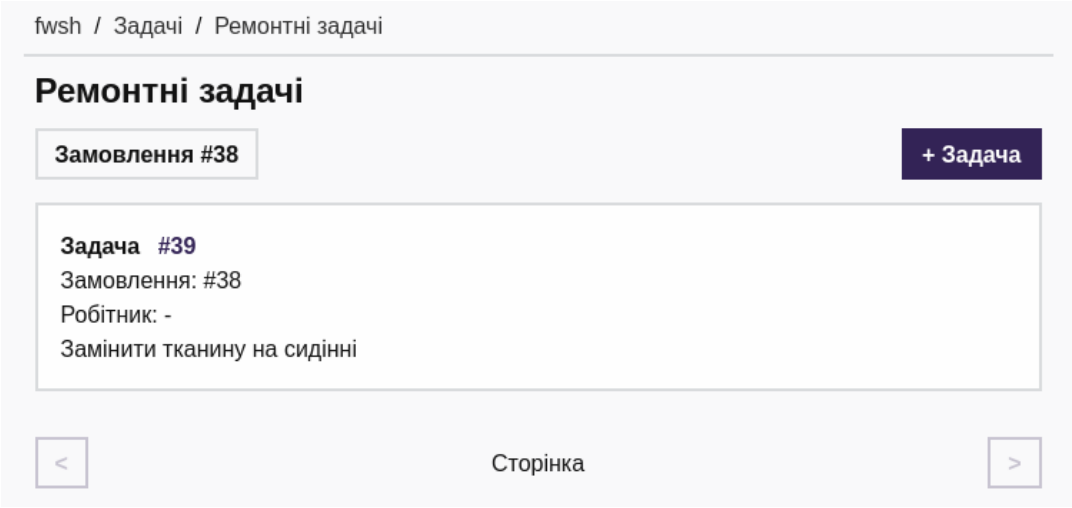

Рисунок Е.40 – Перегляд списку ремонтних задач

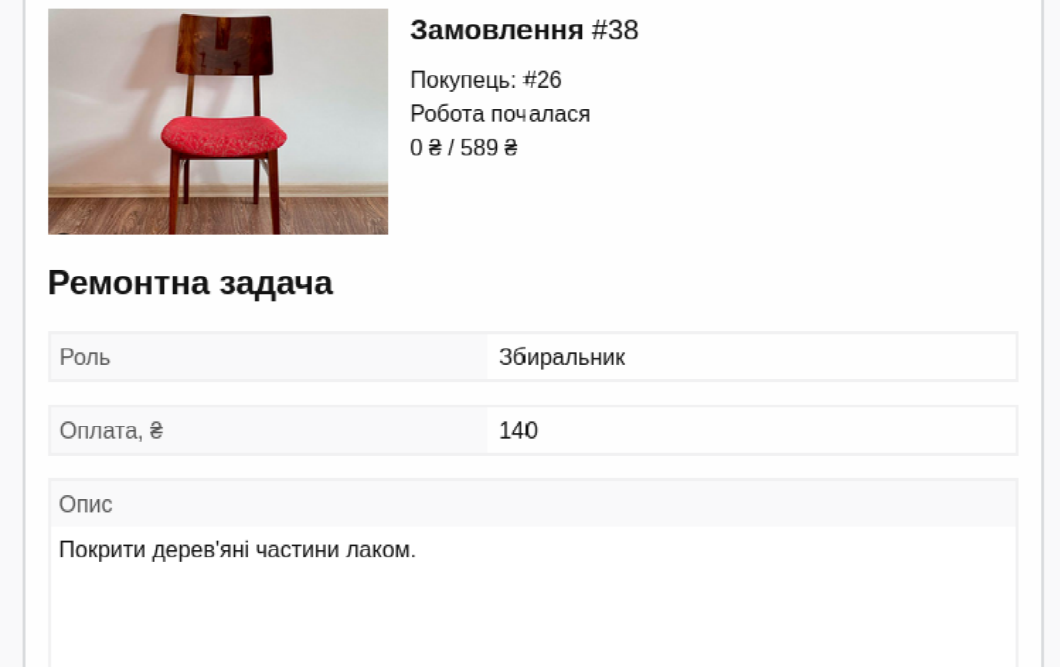

Рисунок Е.41 – Редагування ремонтної задачі

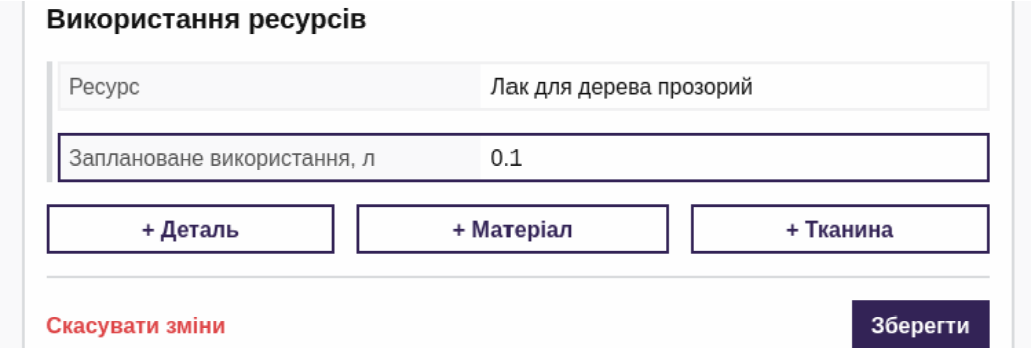

Рисунок Е.42 – Додавання ресурсів до задачі

Рисунок Е.43 – Позначення ресурсу, що унеможливлює виконання робіт

