МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Інститут післядипломної освіти

Кафедра інформаційних технологій

Кваліфікаційна робота бакалавра

на тему: Розробка веб застосунку для підтримки бізнесу з прокату речей

Виконав студент групи КН-5
спеціальності 122 «Комп'ютерні науки»
Таченко Олексій Ігорович

Керівник д.техн.н., професор
Казакова Надія Феликсівна

Консультант _____

Рецензент Попов В.Л. _____

Одеса 2023

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – база даних

ОС – операційна система

ПЗ – програмне забезпечення

ADT – Android Development Tools – Інструменти розробки в ОС Android

API – Application Programming Interface – програмний інтерфейс програми

AR – Augmented Reality – доповнена реальність

IDE – Integrated Development Environment – інтегроване середовище розробки

SDK – Software Development Kit – набір засобів розробки

ВСТУП

В наш час досить актуальним є розробка зручних та гнучких сервісів для ведення власного бізнесу. Практично кожна фірма, компанія або бізнес використовує подібні програмні продукти з метою обліку клієнтів та виконаних операцій з ними.

Все частіше подібні сервіси розробляються у вигляді Web-застосувань чи Web-сайтів. Переваги цього виду сервісу в тому, що користувач може використати його на любій операційній системі, з любого робочого місця через Інтернет і робота з ним не потрібує встановлення десктопної програми. Крім того, веб-застосунок можна запускати за допомогою смартфону, який завжди в кишені.

В роботі необхідно розробити подібний сервіс у вигляді Web-сайту для ведення обліку операцій з клієнтами та контролю бізнесу для підприємців, які здають своє обладнання в оренду. На сайті буде міститися інформація про обладнання, послуги, сервіс, клієнтів та інше, що безумовно скоротить витрати на телефонні переговори, диспетчерську службу, прайс-лист та інше.

Важливо перед тим як створювати Web-сайт ретельно продумати дизайн сайту, його меню, він повинен бути цікавим, легким у використанні і доступним. Інформація, яка розміщується на сайті, повинна бути завжди актуальною і оновлюватися. Також слід проаналізувати конкурентів, їх недоліки та переваги, що б зробити сайт таким, яким би його хотіли бачити потенційні замовники. Проаналізувати яка інформація цікавила б їх в першу чергу. Дизайн додатку повинен відповідати темі сайту і інформації, яка буде розміщена на ньому.

Крім того, безумовно важливим етапом є вибір засобів розробки як для серверної так і клієнтської частини, а також організація безпеки даних Web-додатку.

Метою кваліфікаційної роботи є створення веб-додатку для допомоги ведення бізнесу з прокату речей. Для досягнення поставленої мети були сформульовані наступні завдання:

- провести аналіз предметної області щодо організації бізнесу з прокату речей;
- провести порівняльний аналіз існуючих програм аналогів;
- обґрунтувати вибір програмних засобів розробки та технологій;
- провести проектування програмної системи з використанням мови UML;
- виконати реалізацію клієнтської та серверної частин веб-додатка;
- підготувати інструкцію користувача;
- виконати тестування веб-додатку.

Структура дипломної роботи складається з вступу, чотирьох розділів, висновків, переліку посилань на 15 найменувань, додатків. Повний обсяг проекту становить 70 сторінок, містить 25 рисунків і 1 таблицю.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПРОГРАМНИХ СИСТЕМ АНАЛОГІВ

1.1 Аналіз предметної області

Прокат – це різновид термінового договору оренди, відповідно до якого особою, що здійснює підприємницьку діяльність, у тимчасове користування орендарю передається майно [1].

Договір прокату – це окремий випадок договору оренди. У прокат може бути передано тільки рухоме майно. Прокат завжди спрямований на отримання прибутку. Плата за прокат речі встановлюється за тарифами наймодавця. Наймач має право відмовитися від договору прокату і повернути річ наймодавцеві у будь-який час.

Процес оформлення заявки на прокат обладнання наступний:

1) З наймачем укладається договір прокату.
2) Після перевірки зовнішнього вигляду і працездатності обладнання, сторонами підписується договір прокату і акт передачі, який є важливою частиною договору, так як містить опис товару, взятого напрокат.
3) Після підписання, наймач погоджується з тією комплектацією, яка в ньому вказана, тому варто перевірити її уважно, перш ніж підписувати.
4) Після закінчення терміну прокату, якщо договір не подовжено, наймач зобов'язаний повернути взяте напрокат обладнання в належному стані. Наявність всіх комплектуючих, зазначених в договорі є обов'язковою. Якщо клієнт повертає не все обладнання, то додатковий термін прокату оплачується як продовження договору.
5) У разі появи у обладнання дефектів, які виникли в процесі експлуатації наймачем, він оплачує штраф у вигляді застави, взятого у нього при оформленні договору.

Система обліку є найважливішою частиною бізнес-інструментів підприємця. Без цієї системи неможливо впоратися з урахуванням матеріалів при веденні підприємницької діяльності.

Щоб використовувати інформацію в обліковій підприємницької діяльності, вона повинна дотримуватися певних критеріїв:

- інформація повинна бути чіткою, не містити нічого зайвого;
- користувач повинен бути впевнений, що інформація достовірна;
- інформація повинна бути своєчасною;
- інформація повинна бути адресною, орієнтованої на конкретний вид діяльності;
- інформація повинна бути цільовою, спрямованої на вирішення конкретних завдань управління;
- інформація повинна бути економічною в отриманні і використанні;
- інформація повинна бути конфіденційною.

Збір та обробка облікових даних, спрямованих на вирішення поставленого управлінського завдання в широкому сенсі називається економічним або управлінським обліком.

Правильна організація інформаційних потоків є найважливішим фактором управління. За якість прийнятих управлінських рішень відповідає об'єктивність і достовірність отриманої інформації, а також оперативність її отримання.

Інформація є центральною ланкою в сполучних процесах управління. Значимість сполучних процесів визначається тим, що за допомогою їх здійснюється взаємозв'язок загальних функцій управління. Також інформація є сировиною. Вона необхідна для прийняття рішень управлінської діяльності.

Система звітності та інформації для управління грають роль засобів комунікації і виконують найважливіше завдання, яка полягає в передачі даних з систем планування і контролю на ті рівні менеджменті, які відповідальні за прийняття рішень з певних питань.

Для зручності і залучення як постійних, так і потенційних клієнтів, було вирішено створити сторінку в мережі Інтернет, тобто веб-сайт.

Сайт необхідний підприємцю для того, щоб відвідувачі змогли дізнатися всю інформацію про обладнання, послуги, нові позиції, просто зайшовши на сайт фірми в Інтернеті. Там же можна дізнатися про саму фірму, залишити по зворотного зв'язку заявку або навіть зв'язатися з фахівцем, і уточнити важливі питання, що цікавлять.

Також зручність полягає в тому, що на сайт можна заходити цілодобово, і знайомитися з наявністю обладнання. Так само до переваг власної сторінки підприємства в мережі Інтернет відноситься: зниження витрат на прайслисти, диспетчерську службу, телефонні переговори.

Так само для клієнтів з інших міст надаються послуги фірми через вебсайт, з обробкою заявки і швидкою доставкою обладнання на об'єкт.

З огляду на те, що у компанії існує висока конкуренція, для сайту потрібно розробити адаптивний дизайн, докладний опис послуг, що надаються, та зручний інтерфейс користування. Що б зробити сайт відомим цільової аудиторії, змусити його працювати на себе, потрібно використовувати основні інструменти Інтернет-маркетингу, такі як:

- розмістити сторінку в відомих пошукових системах;
- розміщення на відвідуваних сайтах;
- обмін банерами, посиланнями;
- контекстна реклама.

Наведені нижче корисні заходи для інформування про компанію в онлайні:

- дошки оголошень;
- прайс-листи і бази даних;
- каталоги;
- новини.

Більш того, через інтернет, маючи e-mail можна просувати свої пропозиції, акції, розповісти про свою компанію і чим вона займається.

Створений сайт підприємства потрібно зареєструвати в пошукових системах, online-каталогах, рейтингах, для легкого виявлення його користувачами мережі Інтернет. Так само, можна буде переглядати відвідуваність користувачів сайту, мати додаткове статичне діаграму про відвідини та користувачів.

Помістивши сайт підприємства у вигляді рекламної форми як банери і кнопки на сторінках самих відвідуваних сайтів Інтернету, такі як, Yandex, Google та інше, він заробить найбільш ефективніше. Такий рекламний хід, менш витратний, але ефективний, тому що користувачі, зайшовши на головну сторінку відомого пошукача, побачать рекламу сайту. Реклама на сайті буде оцінена відвідувачами, що вплине на імідж підприємства.

Також сайт можуть обговорювати абоненти мережі на різних як звичайних, так і спеціальних та професійних форумах. Крім цього обговорювати можуть як саме підприємство, так і викладений на загальний огляд його вебсайт, що здатне додати популярності, якщо відгуки будуть позитивними.

1.2 Аналіз програмних систем аналогів

Сьогодні на ринку представлено великий вибір програмних засобів, для автоматизації різних видів діяльності, в тому числі і в сфері прокату, частина з яких є десктопними застосунками. Для того щоб вирішити перераховані вище завдання, проведемо аналіз існуючих програмних продуктів для обліку підприємницької діяльності по прокату речей.

Українська обликова система від компанії Торгсофт, вартість термінальної версії близько 27 000 грн. на 1 робоче місце. Програма призначена для автоматизації пунктів прокату будь-яких предметів обладнання. Вона веде облік речей, що видаються, платежів і взаєморозрахунків з клієнтами і фор-

мує всі необхідні друковані документи і звіти. Інтерфейс програми показаний на рис.1.1.

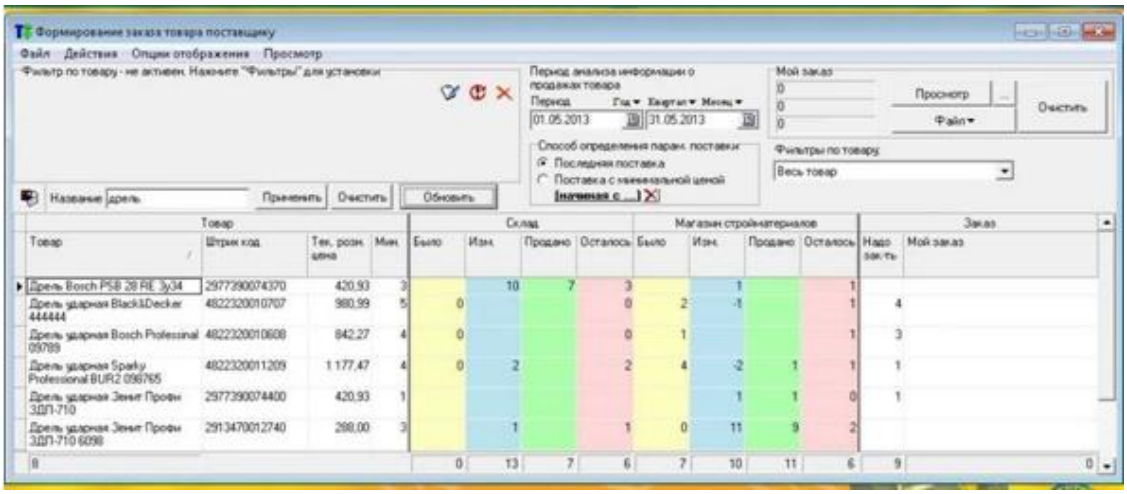

Рисунок 1.1 – Інтерфейс програми «Торгсофт»

Основні функціональні можливості прикладного рішення [2]:

- реєстрація клієнтів;
- зберігання даних про клієнтів;
- формування і друк персональних карток;
- занесення клієнта в «чорний список»;
- робота з фізичними та юридичними особами;
- можливість масової розсилки E-mail і SMS-повідомлень клієнтам;
- введення товарів, друк етикеток товарів;
- швидкий пошук клієнтів і товарів по їх штрих-кодами;
- видача і повернення з прокату;
- можливість обліку і обчислення вартості прокату, як по днях, так і по годинах;
- гнучка система налаштувань;
- прийом оплати від клієнтів, друк чеків;
- облік взаєморозрахунків з кожним клієнтом за всю історію;

– автоматичне введення касової книги, в якій реєструються всі операції з грошима;
– всілякі звіти по товарах;
– експорт даних в Microsoft (MS) Excel;
– різні звіти по клієнтах;

Крім того програма дозволяє проводити користувачу наступні операції:

– змінювати і додавати нові форми друкованих документів можуть самі користувачі програми;
– працювати в будь-якій валюті;
– розмежування доступ до функцій системи для співробітників пункту прокату;
– можливість обліку застав в різній формі;
– можливість обліку і зберігання документів клієнтів, товарів;
– простий інтерфейс користувача;
– детальна довідкова система.

Переваги програми «Торгсофт»:

– ретельно опрацьована система обліку клієнтів;
– отримання всіляких звітів;
– можливість створення своїх бонусних систем обліку;
– робота зі сканером штрих-кодів.