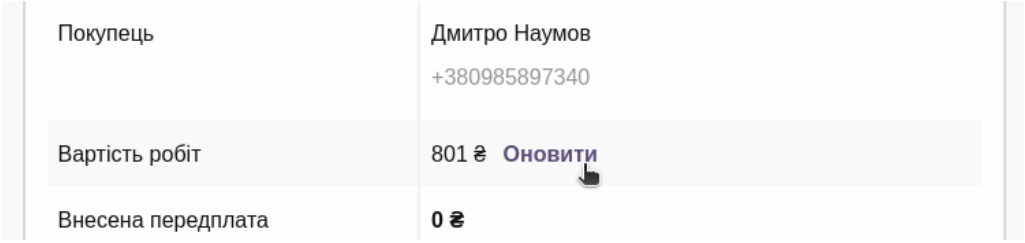

Рисунок Е.44 – Обчислення вартості замовлення

Е.3 Використання панелі робітника

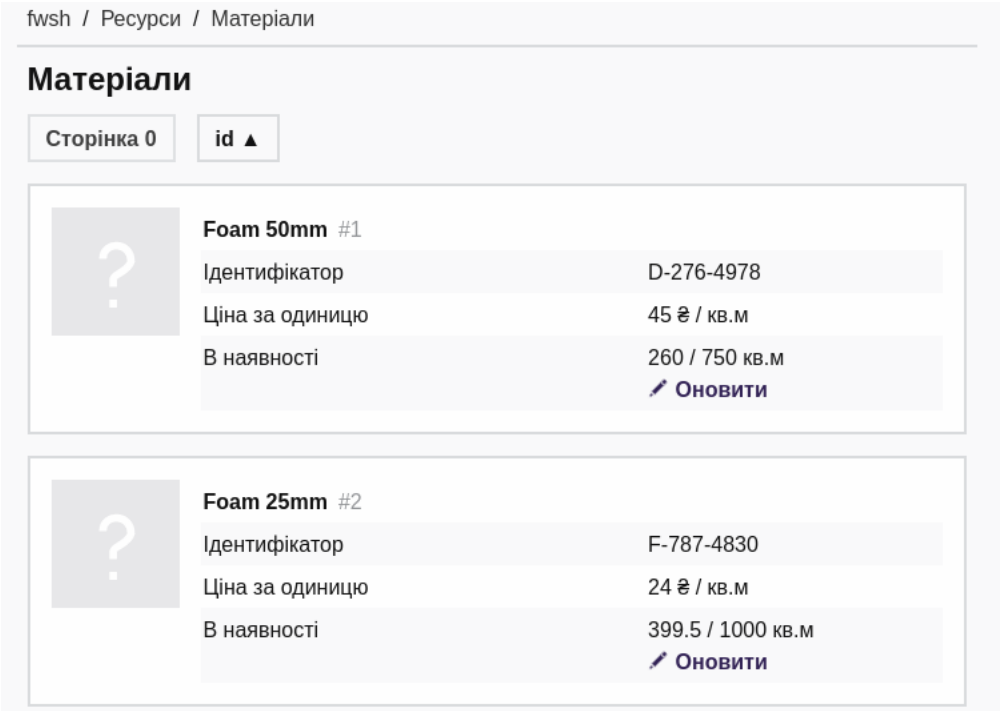

Рисунок Е.45 – Перегляд списку матеріалів

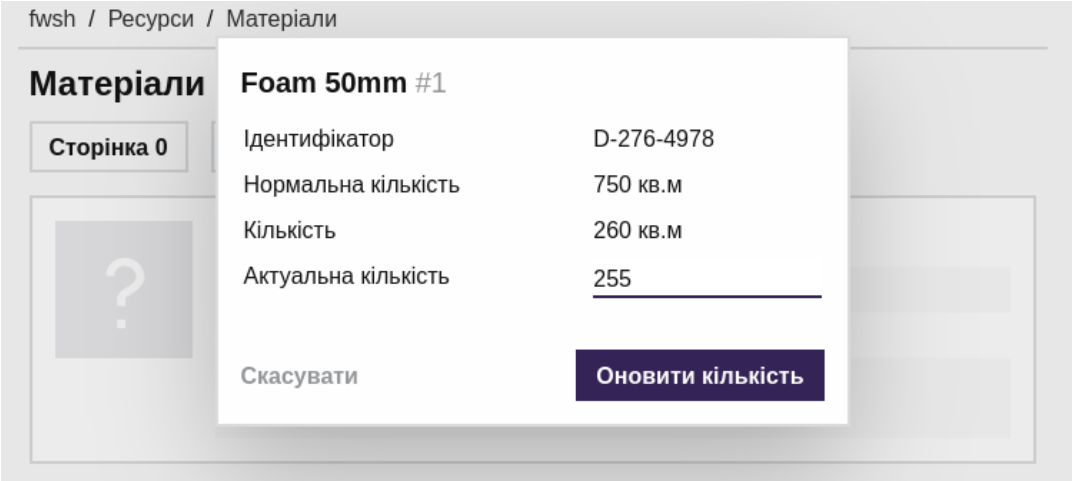

Рисунок Е.46 – Оновлення кількості ресурсу на складі