Недоліки:

– система представлена у десктопному варіанті;
– платна підтримка;
– надмірний функціонал.
– Висновок: система має низку незаперечних переваг, однак її використання є недоцільним через наявні недоліки, перш за все стосуються надмірної функціоналу, наявності лише десктопної версії та вартості.

Програма «RemOnline» вартість від 29$ до 99$ на місяць за 1 робоче місце. Програма призначена для автоматизації прокату будь-якого обладнан-

ня або мережі прокатів. Веде облік клієнтів і обладнання. Програма підвищує якість і швидкість обслуговування. Загальний вигляд програми представлений на рис.1.2.

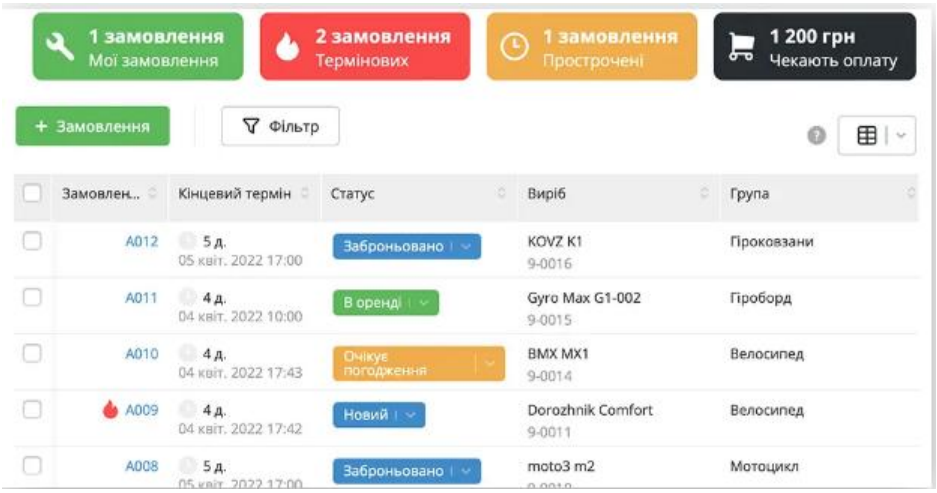

Рисунок 1.2 – Интерфейс программы «EasyPro»

Основні функціональні можливості прикладного рішення [3]:

- зручний каталог обладнання;
- база клієнтів;
- чорні списки клієнтів;
- настройка бонусної системи;
- швидкий пошук потрібних товарів на обрані дати;
- нагадування про прострочені прокатах;
- робота зі сканером штрих-кодів і чекових принтером;
- можливість роботи з комп'ютера або планшета;
- облік витрат;
- оцінка ефективності реклами;
- необхідні звіти;
- експорт даних в MS Excel, PDF, MS Word;
- друк документів і звітності;

- можливість інтеграції додатка з сайтом;
- SMS-повідомлення і E-mail розсилка;
- можливість налаштування програми з урахуванням особливостей бізнесу.

Переваги сервісу:

- великий набір звітів, що настроюються, експорт в різні формати;
- не вимагає установки;
- скорочення витрат часу на облік;
- система працює через Інтернет на хмарному сервері.

Недоліки сервісу:

- додаткова плата за кожне робоче місце;
- надмірний функціонал.

Висновок: дане рішення має ряд переваг, однак його використання недоцільно через наявних недоліків, перш за все стосується вартості продукту, впровадження та супроводу.

Програма «АБ Система», призначена для ведення своєї діяльності з прокату товарів, представляє собою єдину систему управління прокатним бізнесом, вартість 15 000 грн. Загальний вигляд програми представлений на рис.1.3.

Основні функціональні можливості прикладного рішення [4]:

- програма веде облік абсолютно всіх операцій з товарами, клієнтами та грошима;
- тонка настройка параметрів;
- наочна аналітика;
- внутрішня CRM веде історію відвідувань, нагадує про простроченнях платежів, а також веде облік призначених для користувача переваг;
- мотивації співробітників, сервіс дозволяє регулювати відсоток від продажів для співробітників;

– управління доступом до функцій програми для окремих співробітників;
– технічна підтримка;
– автоматичний розрахунок вартості.

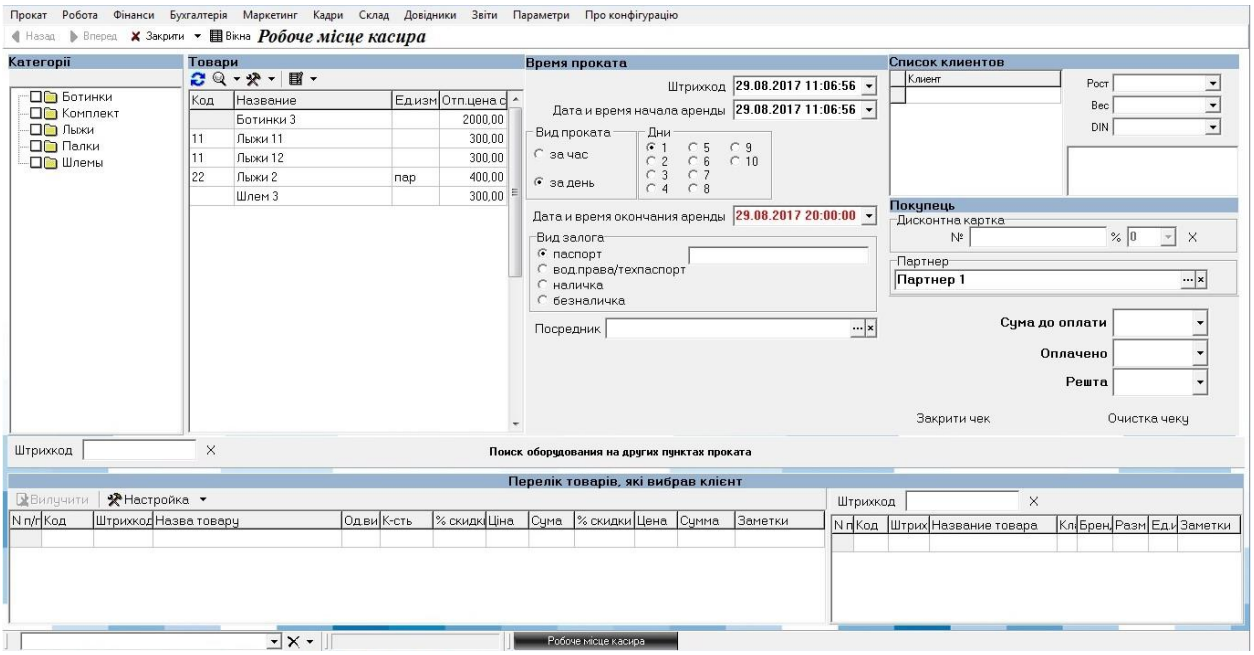

Рисунок 1.3 – Інтерфейс програми АБ Система

Переваги сервісу:

– універсальність, підходить для різних типів прокатних бізнесів;
– простий і зрозумілий інтерфейс;
– наявність мобільного застосування та веб-панелі дозволять організувати і контролювати роботу на будь-якому мобільному пристрої з будь-якої точки світу;
– база даних зберігається на захищеному сервері в хмарі, що забезпечує додаткові гарантії збереження даних.

Недоліки сервісу:

– додаткова плата за кожне робоче місце;
– надмірний функціонал.

Висновок: дане рішення має ряд переваг, однак його використання недоцільно через наявних недоліків, перш за все стосується вартості продукту, та надмірного функціоналу, що потрібує професійної технічно підтримки. Виконав аналіз існуючих розробок для прокату, був зроблений висновок, що всі вони мають свої переваги і недоліки. Головним недоліком став перевантажений функціонал, який зазвичай не потрібний для малого підприємства, а також висока вартість, тому є необхідність в розробці нового програмного забезпечення. В таблиці 1.1 представлені зведені дані аналізу існуючих програмних продуктів і програми, що розробляється («Rent Manager»).

Таблиця 1.1 – Порівняння існуючих програмних продуктів для обліку прокатного бізнесу

Характеристики	Торгсофт	RemOnline	АБ Система	Rent Manager
Зручний інтерфейс	+	+	+	+
Універсальність налаштувань	+	+	+	+
Функціонал необхідний для малого бізнесу	–	–	–	+
Мінімальні технічні вимоги	+	+	+	+
Низька вартість	–	–	–	+
Оцінка	3(+)	3(+)	3(+)	5(+)

Малому бізнесу як правило потрібне програмне забезпечення, що враховує особливості його діяльності, специфіку функціонування, що дозволяє виконувати певні функції обліку, актуальні на даний момент, а також

проводити гнучку настройку і розширення функціоналу в разі необхідності в процесі використання програмного продукту.

2 ВИБІР ТА ОБГРУНТУВАННЯ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Функціональні та нефункціональні вимоги

Перед початком розробки веб-додатку були визначені функціональні та нефункіональні вимоги до нього.

До функціональних вимог належать:

– можливість додавати, редагувати та видаляти товари;

– можливість додавати, редагувати та видаляти замовлення;

– можливість додавати, редагувати та видаляти клієнтів;

– реєстрація та авторизація;

– можливість переглядати замовлення в календарі.

До нефункціональних вимог віднесемо такі пункти:

– браузер найновішої версії;

– підключення до Інтернету.

2.2 Вибір мов програмування та середовищ розробки

Для розробки клієнтської частини було обрано середовище розробки IDE Visual Studio Code та мова програмування TypeScript. Нижче обґрунтуємо саме такий вибір засобів реалізації для клієнта.

Мова програмування TypeScript позиціонується як засіб розробки веб-застосунків, що розширює можливості JavaScript. Її перевагами виступать: статична типізація, підтримка використання повноцінних класів та підтримка підключення модулів [5].

Розробником мови TypeScript є Андерс Гейлсберг, який створив раніше C#, Turbo Pascal і Delphi.

Код експериментального компілятора, котрий транслює код TypeScript в представлення JavaScript, поширюється під ліцензією Apache, розробка ведеться в публічному репозиторії через сервіс CodePlex. Специфікації мови

відкриті і опубліковані в рамках угоди Open Web Foundation Specification Agreement (OWFa 1.0).

TypeScript є зворотньо сумісним з JavaScript. Фактично, після компіляції програму на TypeScript можна виконувати в будь-якому сучасному браузері або використовувати спільно з серверною платформою Node.js.

Переваги над JavaScript:

- можливість явного визначення типів (статична типізація);
- підтримка використання повноцінних класів (як в традиційних об'єктно-орієнтованих мовах);
- підтримка підключення модулів.

Ці нововведення підвищили швидкість розробки, прочитність, рефакторинг і повторне використання коду, пошук помилок на етапі розробки та компіляції, а також швидкодію програм.

Підтримка динамічної типізації зберігається – компілятор TypeScript успішно обробляє і не модифікований код на JavaScript. Основний принцип мови – весь існуючий код на JavaScript сумісний з TypeScript, тобто в програмах на TypeScript можна використовувати стандартні JavaScript-бібліотеки і раніше створені напрацювання. Більш того, можна залишити існуючі JavaScript-проекти в незмінному вигляді, а дані про типізації розмістити у вигляді анотацій, які можна помістити в окремі файли, які не заважатимуть розробці і прямому використанню проекту (наприклад, подібний підхід зручний при розробці JavaScript-бібліотек).

Visual Studio Code – засіб для створення, редагування та зневадження сучасних веб-застосунків і програм для хмарних систем. Visual Studio Code розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux і OS X [6].

Компанія Microsoft представила Visual Studio Code у квітні 2015 на конференції Build 2015. Це середовище розробки стало першим кросплатформовим продуктом у лінійці Visual Studio.

Visual Studio Code – це інтегроване середовище розробки для роботи з фреймворком Angular, та і не тільки. В данній IDE присутній інтегрований регулятор версій git, що полегшує роботу з github.

Переваги даної IDE в тому, що вона надає різні інструменти і широкі можливості для швидкого та комфортної розробки програмного забезпечення. Також за допомогою Visual Studio Code можна проводити тестування готового продукту.

Редактор містить вбудований зневаджувач, інструменти для роботи з Git і засоби рефакторингу, навігації по коду, автодоповнення типових конструкцій і контекстної підказки. Продукт підтримує розробку для платформ ASP.NET і Node.js, і позиціонується як легковагове рішення, що дозволяє обійтися без повного інтегрованого середовища розробки

Для серверної частини було обрано мову програмування Java та середовище розробки Intellij IDEA.

Java – об'єктно-орієнтована мова програмування, що була розроблена компанією Sun Microsystems в 1991 році і офіційно випущена 23 травня 1995 року. Відмінною особливістю Java в порівнянні з іншими мовами програмування загального призначення є забезпечення високої продуктивності програмування та ефективність використання пам'яті [7].