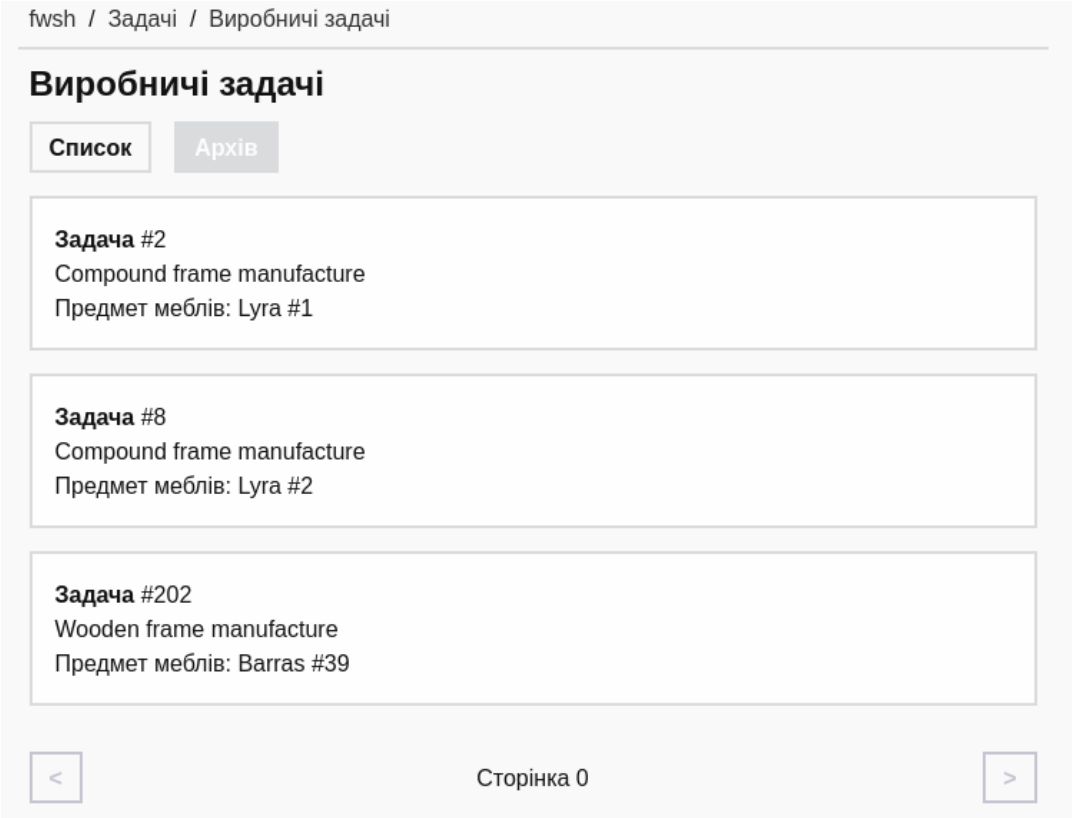

Рисунок Е.47 – Перегляд списку виробничих задач

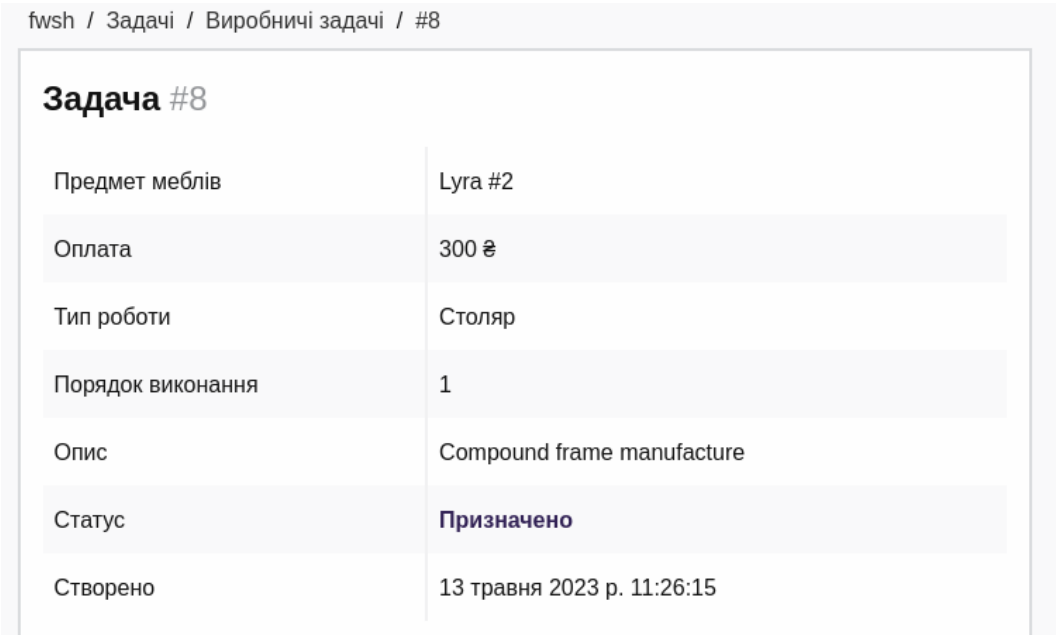

Рисунок Е.48 – Детальний опис задачі

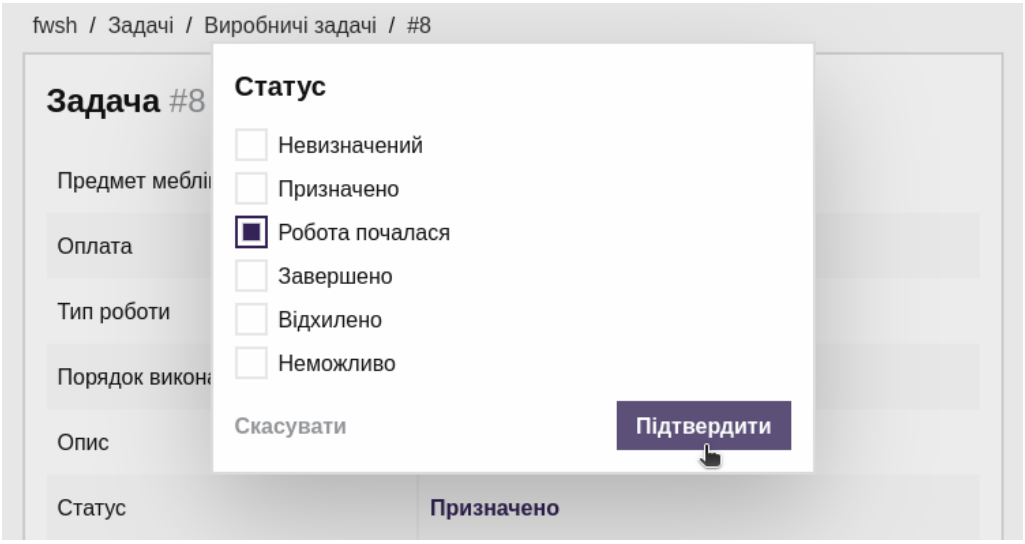

Рисунок Е.49 – Зміна статусу задачі

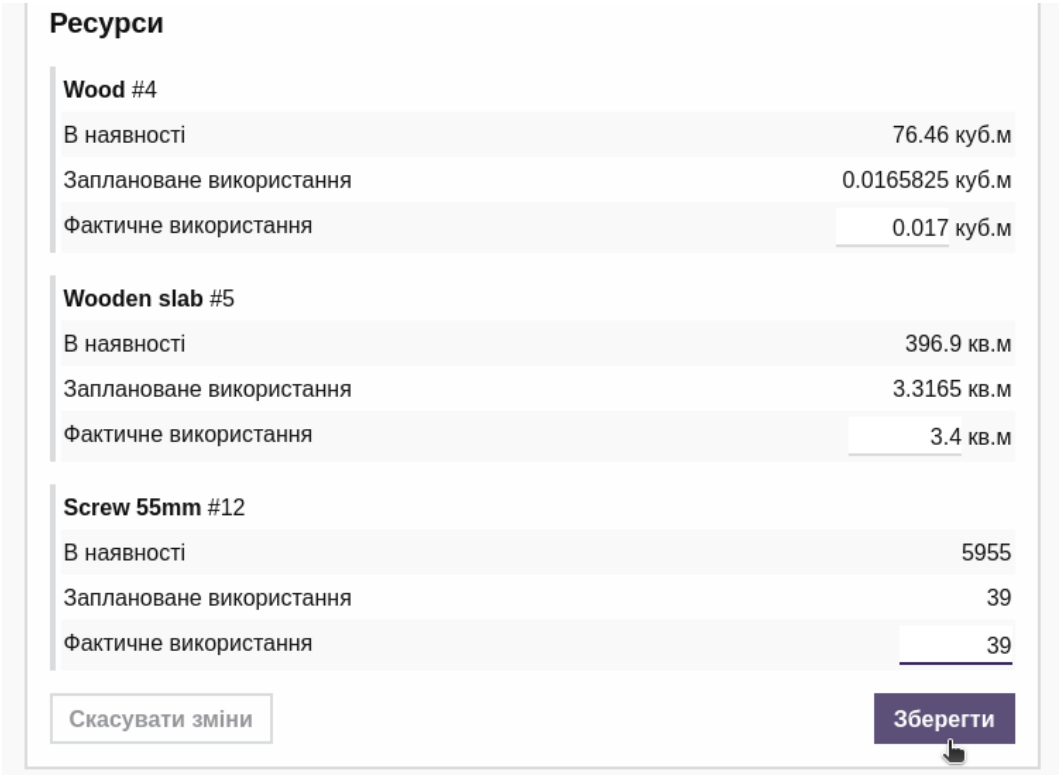