В Java використовуються практично ідентичні угоди для оголошення змінних, передачі параметрів, операторів і для управління потоком виконанням коду. В Java додані всі хороші риси C ++. Мова Java вже багато років є сандартом у розробці корпотивних систем та має величезну кодову базу та багато готових рішень.

Java надає програмісту багатий набір класів об'єктів для ясного абстрагування багатьох системних функцій, використовуваних при роботі з вікнами, мережею і для введення-виведення. Ключова риса цих класів полягає в тому, що вони забезпечують створення незалежних від використовуваної платформи абстракцій для широкого спектра системних інтерфейсів.

Величезна перевага Java полягає в тому, що на цій мові можна створювати застосунки, здатні працювати на різних платформах.

Середовище розробки Intellij IDEA – комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala) від компанії JetBrains. Дана IDE дає можливість роботи з системами контролю версій (Git), різноманітними базами даних, а також підключати різноманітні плагіни для комфортної розробки ПЗ. Також за допомогою Intellij IDEA можна проводити тестування готового продукту [8].

Програма містить повний набір необхідних для створення повноцінних додатків компонент: редактор, середа компіляції і виконання, а також відладчик.

Природно, IntelliJ IDEA – не єдина середовище створення додатків для Java, можна також використовувати Eclipse або NetBeans, але IntelliJ IDEA все ж дуже популярна серед розробників. Програму можна розглядати як інтелектуальне середовище розробки, яка розуміє код. У процесі його написання програмістом вона займається побудовою синтаксичного дерева, визначенням особливостей розміщених посилань, аналізом можливих шляхів виконання операторів і передачі даних. Ґрунтуючись на отриманих результатах, програма звертає увагу фахівця на існуючі помилки і самостійно усуває їх, надає варіанти автоматичного доповнення коду. Завдяки зазначеним особливостям вона позбавляє користувача від повсякденної рутини і дозволяє йому сконцентруватися на більш важливих завданнях.

Дана програма допомагає фахівцеві економити час внаслідок глибокого аналізу контексту і видалення невідповідних варіантів. Ця та інші деталі забезпечують підвищення рівня продуктивності користувача, одночасно дозволяючи йому отримувати більше задоволення від діяльності. Враховуючи все вище сказане та суб’єктивний досвід будемо використовувати в роботі саме IDE Intellij IDEA.

2.3 Опис технологій розробки

Розробка клієнта велася за допомогою фреймворка Angular та Bootstrap.

2.3.1 Опис технології Angular

Angular – написаний на TypeScript front-end фреймворк з відкритим кодом, який розробляється під керівництвом Angular Team у компанії Google, а також спільнотою приватних розробників та корпорацій. Angular – це AngularJS, який переосмислили та який був повністю переписаний тією ж командою розробників. [9].

Додано Angular CLI, що дає змогу розпочати створення нового додатка, просто написавши команду ng new [app name]. Angular не використовує концепцію «області видимості» або контролерів, натомість як головну архітектурну концепцію він застосовує ієрархію компонентів

Angular має інакший синтаксис написання виразів. Значна частина основного функціоналу перенесена у модулі. Angular рекомендує та застосовує розроблену Microsoft мову – TypeScript, що містить такі можливості, як:

- класи, що відповідають парадигмі об'єктно-орієнтованого програмування;
- система типізації;
- узагальнене програмування.

Angular також має такі можливості, як:

- анонімні функції;
- ітератори;
- цикли типу for/of
- python-подібні генератори;
- рефлексія.

Angular надає можливості розділення логіки від представлення, тобто логіка клієнта йде окремо від html-розмітки.

2.3.2 Опис технології Bootstrap

HTML-розмітка та CSS-стилі будуть написані на Bootstrap.

Bootstrap – це вільний набір інструментів з відкритим кодом для створення сайтів і веб-додатків. Включає в себе HTML і CSS-шаблони оформлення для типографіки, веб-форм, кнопок, міток, блоків навігації та інших компонентів веб-інтерфейсу, включаючи JavaScript-розширення. Він спрощує розробку динамічних веб-сайтів і веб-додатків [10].

Четверта версія фреймворку має числені оновленнями:

- використано синтаксис Sass замість Less;
- покращено процес верстки макетів, зокрема блочної структури;
- додано підтримка flexbox, компоненту з HTML5;
- прев'ю та панелі замінено компонентом «карти» – це віднині невеликі за форматом елементи з прев'ю зображень, текстових блоків з бордерами;
- всі HTML-резети зібрано в єдиному модулі під назвою «Reboot» (в попередніх версіях цей код зберігався в Normalize.css);
- додано нові можливості з кастомізації шаблонів. Для оновлення стилів за замовчанням досить відредагувати змінну в Sass-файлі і отримати оновлених файл css.
- скасовано підтримку IE8;
- розміри вказано у rem і em замість пікселів, що покращує мобільний вигляд фреймворку;
- оновлено всі плагіни JavaScript;
- оновлено роботу спливних вікон і підказок;
- покращено документацію і пошук сайтом фреймворку.

Bootstrap сумісний з останніми версіями браузерів Google Chrome, Firefox, Internet Explorer, Opera і Safari (деякі з цих браузерів підтримуються не на всіх платформах).

Bootstrap має модульну структуру і складається переважно з наборів таблиць стилів LESS, які реалізують різні компоненти цього набору інструментів. Розробники можуть самостійно налаштовувати файли Bootstrap, обираючи компоненти для свого проекту.

Основні інструменти Bootstrap:

- сітки (grid) – наперед задані, готові до використання колонки
- шаблони (template) – фіксовані чи адаптивні шаблони сторінок
- типографіка (typography) – опис та визначення класів для шрифтів, таких як шрифти для коду, цитат тощо
- мультимедіа (media) – засоби управління зображеннями та відео
- таблиці (table) – засоби оформлення таблиць, які зокрема забезпечують сортування
- форми (form) – класи для оформлення як форм, так і деяких подій
- навігація (nav, navbar) – класи для оформлення вкладок, сторінок, меню і панелей навігації
- сповіщення (alert) – класи для оформлення діалогових вікон, підказок і спливаючих вікон
- іконочний шрифт (icon font) – набір іконок у вигляді шрифту, складається майже з 500 компонентів.

2.3.3 Опис технології Spring

Розробка серверної частини велася за допомогою ряду підпроектів Spring Framework.

Spring Framework — це програмний каркас (фреймворк) з відкритим кодом та контейнери з підтримкою інверсії управління для платформи Java. Основні особливості Spring Framework можуть бути використані будь-яким

додатком Java, але є розширення для створення веб-додатків на платформі Java EE. Незважаючи на це, Spring Framework не нав'язує якоїсь конкретної моделі програмування, Spring Framework став популярним в спільноті Java як альтернатива, або навіть доповнення моделі Enterprise JavaBean (EJB) [11].

Spring Framework складається з кількох модулів, які надають широкий спектр послуг [12]:

- контейнер інверсії управління: конфігурація компонентів додатків і управління життєвим циклом об'єктів Java, здійснюється головним чином через інверсію управління;
- аспектно-орієнтоване програмування: дозволяє реалізувати наскрізні процедури;
- доступ до даних: робота з реляційною системою управління базами даних на платформі Java з використанням JDBC і об'єктно-реляційні відображення та інструментів з NoSQL баз даних;
- управління транзакціями: об'єднує кілька API, управління транзакціями та координує операції для Java-об'єктів;
- модель-вигляд-управління (Model-View-Controller): програмний каркас на основі HTTP сервлета, що забезпечує створення веб-додатків і веб-служб RESTful;
- аутентифікація і авторизація: налаштовувані процеси безпеки, які підтримують цілий ряд стандартів, протоколів, інструментів і практик за допомогою підпроекту Spring Security (колишня система безпеки AcerI для Spring).
- віддалене керування: конфігураційний вплив і управління Java-об'єктами для місцевої (локальної) або віддаленої конфігурації через JMX;
- тестування: підтримка класів для написання юніт-тестів та інтеграційних тестів.

Spring Framework неодноразово піддавали критиці за надмірну прив’язаність до XML в контейнерах Spring. Проте, починаючи з версії 3.0.0, розробники мають можливість використовувати повністю або частково в своїх застосунках анотації. Spring Boot широко використовує даний спосіб для власних конфігурацій. Понад те, Spring Tool Suite (STS), побудований на базі Eclipse, забезпечує автодоповнення коду, валідацію, контекстну інформацію та графічну візуалізацію під час редагування файлів конфігурації Spring XML [13].

Для швидкого, якісного та зручного написання бізнес логіки проекту використовувався Spring Boot. Він дозволяє забути про довготривалі налаштування та конфігурації сервера додатку та бази даних.

Spring Security використовувався для реалізації серверної безпеки, а також реєстрації та авторизації.

Spring Security – це Java/Java EE фреймворк, що надає механізми побудови систем аутентифікації та авторизації, а також інші можливості забезпечення безпеки для промислових додатків, створених за допомогою Spring Framework. Проект був розпочатий Беном Алексом (Ben Alex) в кінці 2003 року під ім'ям «Acegi Security» і був публічно представлений під ліцензією Apache License в березні 2004. Згодом був включений в Spring як офіційний дочірній проект.

2.3.4 Опис фреймворку Apache Maven

В якості засобу автоматизації управління та складання проекту використовувався Apache Maven.

Apache Maven – це засіб автоматизації роботи з програмними проектами, який спочатку використовувався для Java проектів. Використовується для управління (management) та складання (build) програм. За принципами роботи кардинально відрізняється від Apache Ant, та має простіший вигляд щодо build-налаштувань, яке надається в форматі XML. XML-файл описує проект,

його зв'язки з зовнішніми модулями і компонентами, порядок будування (build), папки та необхідні плагіни. Сервер із додатковими модулями та додатковими бібліотеками розміщується на серверах. Раніше Maven був частиною Jakarta Project [14].

Для опису програмного проекту, який потрібно побудувати (build), Maven використовує конструкцію відому як Project Object Model (POM), залежності від зовнішніх модулів, компонентів та порядку побудови. Виконання певних, чітко визначених задач – таких, як компіляція коду та пакетування відбувається шляхом досягнення заздалегідь визначених цілей (targets).

Ключовою особливістю Maven є його мережева готовність (network-ready).

Двигун ядра може динамічно завантажувати плагіни з репозиторію, того самого репозиторію, що забезпечує доступ до багатьох версій різних Java-проектів з відкритим кодом, від Apache та інших організацій та окремих розробників. Цей репозиторій та його реорганізований наступник, – Maven 2 репозиторій, – намагається бути де-факто механізмом для дистрибуції Java програм, але прийняття його в такій якості йде повільно.

Maven забезпечує підтримку побудови не просто перебираючи файли з цього репозиторію, але й завантажуючи назад артефакти у кінці побудови. Локальний кеш звантажених артефактів діє як первісний засіб синхронізації виходу проектів на локальній системі.

Maven базується на плагін-архітектурі, що дозволяє зробити використання будь-якої програми контрольованим через стандартний вхід. Теоретично, це могло б дозволити будь-кому писати плагіни для інтерфейсу з інструментами для побудови (компілятори, тестери тощо) для будь-якої мови.

2.3.5 Опис системи керування базами даних PostgreSQL

Для безпечного зберігання даних користувача використовувалась PostgreSQL.

PostgreSQL – об'єктно-реляційна система керування базами даних. Є альтернативою як комерційним СКБД (Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СКБД з відкритим кодом (MySQL, Firebird, SQLite) [15].

Порівняно з іншими проектами з відкритим кодом, такими як Apache, FreeBSD або MySQL, PostgreSQL не контролюється якоюсь однією компанією, її розробка можлива завдяки співпраці багатьох людей та компаній, які хочуть використовувати цю СКБД та впроваджувати у неї найновіші досягнення.

Сервер PostgreSQL написаний на мові C. Зазвичай розповсюджується у вигляді набору текстових файлів із сирцевим кодом. Для інсталяції необхідно відкомпілювати файли на своєму комп'ютері і скопіювати в деякий каталог.

Дана БД надає величезну кількість інструментів та можливостей для зберігання та обробки даних різних типів, від чисел до даних в форматах JSON та LOB.

Для легких маніпуляцій з базою даних вбула використана технологія об’єктно-реляційного мапінгу. Найпопулярнішим представником, який неофіційно став стандартом, є ORM Hibernate.

До переваг PostgreSQL слід віднести:повну сумісність з SQL та об'єктно-орієнтований принцип.

Недоліки PostgreSQL:складність розробки та низька швидкість читання даних.

Висновки до розділу: у другому розділу бакалаврської роботи були проаналізовані технології та засоби реалізації за допомогою яких було реалізовано мобільний застосунок. Були складені функціональні та не функціональні вимоги до застосунку.

3 ПРОЕКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

На етапі проектування будуть створені мокапи інтерфейсу користувача, діаграма прецедентів та діаграма класів. Кожна із діаграм наглядно зображує систему, яка була реалізована, тому вони є необхідними для створення якісного програмного продукту.