Рисунок Е.50 – Звітування про використані ресурси

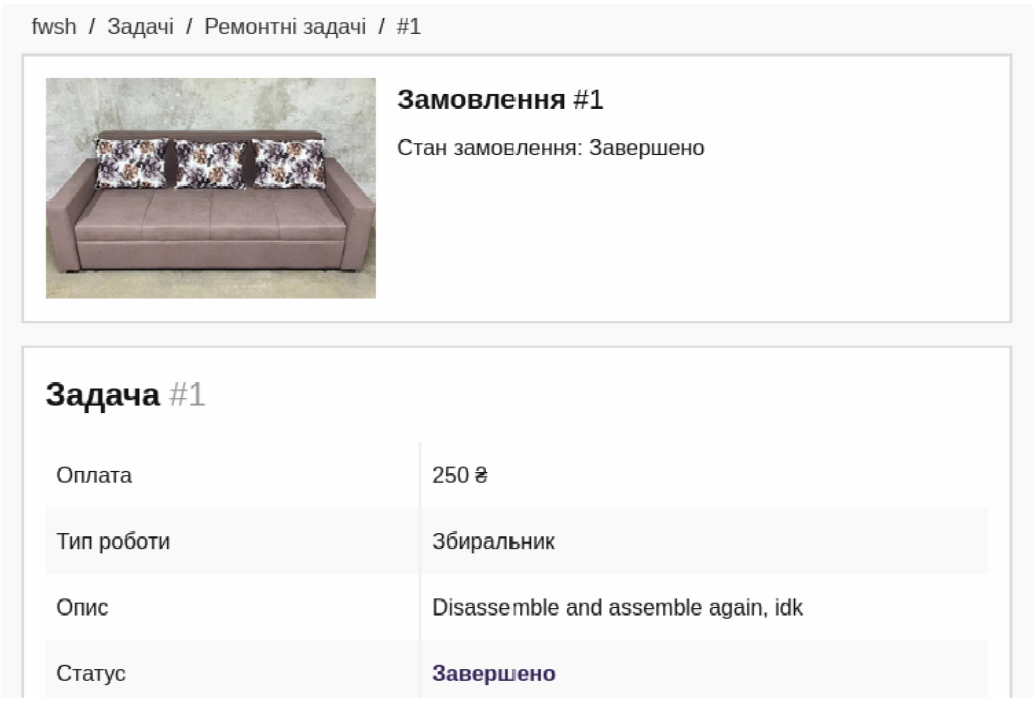

Рисунок Е.51 – Завершене замовлення

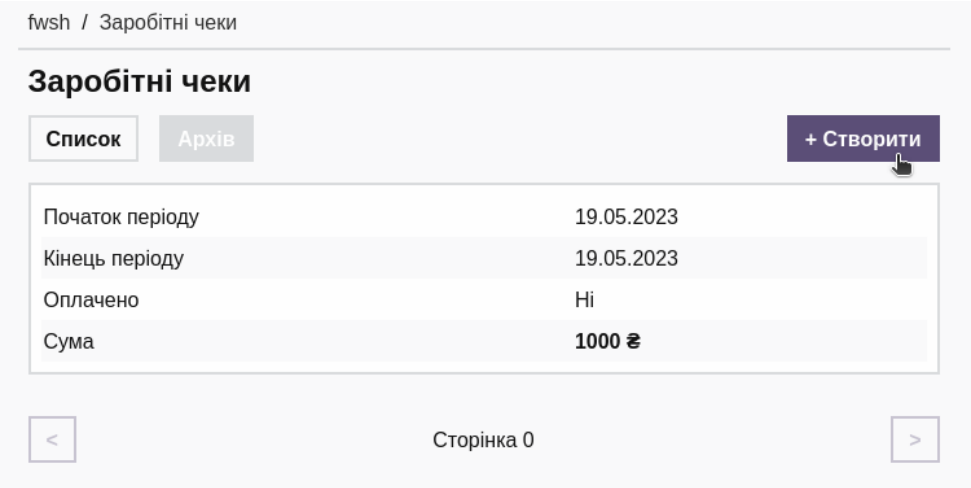

Рисунок Е.52 – Список заробітних чеків

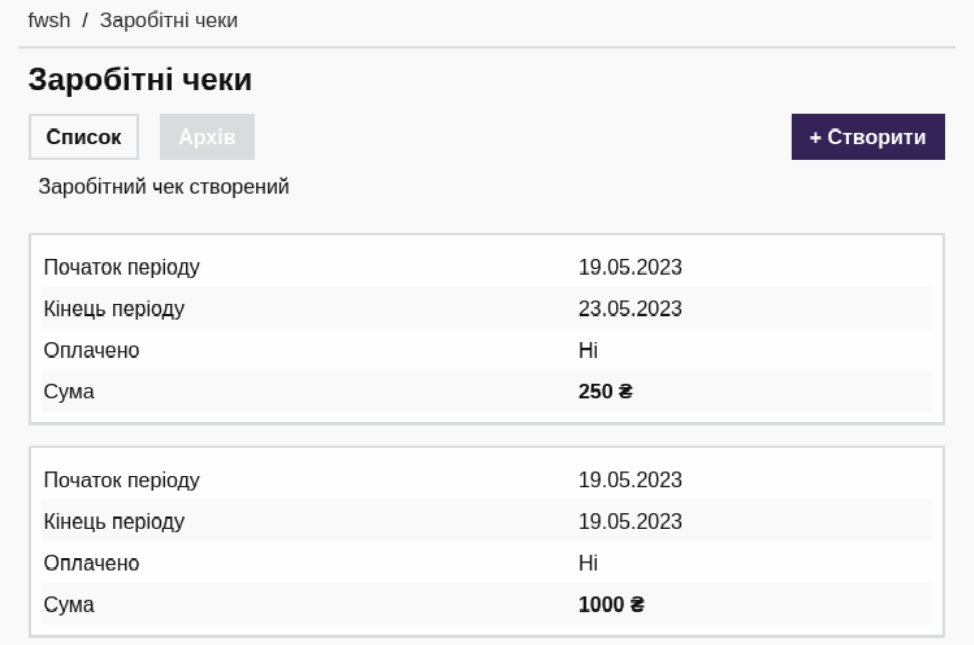

Рисунок Е.53 – Перегляд створеного заробітного чека

Е.4 Використання вебсайту покупця

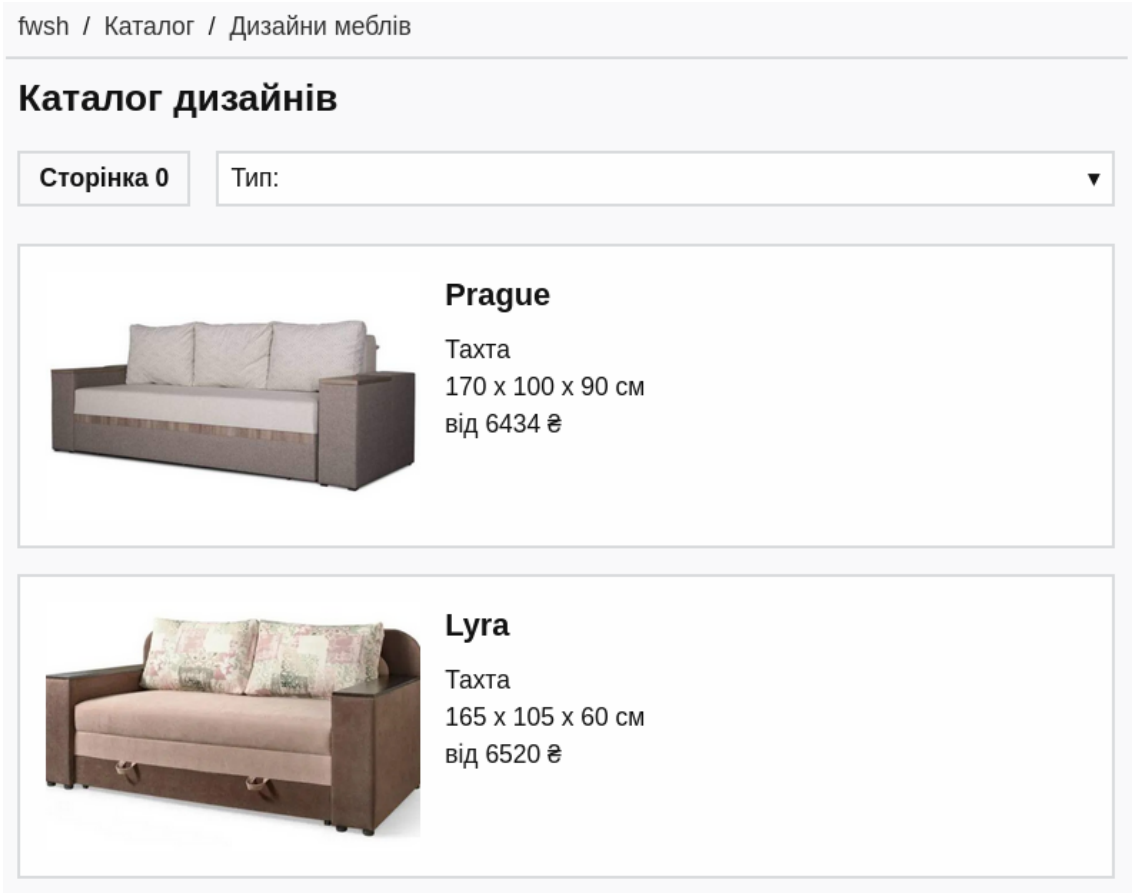

Рисунок Е.54 – Перегляд каталогу меблів

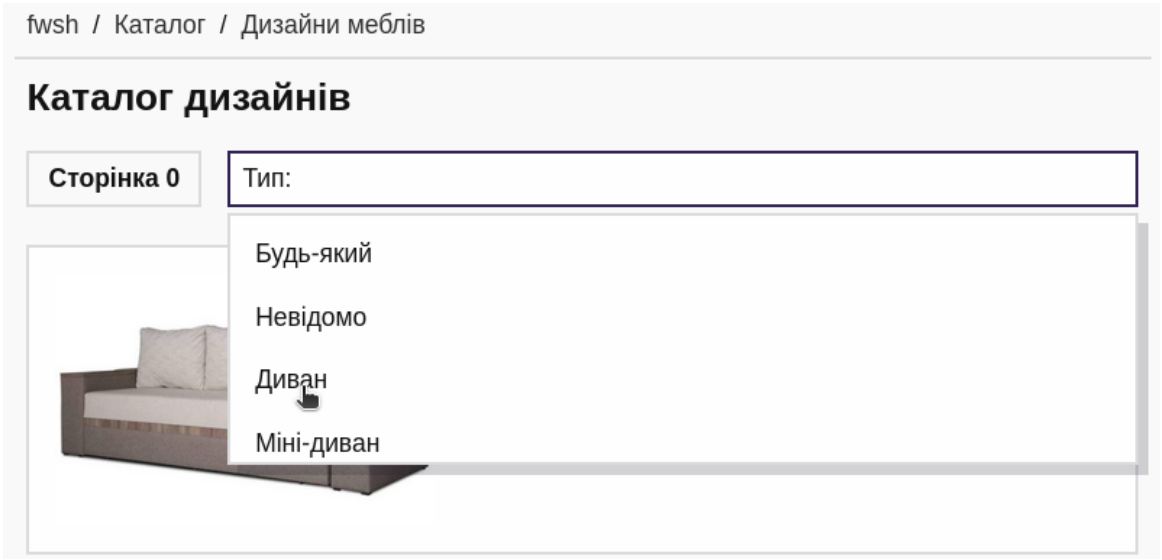

Рисунок Е.55 – Вибір типу меблів

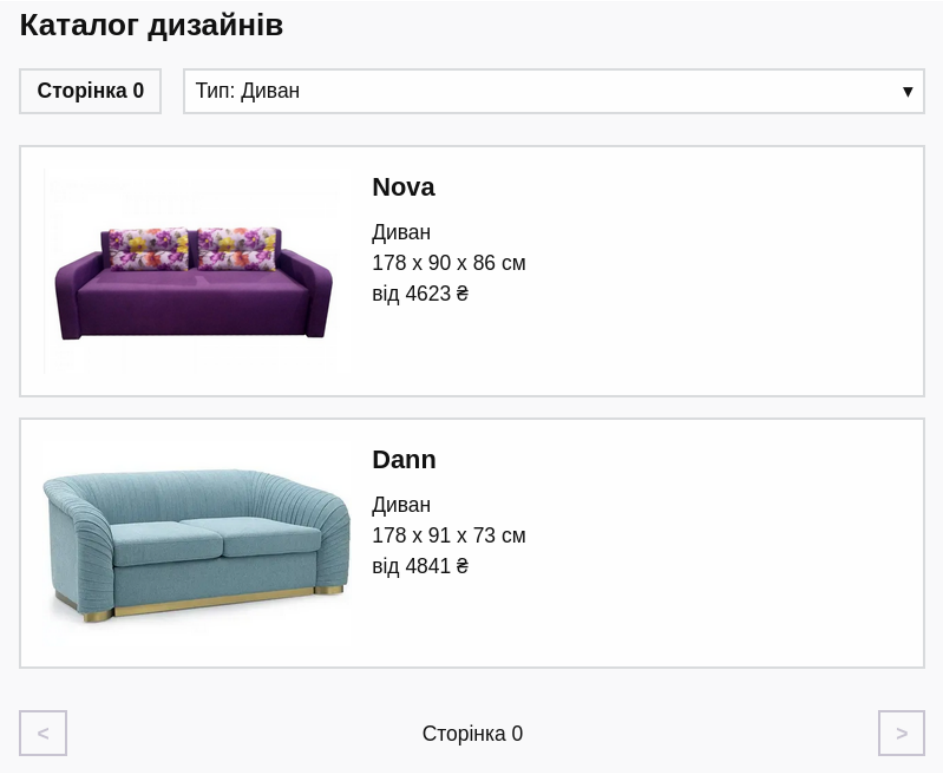

Рисунок Е.56 – Результат фільтрації по типу меблів

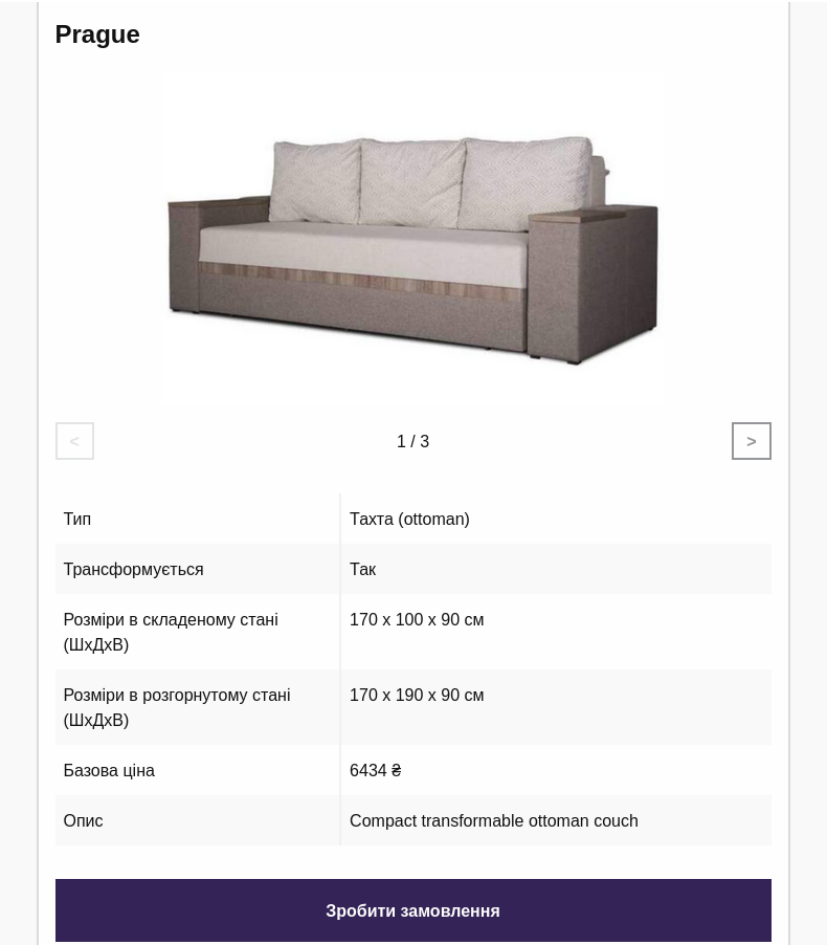