3.1 Створення мокапів

Мокап (mock-up) – це макет, який дизайнери використовують для демонстрації своєї роботи. Таким способом замовник може переглянути дизайн сайту на екрані комп'ютера або мобільних пристроях і т.д. Завдяки мокап можна отримати не просто плоску фотографію, а реалістичне зображення. Використання шаблону спрощує взаємодію дизайнера і замовника.

Передбачається, що програма буде складатися з декілька вікон:

- вікно реєстрації;
- вікно авторизації;
- вікно головної сторінки, яке буде містити таблицю з поточними замовленнями, клієнтами та орендованими товарами користувача;
- вікно клієнтів, яке містить детальну інформацію про клієнтів користувача;
- вікно замовлень, яке містить детальну інформацію про замовлення клієнтів;
- вікно товарів – містить інформацію про товари користувача.

Вікно реєстрації (рис. 3.1) складається з основних полів вводу та кнопки реєстрації.

Вікно авторизації (рис/ 3.2) складається із полів вводу логіну та паролю, а також кнопки «Увійти» та «Зареєструватись».

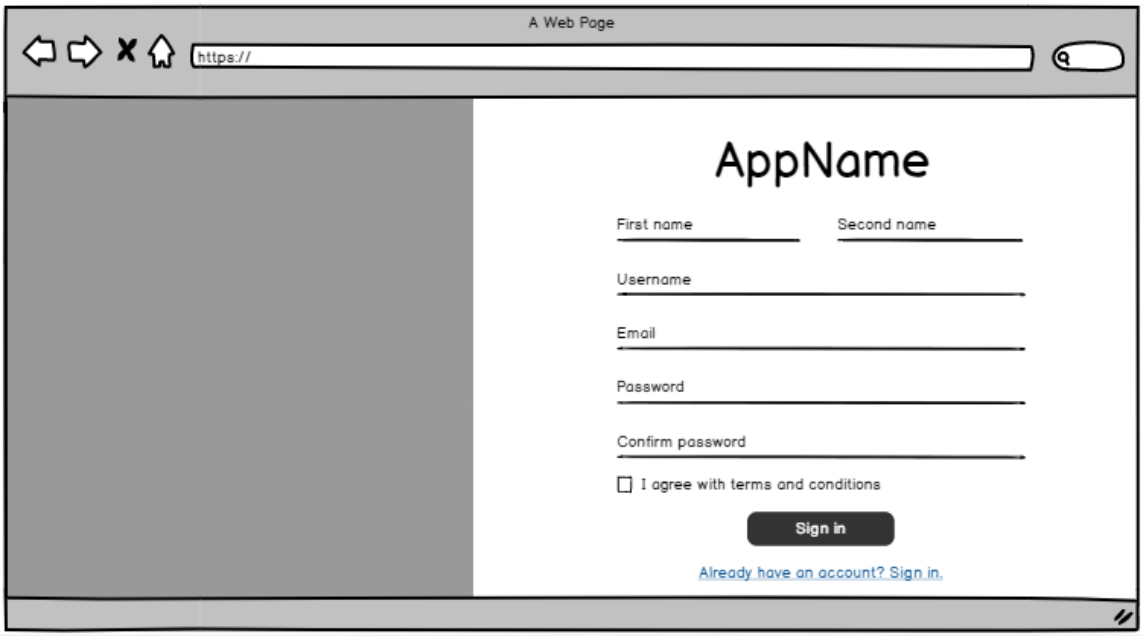

Рисунок 3.1 – Мокапи вікна реєстрації

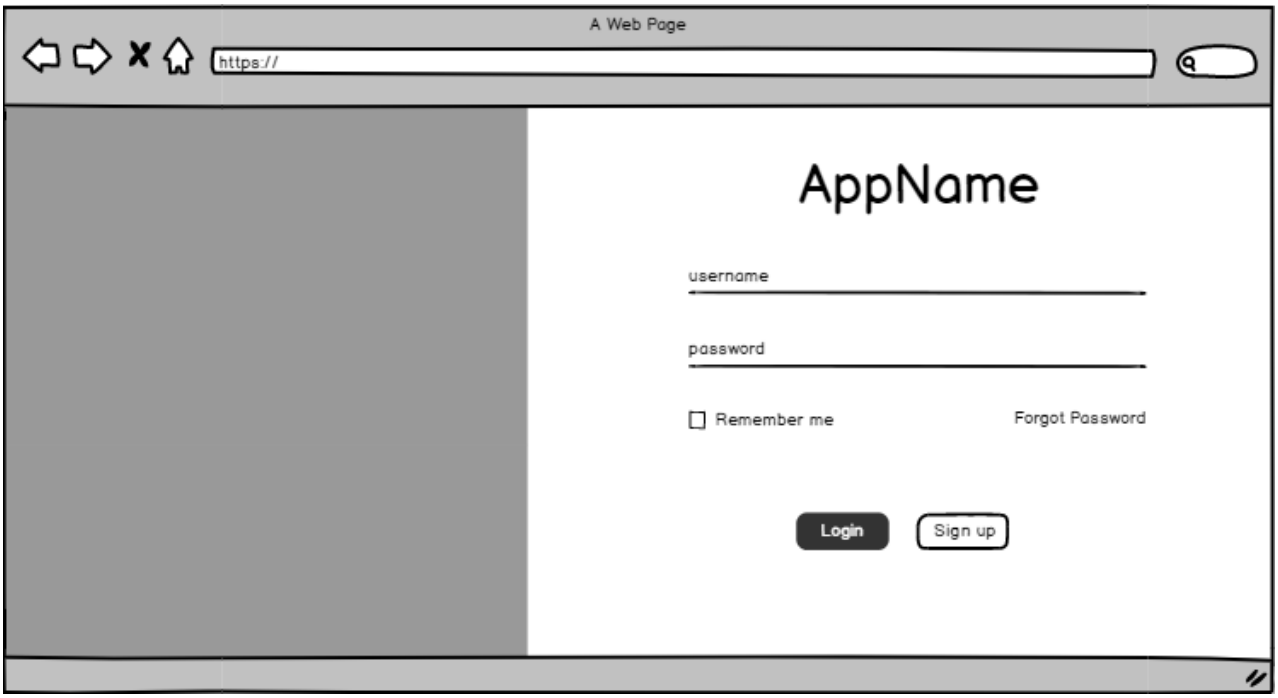

Рисунок 3.2 – Мокап вікна авторизації

Вікно головної сторінки (рис. 3.3) складається з таблиці з поточними замовленнями, клієнтами та орендованими товарами на день, вказаний користувачем.

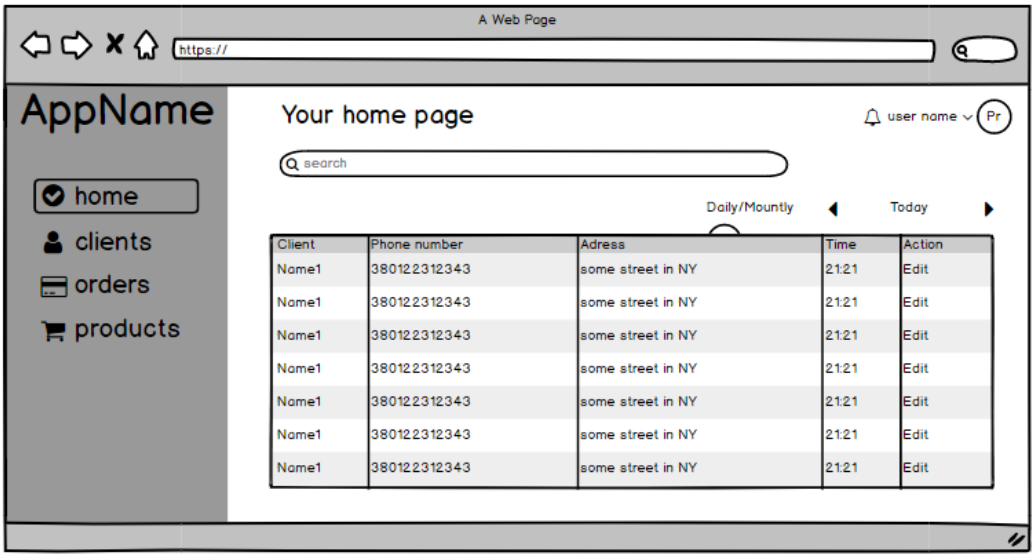

Рисунок 3.3 – Мокап головної сторінки на день

Вікно головної сторінки (рис. 3.4) складається з таблиці з поточними замовленнями, клієнтами та орендованими товарами користувача на місяць, а також календар.

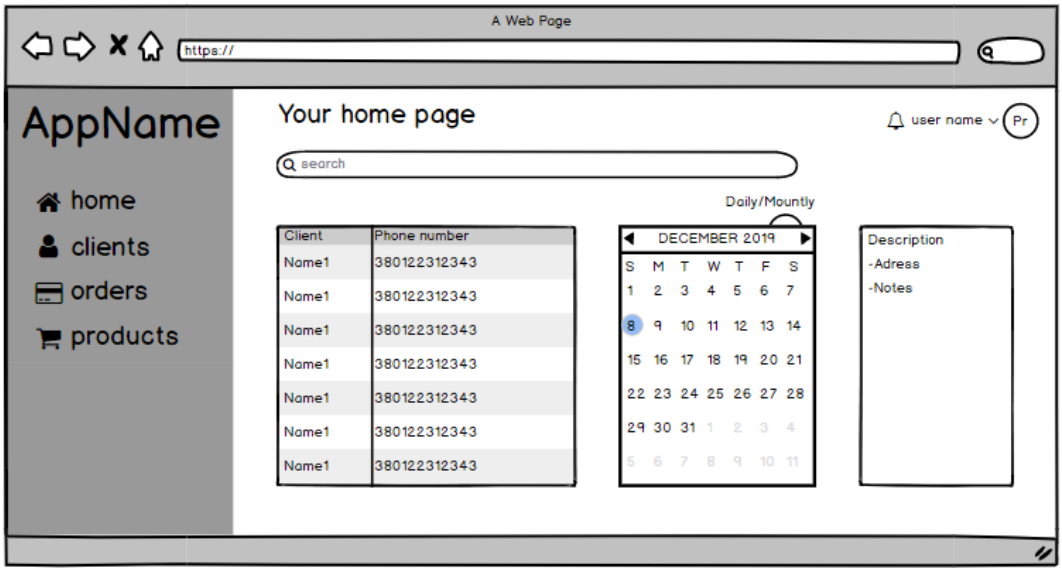

Рисунок 3.4 – Мокап головної сторінки на місяць

Вікно клієнтів (рис. 3.5) складається з карток, які містять інформацію про клієнта, кнопки створення нового клієнта.

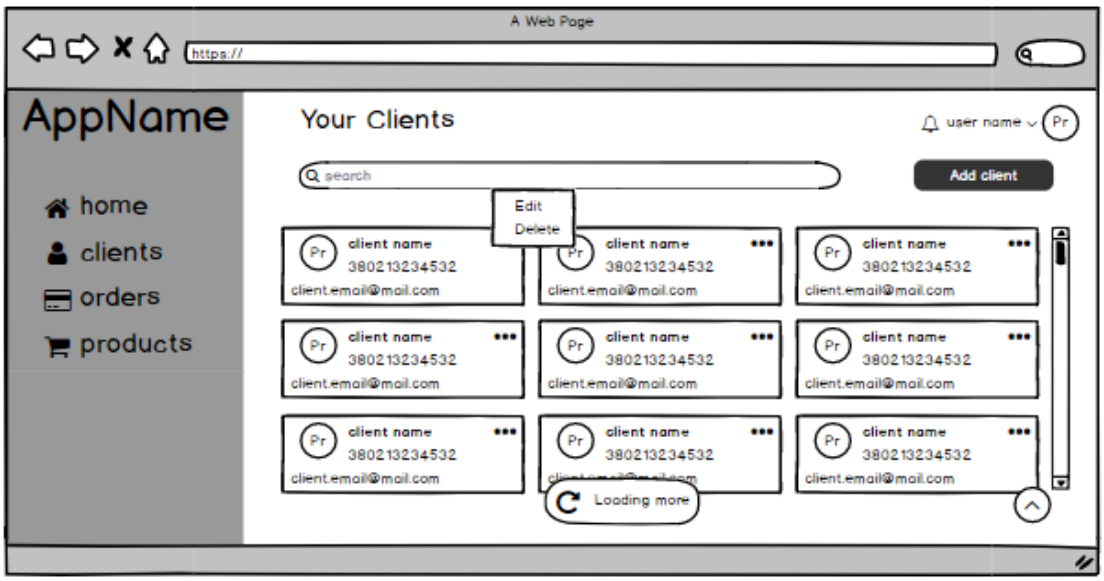

Рисунок 3.5 – Мокап сторінки з клієнтами

Вікно замовлень (рис. 3.6) складається з таблиці з детальною інформацією про замовлення, кнопки додавання нового замовлення.