Рисунок Е.57 – Детальний опис предмету меблів

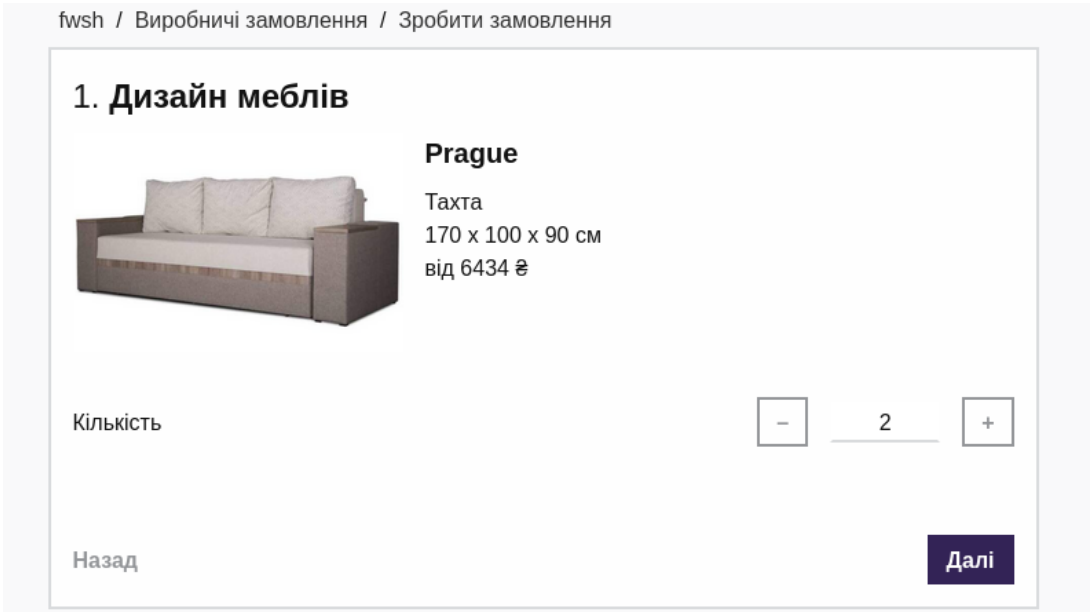

Рисунок Е.58 – Вибір кількості меблів

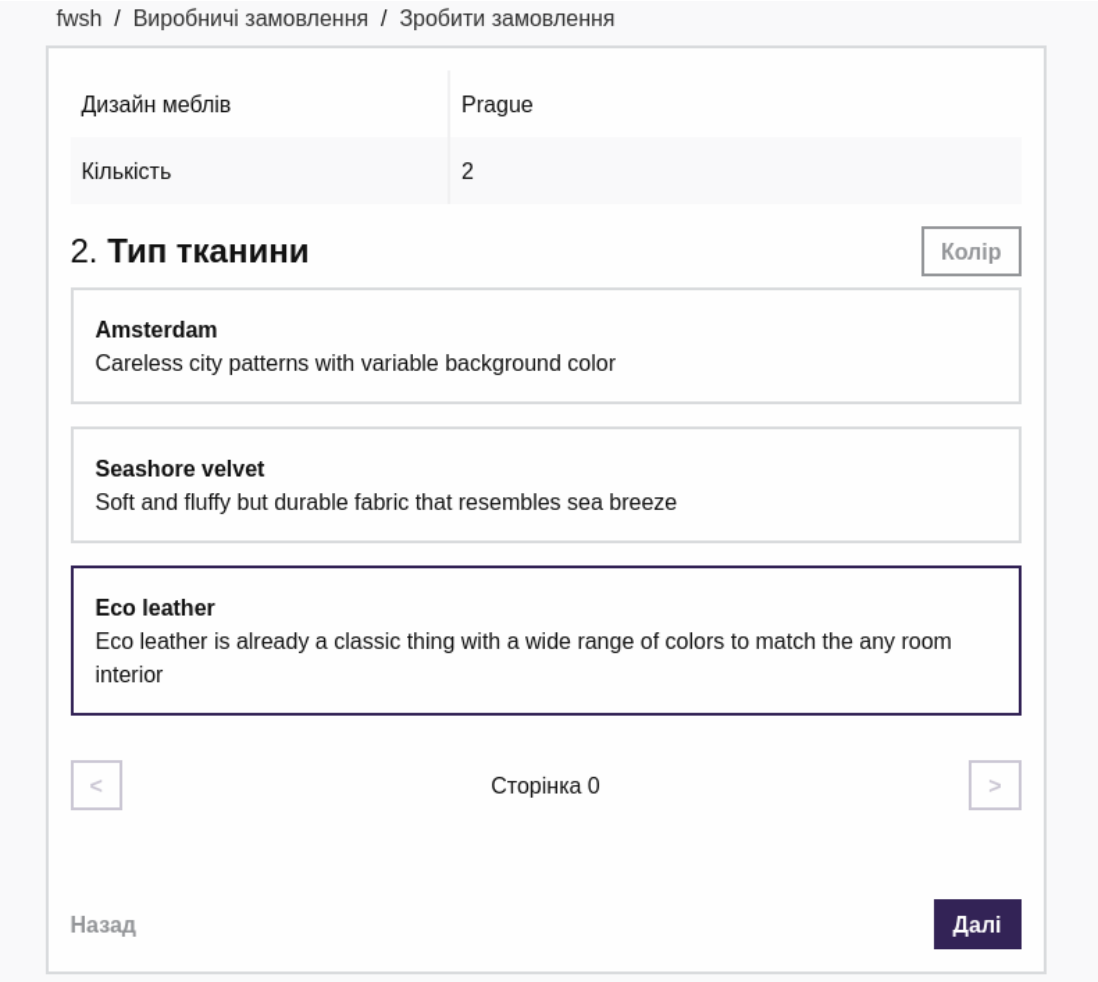

Рисунок Е.59 – Вибір типу тканини

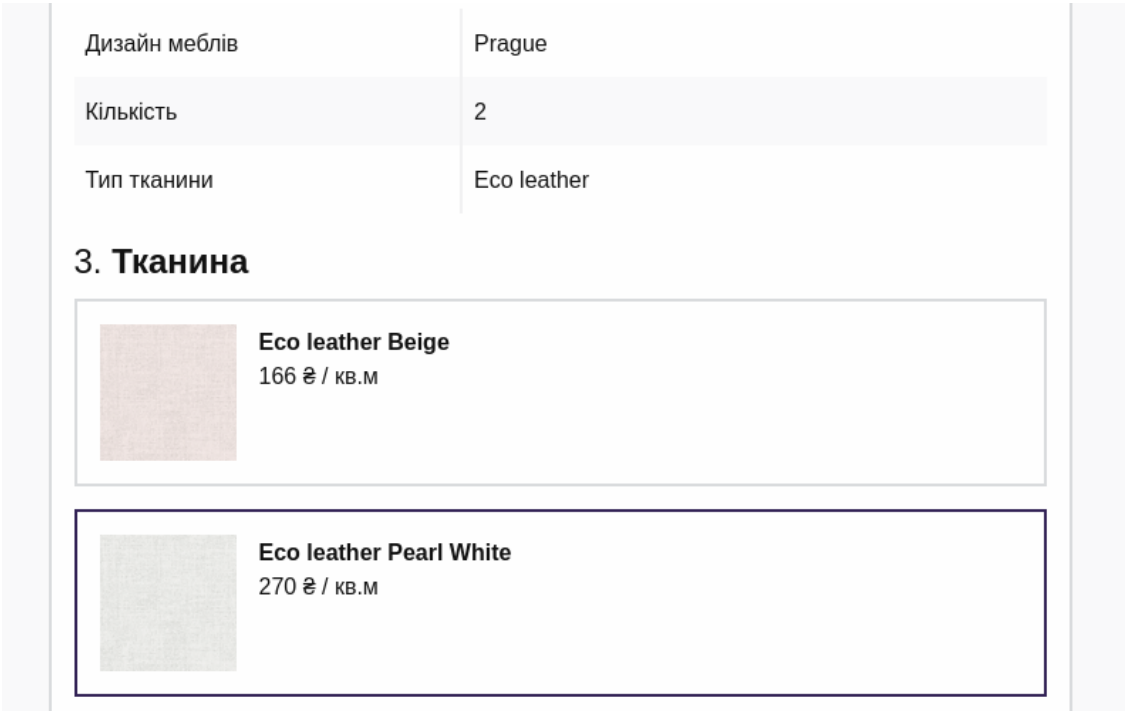

Рисунок Е.60 – Вибір тканини

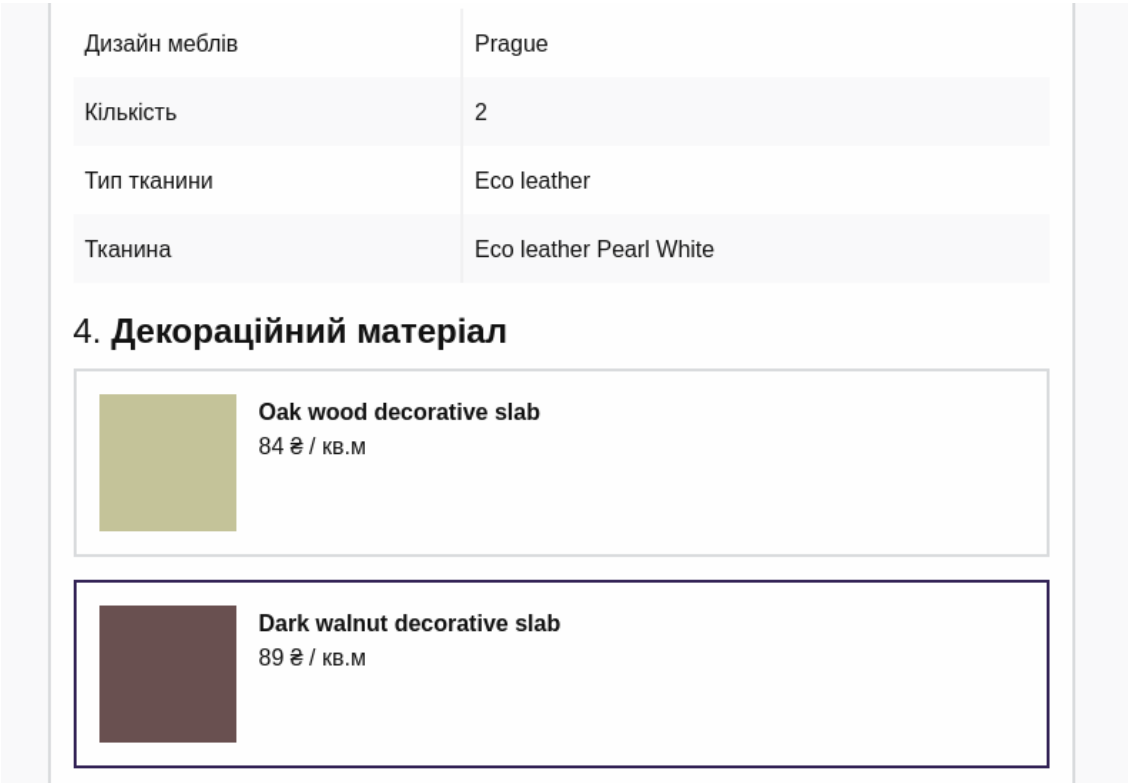

Рисунок Е.61 – Вибір декораційного матеріалу

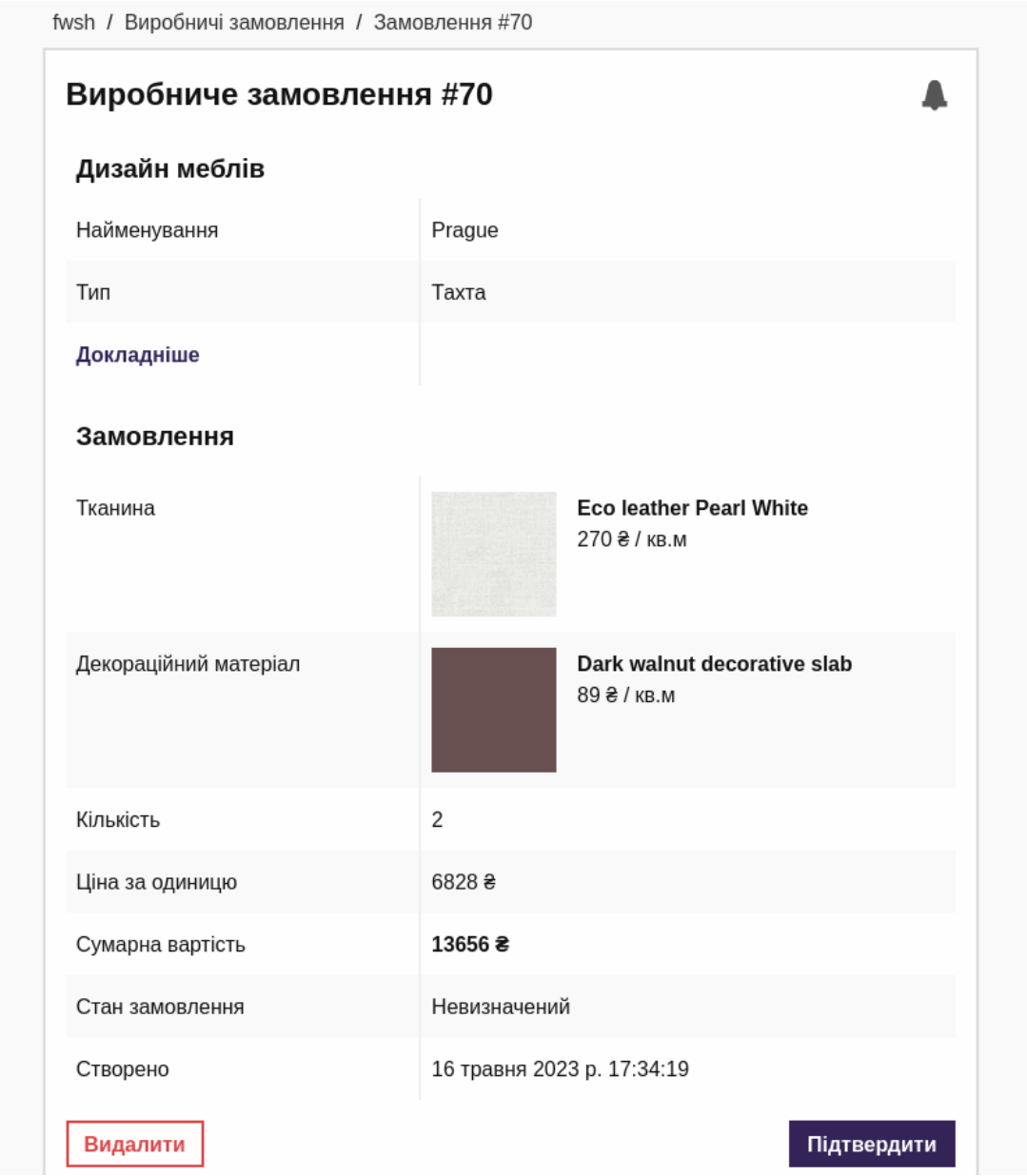

Рисунок Е.62 – Перегляд стану замовлення

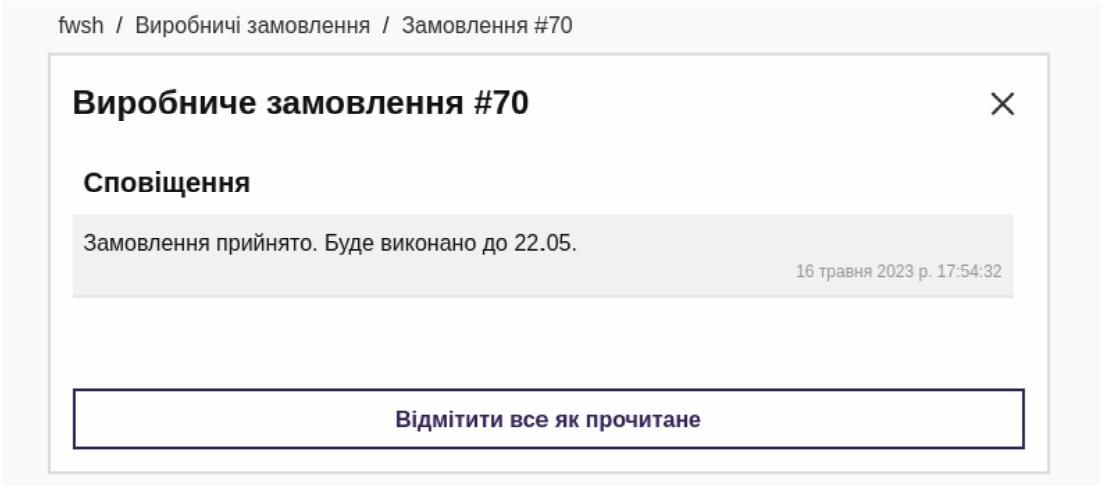

Рисунок Е.63 – Перегляд сповіщень

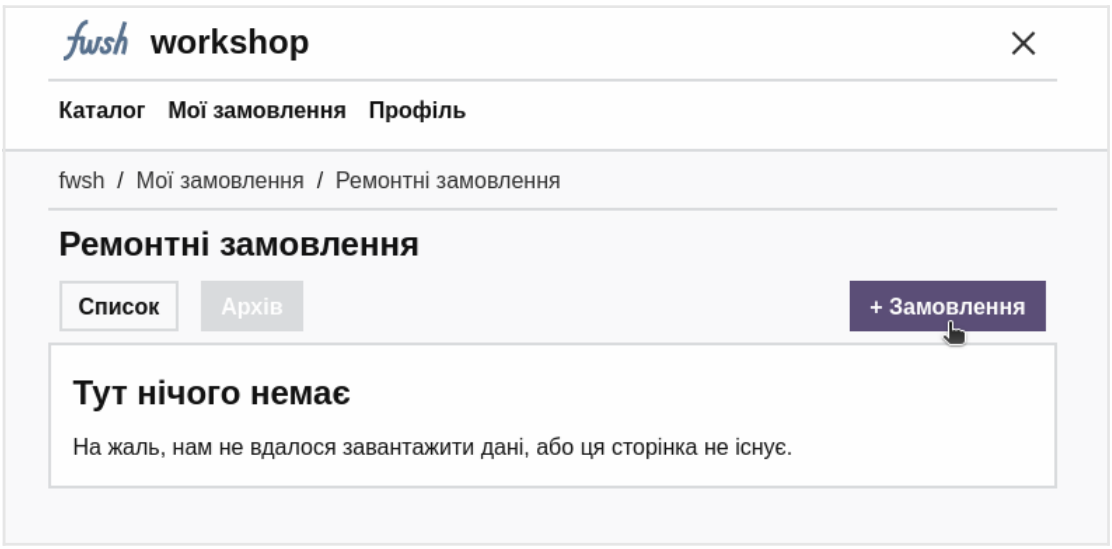

Рисунок Е.64 – Сторінка «Ремонтні замовлення»