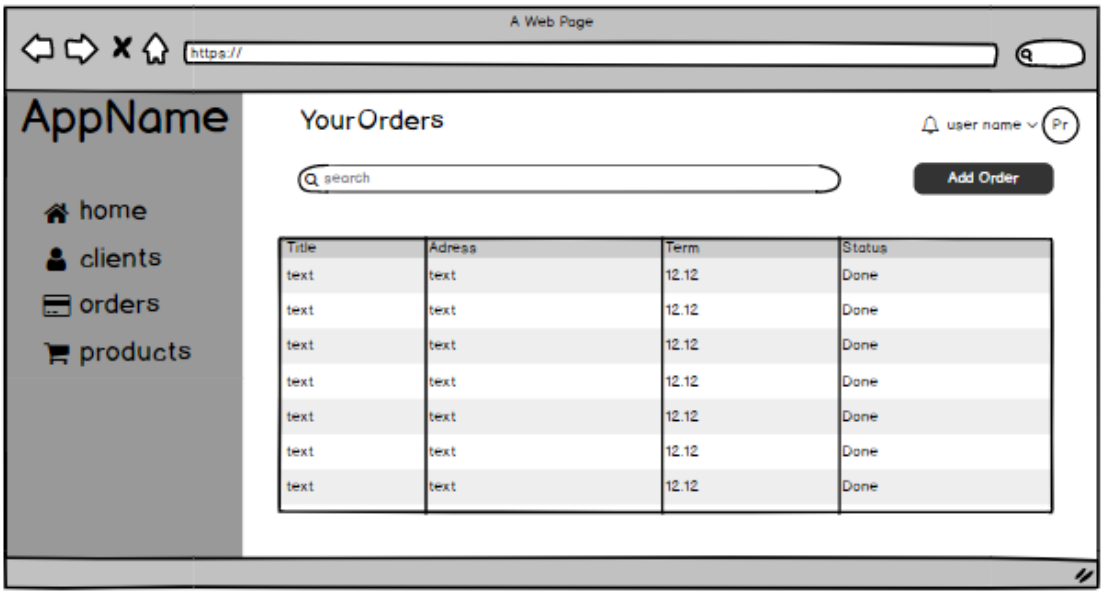

Рисунок 3.6 – Мокап сторінки з клієнтами

Вікно замовлень (рис. 3.7) складається із карток, що містять інформацію про товар, та кнопки додавання нового товару.

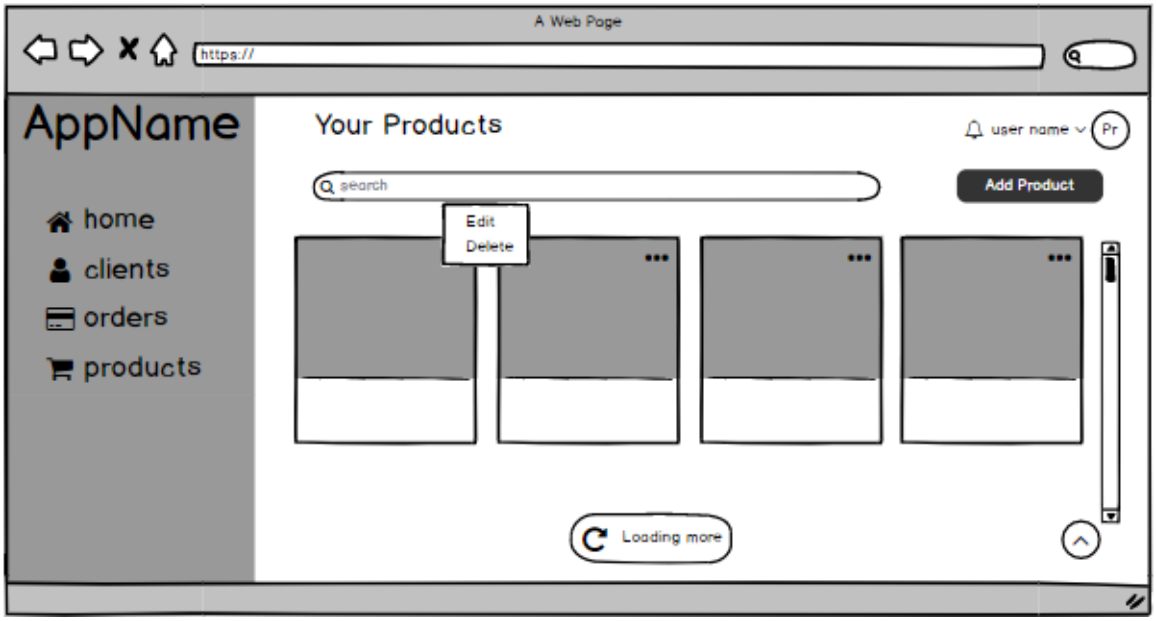

Рисунок 3.7 – Мокап сторінки з клієнтами

3.2 Створення діаграми прецедентів

UML (Unified Modeling Language) – мова графічного опису створення моделей. UML створювався для використання в процесі розробки програмного забезпечення. Головною його метою було досягнення єдиного бачення розробників і користувачів при створювані програми [16].

Створення моделей дозволяє більш наочно документувати рішення. До реалізації ідей в коді зрозуміти і пояснити іншим учасникам проекту, як буде працювати програма. А користувачам надання моделей дозволяє зрозуміти, чи відповідає заявлена робота тому, що їм дійсно потрібно.

Створити модель можна в сотні і тисячі разів швидше, ніж створити реальний прототип програми. Модель набагато легше і швидше допрацювати і змінити, якщо обговорення покаже прийняті рішення не вірними. В результаті створення моделей скорочується необхідність переробок в програмах, що

робить розробку дешевше і швидше. Використання моделей при створенні великих систем, дозволяє охопити систему одним поглядом і досягти кращого його розуміння всіма зацікавленими особами.

Таким чином, цілями аналізу і моделювання є:

- досягнення угоди між розробниками, замовниками і користувачами про те, що повинне робити ПП;
- досягнення кращого розуміння розробниками поведінки ПП;
- обмеження системної функціональності;
- створення базису для планування розробки проекту;
- визначення призначеного для користувача інтерфейсу.

Діаграма прецедентів чи варіантів використання (Use Case Diagram) визначає функціональне призначення модельованої системи або предметної області. Дана діаграма відображає безліч акторів, що взаємодіють з проектованої системою (програмним засобом) за допомогою варіантів використання. Таким чином, основними елементами діаграми варіантів використання є актор і варіант використання.

Актор – це зовнішня по відношенню до модельованої системі сутність, що взаємодіє з системою для вирішення деяких завдань. Як актор може використовуватися людина, інша система, пристрій або програмний засіб.

Варіант використання визначає деякий набір дій (операцій), які повинні бути виконані моделюється системою або програмним засобом при взаємодії з актором.

В роботі була створена діаграма прецедентів для застосунку, що розроблюється (рис. 3.8). На діаграмі зображено два актора: користувач та гість.

Гість має можливість зареєструватися та переглядати сайт.

Користувач має можливість управляти своїми замовленнями, клієнтами та товарами, авторизуватися, продивляти свій профіль та редагувати його.

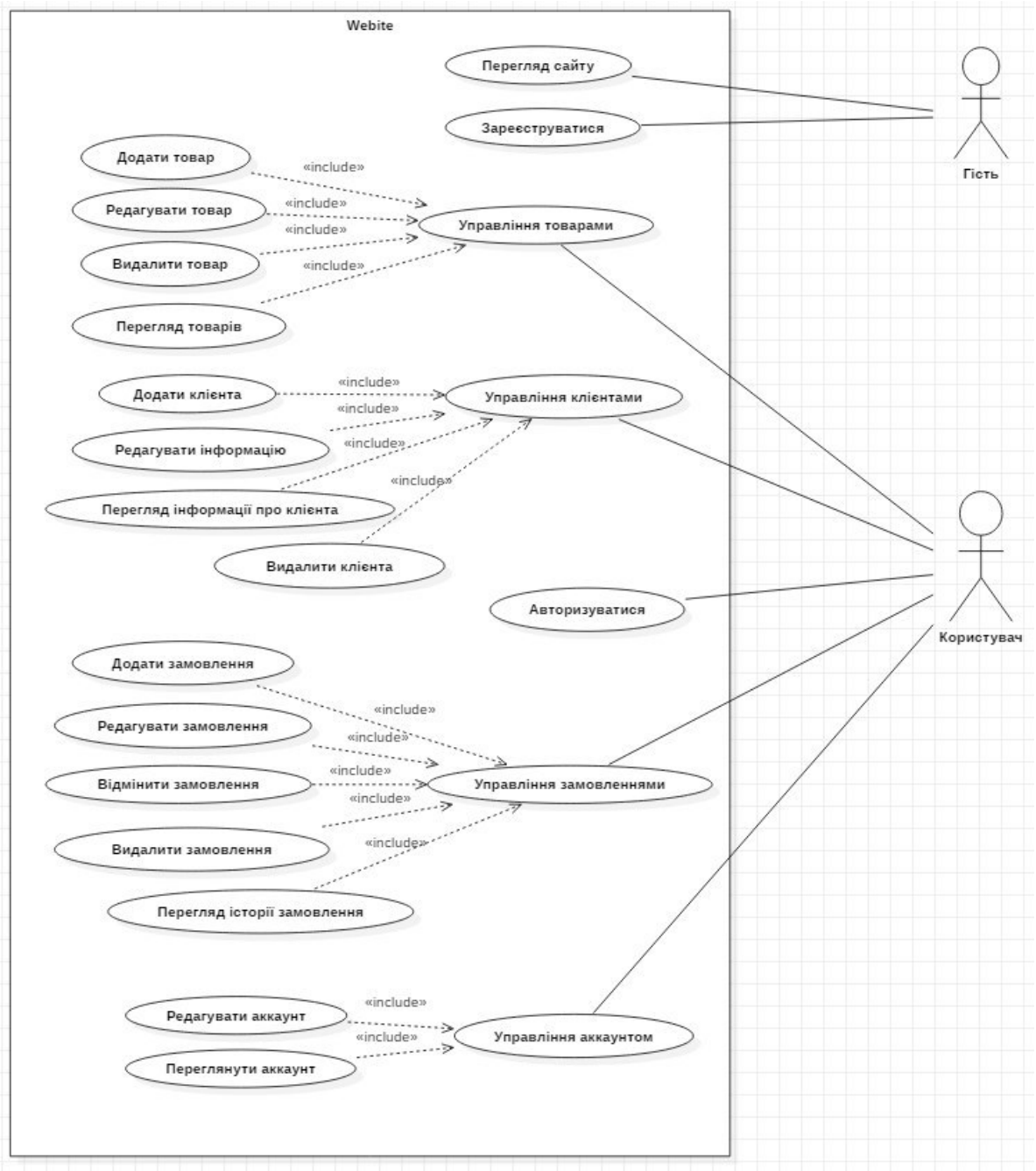

Рисунок 3.8 – Діаграма прецедентів

3.2 Створення діаграми класів

Діаграма класів використовуються при моделюванні програмних систем (ПС) найбільш часто. Вона є однією з форм статичного опису системи з точки зору її проектування, показуючи її структуру. Діаграма класів не відо-

бражує динамічну поведінку об'єктів зображених на ній класів. На діаграмах класів показуються класи, інтерфейси і відносини між ними.

Клас – це основний будівельний блок ПС. Кожен клас має назву, атрибути і операції. Клас на діаграмі показується у вигляді прямокутника, розділеного на 3 області. У верхній міститься назва класу, в середній – опис атрибутів (властивостей), в нижній – назви операцій – послуг, що надаються об'єктами цього класу.

Атрибути класу визначають склад і структуру даних, що зберігаються в об'єктах цього класу. Кожен атрибут має ім'я і тип, який визначає, які дані він представляє. При реалізації об'єкта в програмному коді для атрибутів буде виділена пам'ять, необхідна для зберігання всіх атрибутів, і кожен атрибут матиме конкретне значення в будь-який момент часу роботи програми. Об'єктів одного класу в програмі може бути як завгодно багато, всі вони мають однаковий набір атрибутів, описаний в класі, але значення атрибутів у кожного об'єкта свої і можуть змінюватися в ході виконання програми.

На діаграмах класів зазвичай показуються асоціації та узагальнення. Кожна асоціація несе інформацію про зв'язки між об'єктами всередині ПС. Найбільш часто використовуються бінарні асоціації, що зв'язують два класи. Узагальнення на діаграмах класів використовується, щоб показати зв'язок між класом-батьком і класом-нащадком. Воно вводиться на діаграму, коли виникає різновид будь-якого класу, а також в тих випадках, коли в системі виявляються кілька класів, що володіють схожим поведінкою (в цьому випадку загальні елементи поведінки виносяться на більш високий рівень, утворюючи клас-батько.