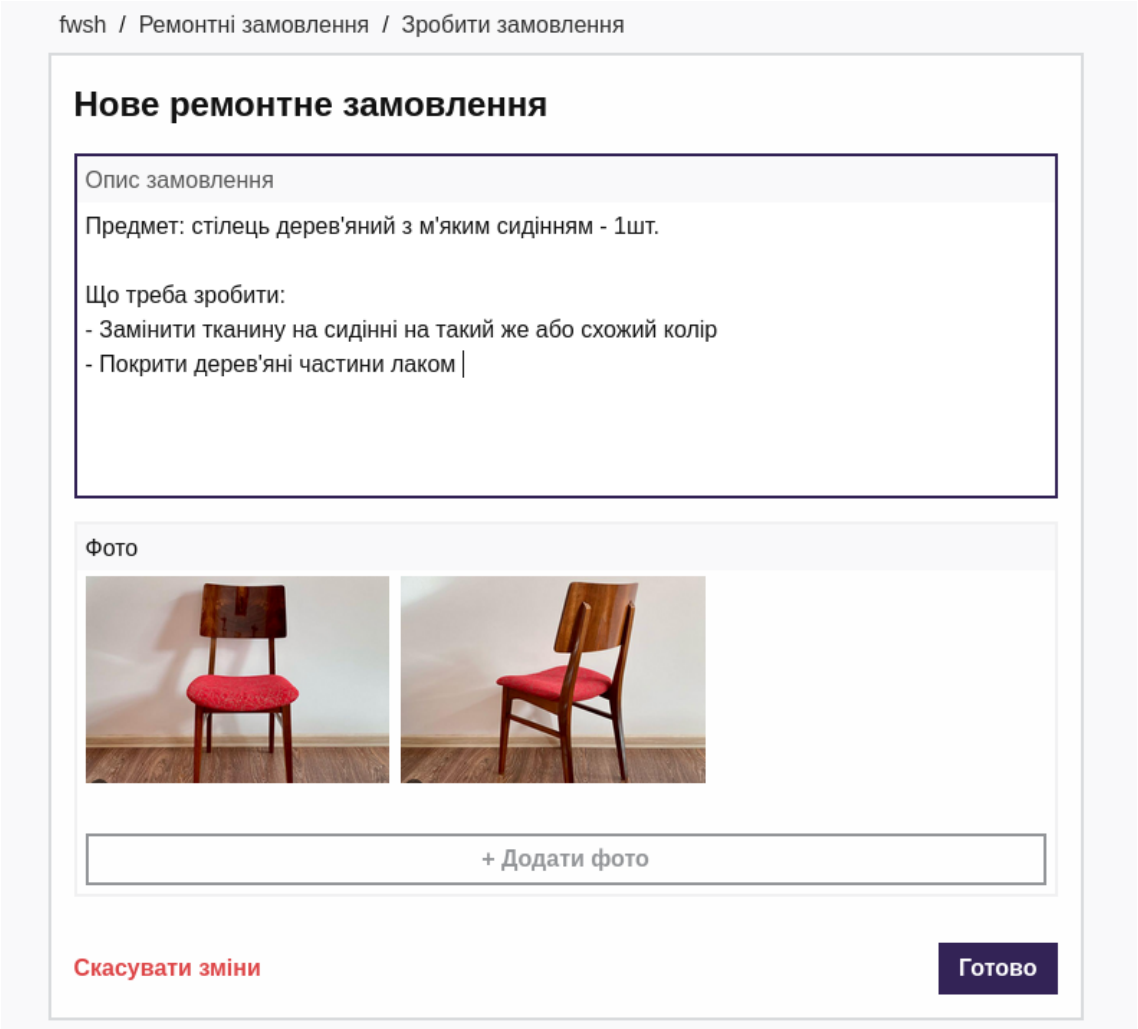

Рисунок Е.65 – Створення ремонтного замовлення

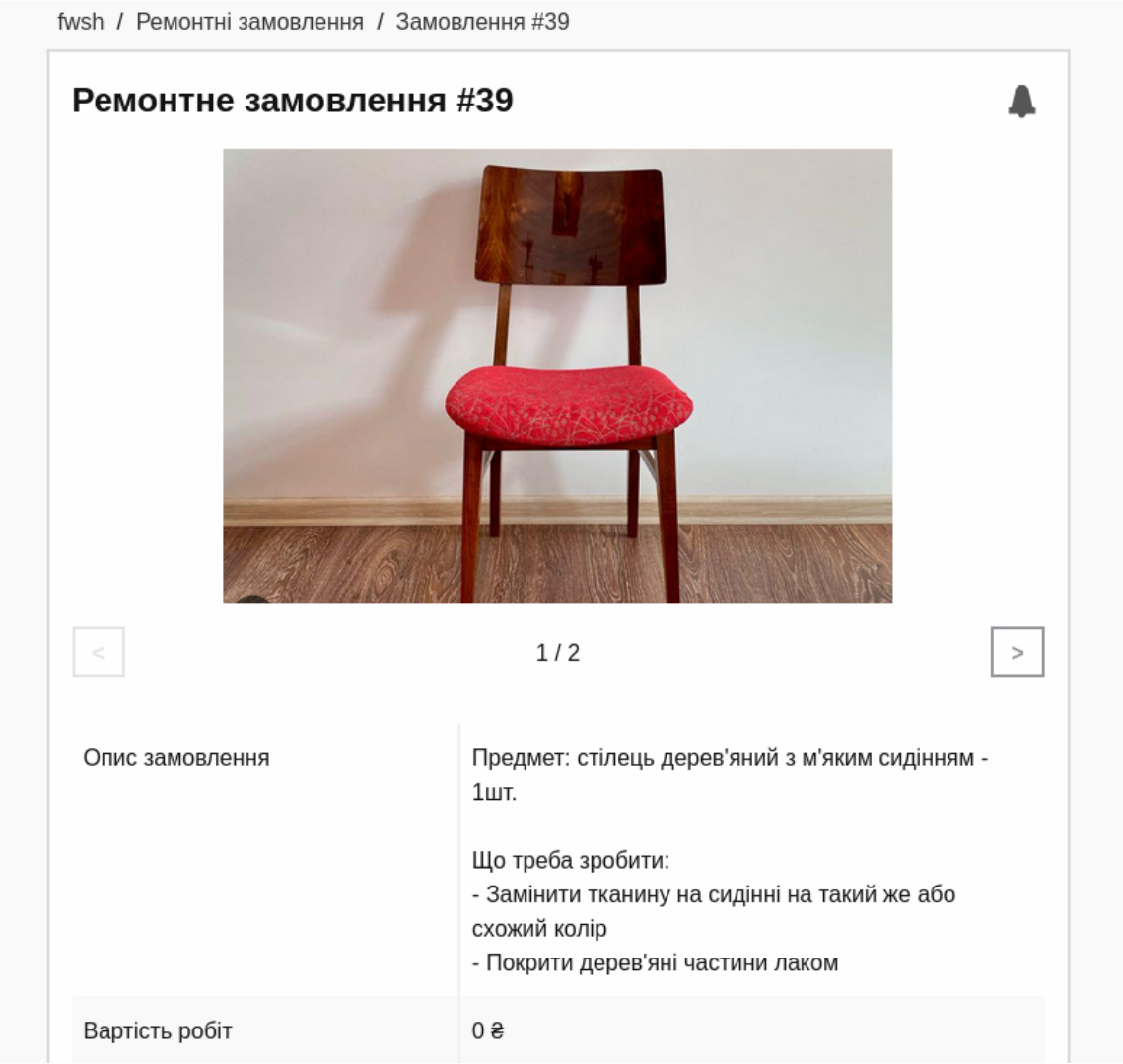

Рисунок Е.66 – Перегляд стану замовлення

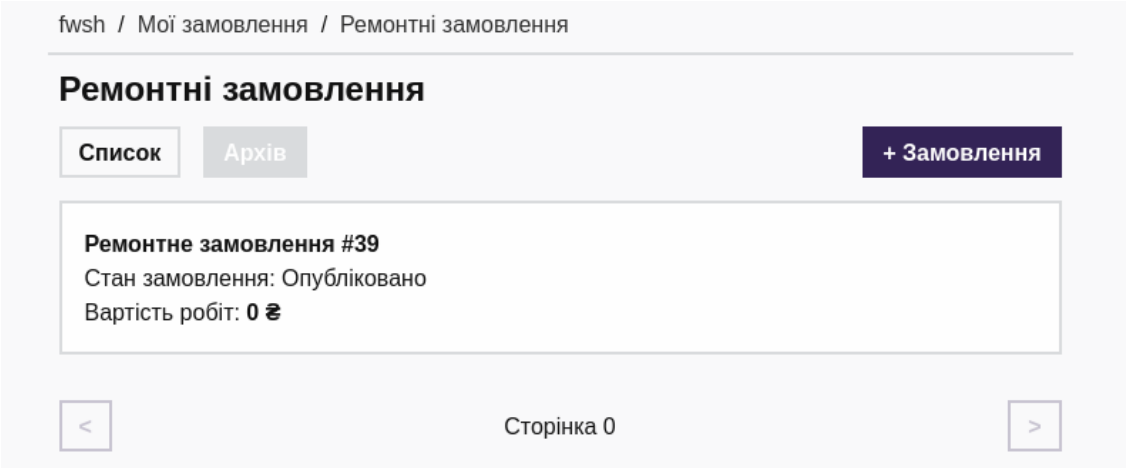

Рисунок Е.67 – Список опублікованих замовлень