Діаграма класів створюються при логічному моделюванні ПС і служать для наступних цілей:

- для моделювання даних;.
- для уявлення архітектури ПС;
- для моделювання навігації екранів;

- для моделювання логіки програмних компонент;
- для моделювання логіки обробки даних.

Після розробки діаграми прецедентів в роботі була створена діаграма класів, яка представлена у Додатку А. На цій діаграмі можна побачити основні та допоміжні класи.

Висновок до розділу: в процесі проектування було створені діаграми: прецедентів, мокапів та класів. За допомогою них мета створення веб-застосунку стала більш прозора, та його розробка стала легшою.

4 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

4.1 Архітектура програмної системи

Під час створення сайту була використана RESTful архітектура, тобто сервер та клієнт розроблялися окремо, а взаємодія між ними відбувається завдяки використанню протоколу HTTP та ідеології REST, та API. На сервері створюються контрольні точки, які відповідають за деяку логіку, а клієнт, використовуючи дані точки, відправляє серверу запити. Далі сервер приймає запити з клієнта, опрацьовує їх (при необхідності звертається до базі даних) та відправляє відповідь клієнту. На рис.4.1 показано клієнт-серверну архітектуру.

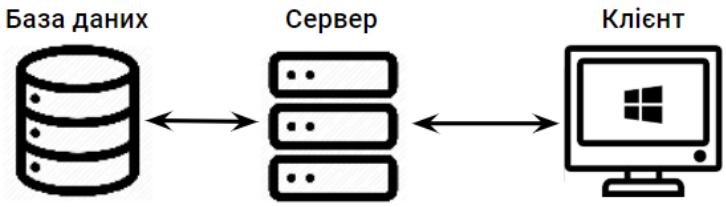

Рисунок 4.1 Клієнт-серверна взаємодія

4.2 Структура бази даних

Добре продумана база даних – це передусім набір пойменованих таблиць. Кожна з яких у свою чергу містить ряд полів, що мають визначені властивостями.

На рис. 4.2 приведена структура бази даних програмної системи. На ній зображено основні таблиці бази даних та їх поля:

- application_user – відповідає за зберігання інформації про користувачів;
- user_role – зберігає ролі користувачів (звичайний користувач, адміністратор тощо);

– product – містить інформацію про товари;
– type – зберігає тип товару;
– price – містить інформацію про ціну товару з розрахунком усіх бонусів та знижок;
– orders – зберігає інформацію про замовлення;
– status – містить інформацію про статус замовлення;
– client – зберігає інформацію про клієнтів.

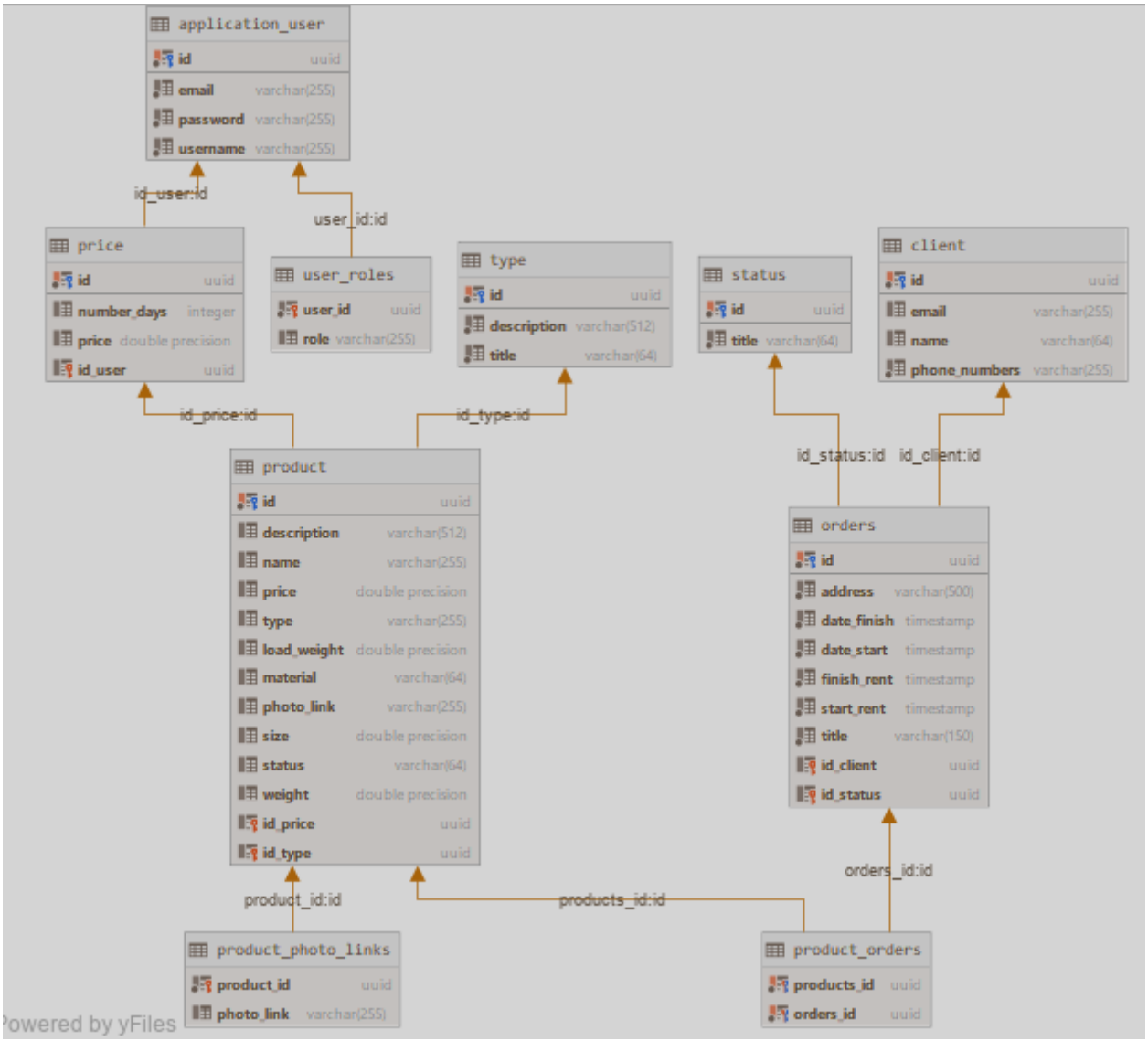

Рисунок 4.2 – Структура бази даних

4.3 Реалізація серверної частини

При реалізації серверної частини була вибрана тришарова-монолітна архітектура. Призначення шарів:

– шар репозиторія (Repository layer) – відповідає за взаємодію зі сховищем даних;
– сервісний шар (Service layer) – містить в собі основну логіку програми, також відповідає за транзакції та авторизацію
– веб-шар (Web layer) відповідає за обробку введених користувачем даних і повернення коректної відповіді.

Також веб-шар відповідає за обробку виключень, які можуть викидатися в інших шарах застосунку. Так як веб-шар є точкою входу в застосунок, він також відповідає за аутентифікацію і є першою лінією захисту програми. На рис. 4.3 показана дана модель взаємодії.

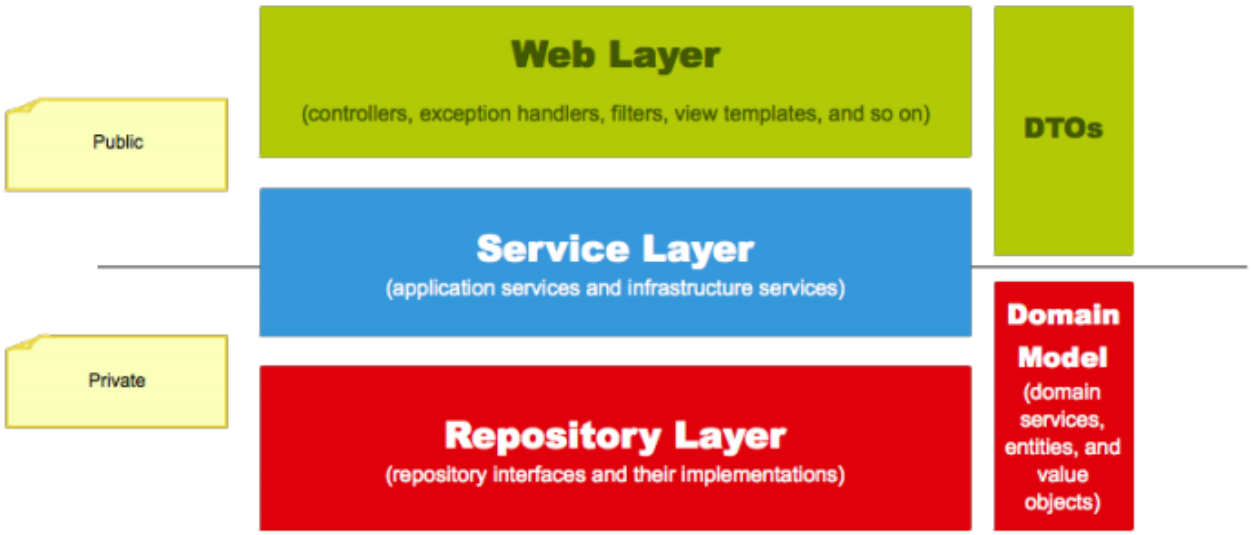

Рисунок 4.3 – Тришарова структура серверу

Отже, розглянемо реалізацію цієї структури на прикладі Price сутності. Спочатку створюємо репозиторій. В Spring Boot достатньо створити ін-

терфейс, що буде наслідувати JpaRepository інтерфейс та повісити на нього анотацію @Repository. Приклад коду нижче.

```
    @Repository
public interface PriceRepository extends JpaRepository<Price,
UUID>, JpaSpecificationExecutor<Price> {
}
```

Далі створюємо сервіс, який буде реалізовувати CRUD операції. Для цього створимо клас PriceService та повісимо на нього анотацію @Service. В самому класі визначимо поле repository та за допомогою dependency injection (DI) зв'яжемо його із своїм репозиторієм. Далі, звертаючись до репозиторію, будемо оперувати даними.

Останнім кроком буде створення контролера. Для цього створимо клас PriceController з анотацією @RestController. В цьому класі, знов ж таки, ми за допомогою DI отримаємо екземпляр класу PriceService та викликаємо його методи для подальшої обробки даних. Над кожним методом необхідно прописати контрольні точки – частина URL, на яку буде відображатися заданий метод, для того, щоб наш створений метод був викликаний, коли на сервер надійде HTTP-запит з певним URL.

4.3 Реалізація клієнтської частини

Клієнтська частина написана завдяки фреймворку Angular. Він дозволяє реалізувати патерн проектування MVC (Model-View-Controller). Структура клієнта показана на рис. 4.4.

Розглянемо структуру детальніше. В проекті є AppModule – він виступає в якості основного модуля. В SharedModule знаходяться усі допоміжні компоненти, які використовуються в інших модулях. CoreModule містить сервіси, які взаємодіють з сервером додатку, сервіси авторизації та інтернаціоналізації. Далі йде HomeModule – в ньому містяться всі сторінки веб-сайту з їх відповідними компонентами.

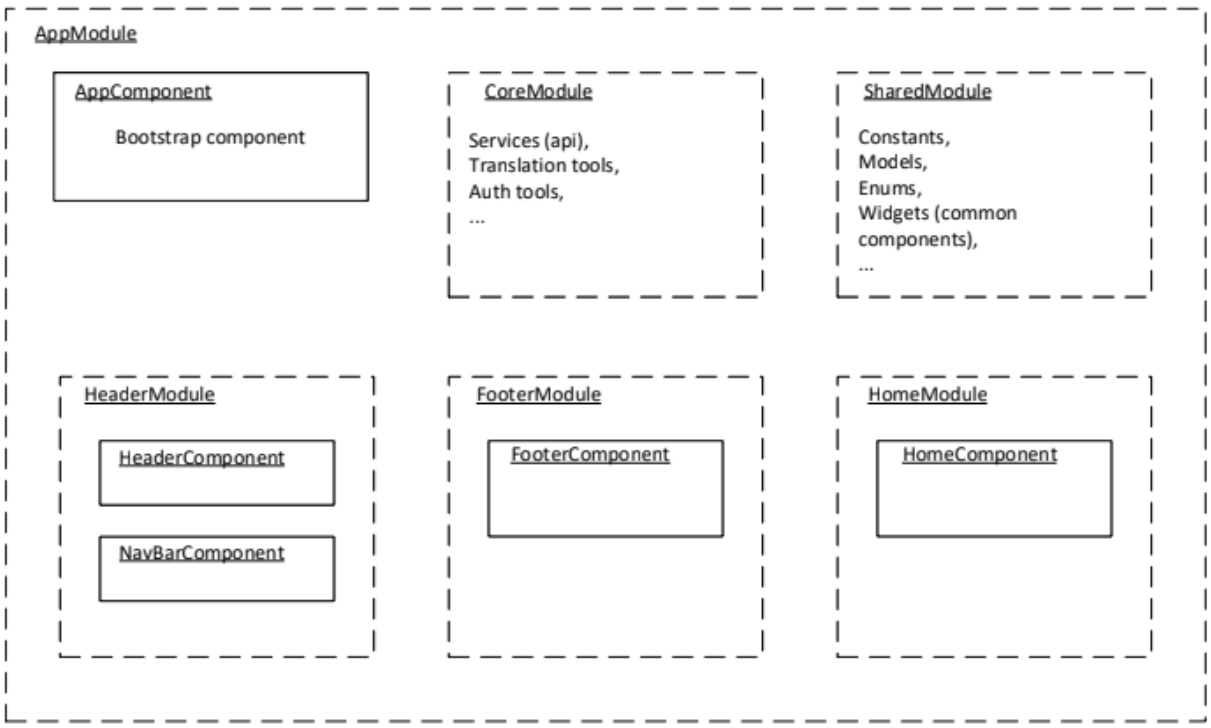

Рисунок 4.4 – Структура клієнта

На клієнті також реалізована можливість авторизації та аутентифікації. Авторизація та аутентифікація здійснюються завдяки JWT (JSON Web Token). Також на стороні клієнта використовується реактивний підхід завдяки бібліотеці RxJs.

Висновок до розділу: в даному розділі ми розглянули програмну реалізацію проекта. Детальніше зупинилися на на розробці серверної частини, бази данних та швидко переглянули реалізацію клієнта. Код програми наведено в Додатку Б.

4.4 Інструкція користувача

Робота з сайтом починається зі сторінки входу, на якій необхідно заповнити форму: логін та пароль, або перейти на сторінку реєстрації у випадку відсутності аккаунту (рис. 4.5). Перегляд сторінок можна здійснювати за допомогою браузерів: Opera, Google Chrome, Safari.

Рисунок 4.5 – Сторінка авторизації

На рис. 4.6 зображена сторінка реєстрації, яка передбачає заповнення форми, що складається з 3 полів введення.

Рисунок 4.6 – Сторінка реєстрації

На рис. 4.7 показано головну сторінку сайта. Сайт складається з лівого меню та основної частини яка відображає пункти меню на які переходить користувач. Нижче відображена сторінка звіту (головна сторінка), де користувач може коротко проглядати спільну інформацію пов'язану з його бізнесом.

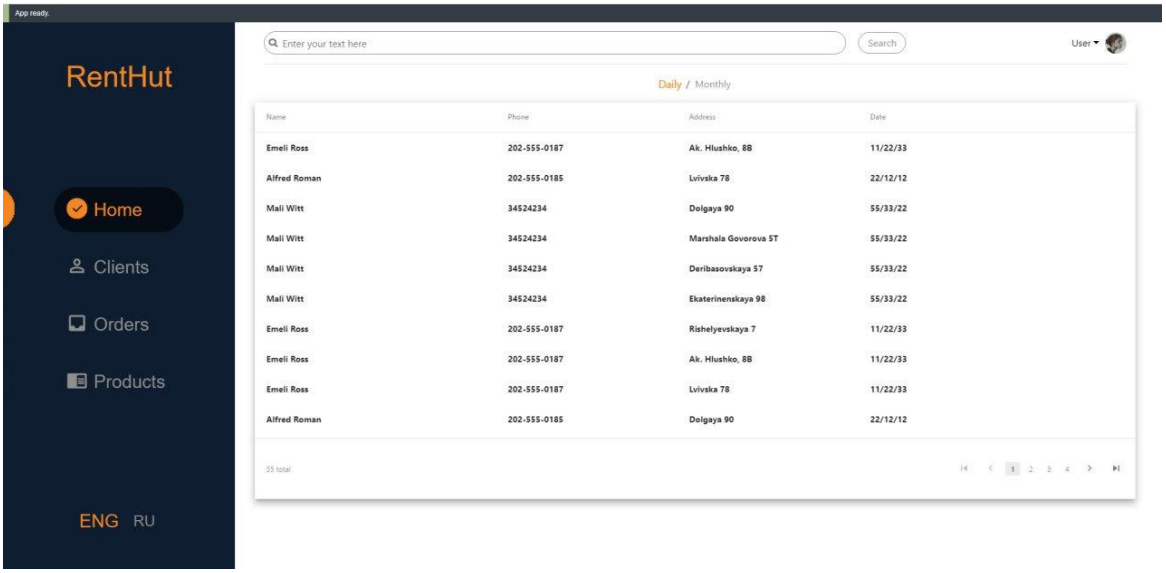

Рисунок 4.7 – Головна сторінка

Перейшовши на пункт клієнт, можна побачити усіх клієнтів які робили замовлення, та їх дані. Картку кожного клієнта можна редагувати, або взагалі стерти у випадку непотрібності (рис. 4.8).

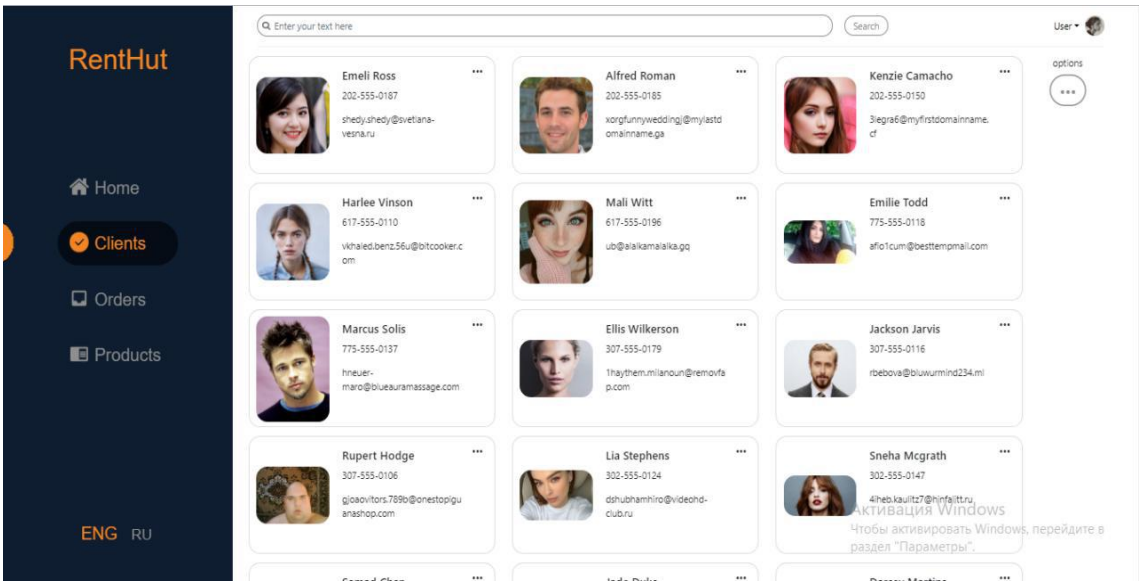

Рисунок 4.8 – Сторінка з клієнтами

Система має додаткові опції, пов’язані з пошуком інформації. Так є можливість знайти клієнтів, які мають заказ на конкретні дати з використанням календаря (рис.4.9). Також є можливість пошуку клієнтів за їх контактними даними (рис.4.10).

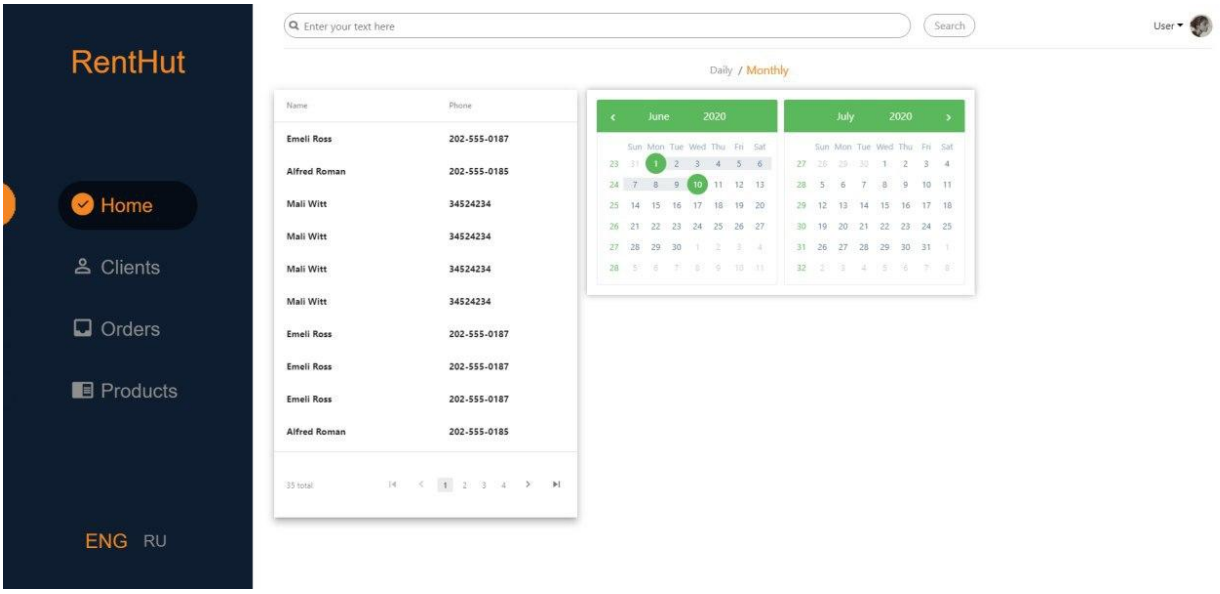

Рисунок 4.9 – Пошук клієнтів за датами заказу

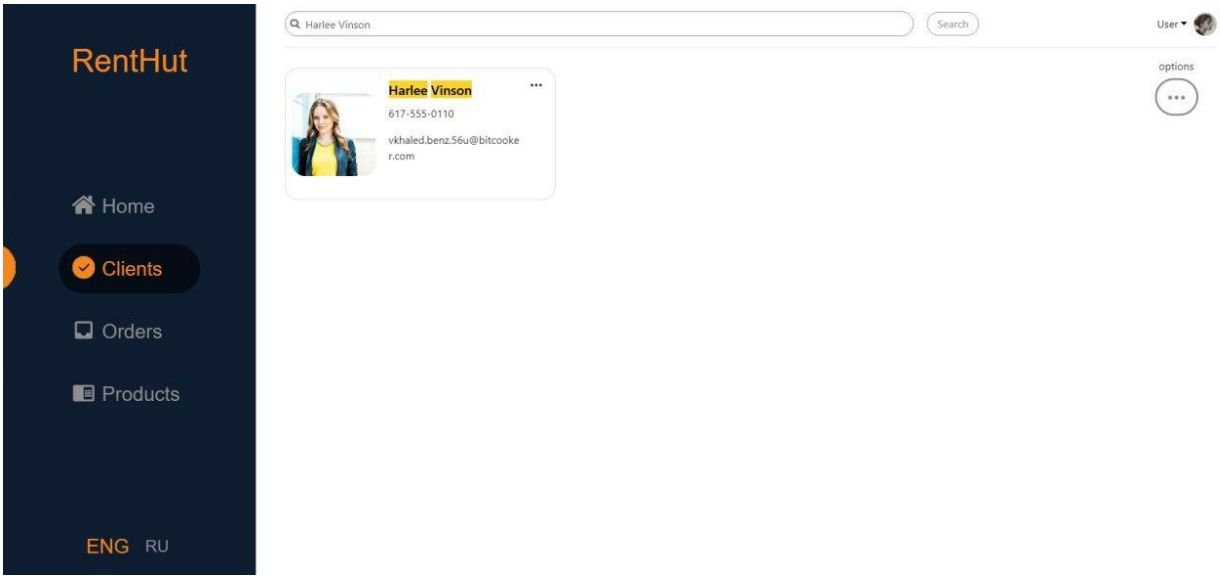

Рисунок 4.10 – Пошук клієнтів за їх контактними даними

Щоб оформити замовлення клієнта слід скористатися сторінкою "заказ", який теж можна відредагувати або видалити (рис.4.11).

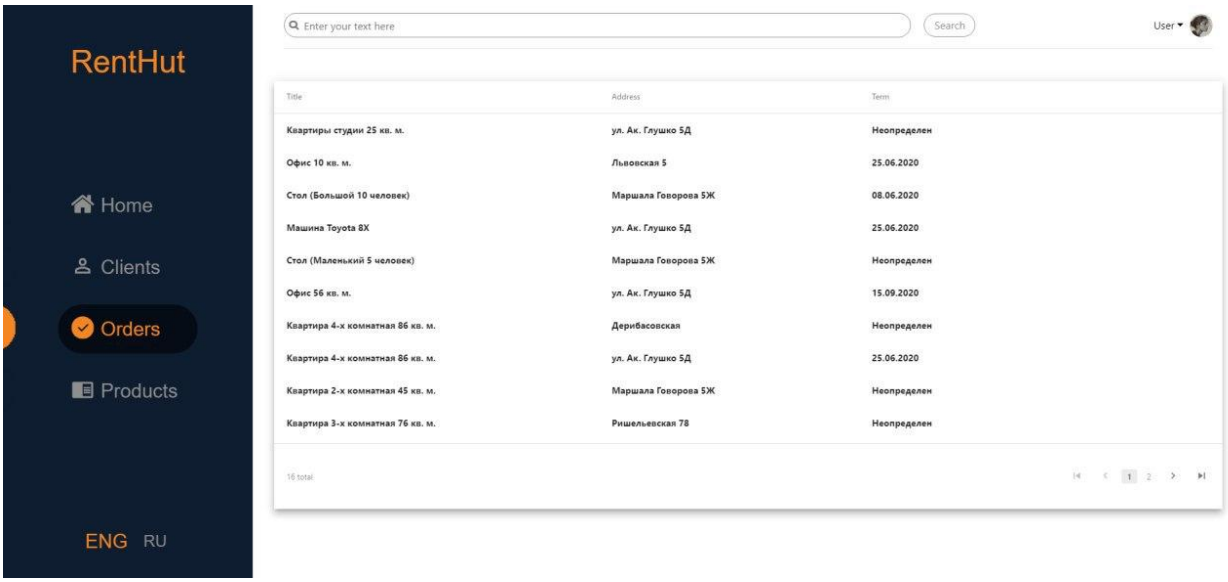

Рисунок 4.11 – Список заказів

Також на сторінці є два режиму: відображення замовлення у теперішній проміжок часу, або взагалі всієї історії замовлення які коли-небудь були.

Для того щоб на сторінці замовлень вибрати необхідний продукт, треба додати його у список продуктів, що є в наявності (рис.4.12).

Щоб це зробити потрібно скористатися сторінкою "Товари", де можна додати фото, вартість і всі необхідні характеристики продукту (рис.4.13). Картку с продуктом можна редагувати або видалити (рис.4.14).

Висновок до розділу: в даному розділі було наведено інструкцію користувача. Показано головні можливості інтерфейсу користувача, та послідовність його використання. Як можна побачити, система, що була розроблена, має простий та зручний інтерфейс і є зрозумілою для користувача без спеціальної підготовки. Використання запропонованого веб-застосунку дозволить істотно полегшити ведення власного бізнесу з прокату речей.

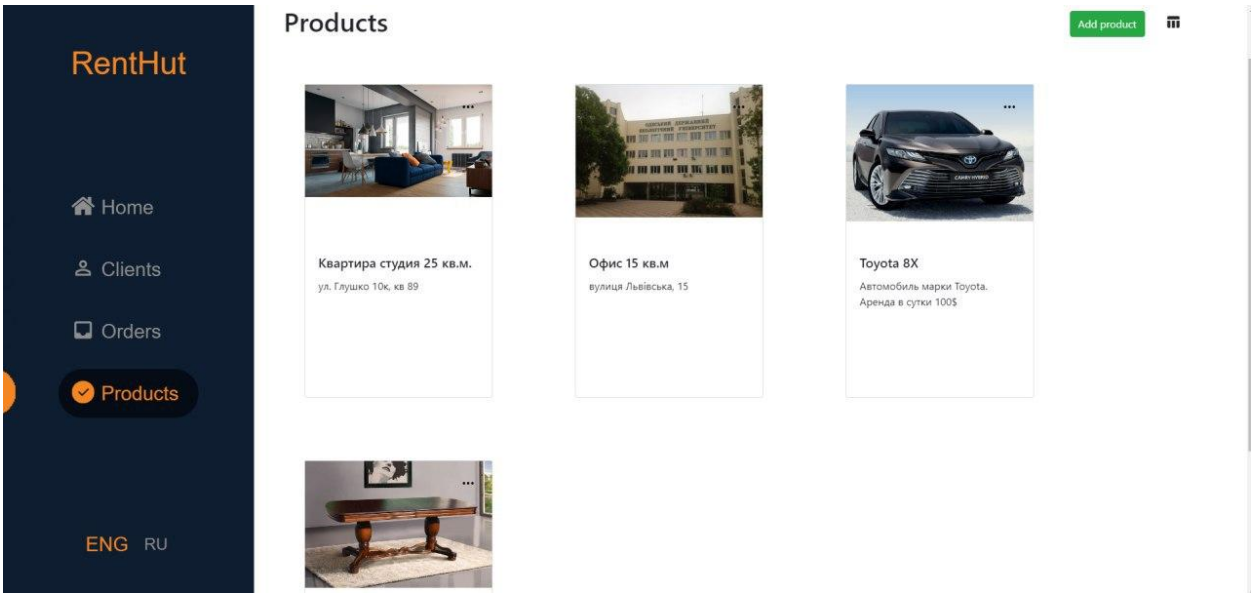

Рисунок 4.12 – Список товарів

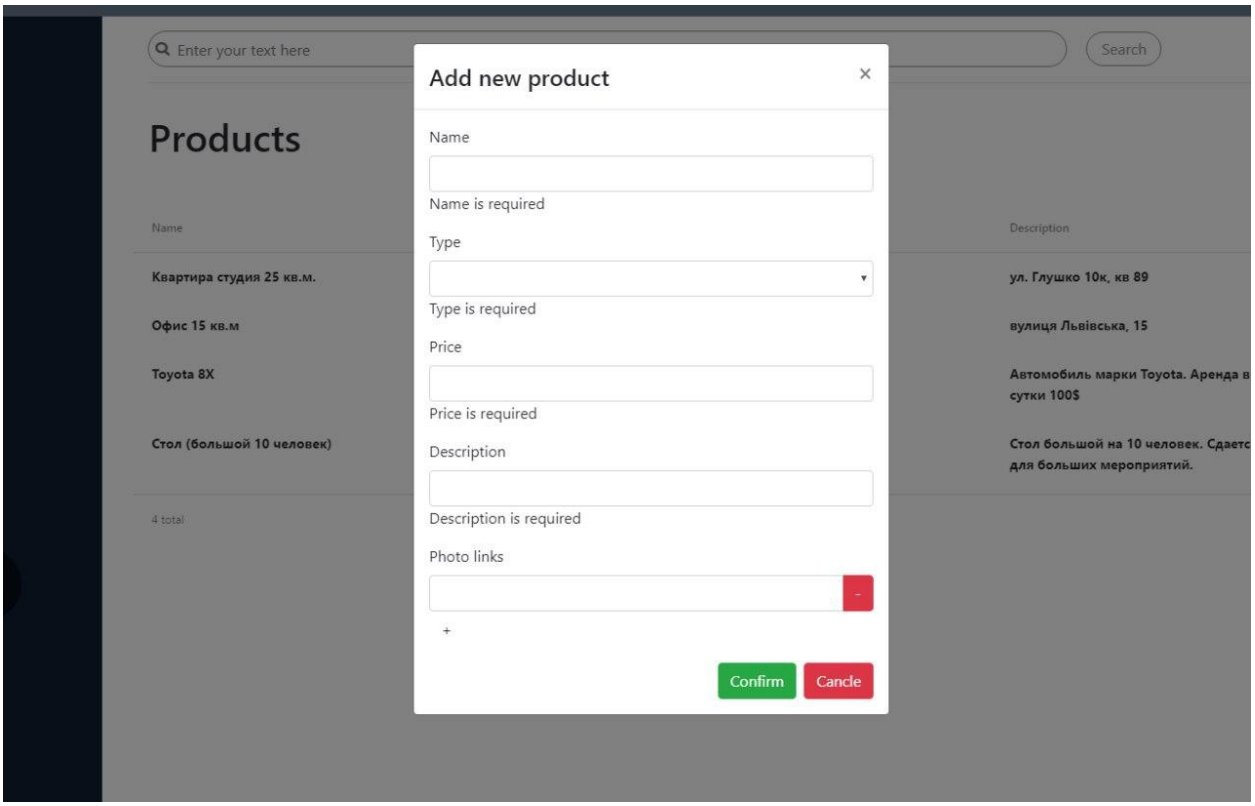

Рисунок 4.13 – Додавання нового товару

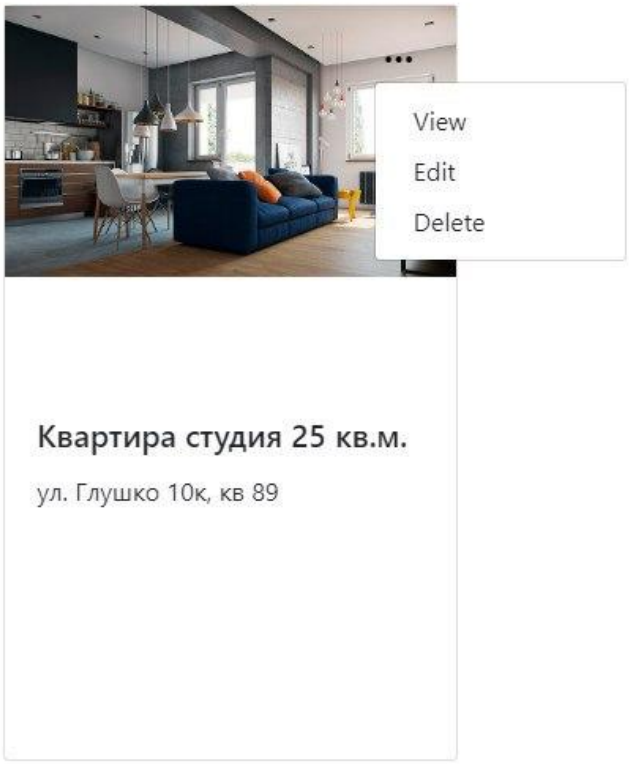

Рисунок 4.14 – Редагування товару

ВИСНОВКИ

В результаті виконання бакалаврської роботи було проведено аналіз предметної області та розглянуті існуючи аналоги запропонованої програмної системи для допомоги ведення бізнесу з прокату речей. Здійснено обґрунтований вибір програмних засобів розробки. Для клієнтської частини обрано мову програмування TypeScript, для серверної – мову Java.

Розробка клієнта велася за допомогою фреймворка Angular та Bootstrap. Розробка серверної частини велася за допомогою ряду підпроектів Spring Framework. Для швидкого, якісного та зручного написання бізнес логіки проекту використовувався Spring Boot. Spring Security використовувався для реалізації серверної безпеки, а також реєстрації та авторизації. В якості засобу автоматизації управління та складання проекту використовувався Apache Maven. В якості БД була використана об’єктно-реляційна СКБД PostgreSQL. Вона надає величезну кількість інструментів та можливостей для зберігання та обробки даних різних типів, від чисел до даних в форматах JSON та LOB. Для легких маніпуляцій з базою даних була використана технологія об’єктно-реляційного мапінгу.

На етапі проектування програмної системи було створено діаграми: прецедентів, мокапів та класів. За допомогою них мета створення вебзастосунку стала більш прозора, та його розробка стала легшою.

Також в роботі наведено інструкцію користувача. Показано головні можливості інтерфейсу користувача, та послідовність його використання.

Було проведено тестування програмної системи. Та не виявлено проблем при її використанні. Був проведений рефакторінг коду, що у разі необхідності допоможе доробити програмну систему у майбутньому.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Цивільний кодекс України. Глава 58. Найм (оренда). §2. Прокат. URL: https://zakon.rada.gov.ua/laws/show/435-15 (дата звернення 27.05.2023)
2. Торгсофт. URL: https://torgsoft.ua/soft/ (дата звернення 01.04.2023)
3. RemOnline. URL: https://remonline.ua/hire-and-rental/ (дата звернення: 01.05.2023)
4. Сервіс «АБ Система». URL https://ab.biz.ua/products/complex-configurations/rental/#Можливості (дата звернення: 01.05.2023)
5. TypeScript – Вікіпедія. (загол. з екрана). URL: https://uk.wikipedia.org/wiki/ TypeScript (дата звернення 04.01.2020).
6. Visual Studio Code – Вікіпедія. (загол. з екрана). URL: https://uk.wikipedia.org/ wiki/Visual_Studio_Code (дата звернення 04.05.2023).
7. Тарнавський Ю.А. Java-програмування: комп'ютерний практикум. Київ : КПІ ім. Ігоря Сікорського, 2021. – 95 с. URL: https://ela.kpi.ua/bitstream/123456789/41885/1/Java-programming.pdf (дата звернення 04.04.2023).
8. Intellij IDEA – Вікіпедія. (загол. з екрана). URL: https://uk.wikipedia.org/wiki/ IntelliJ_IDEA (дата звернення 04.05.2023)
9. Angular (фреймворк) – Вікіпедія. (загол. з екрана). URL: https://uk.wikipedia.org/wiki/ Angular_(фреймворк) (дата звернення 04.04.2023).
10. Bootstrap – Вікіпедія. (загол. з екрана). URL: https://uk.wikipedia.org/wiki/ Bootstrap (дата звернення 04.04.2023).
11. Spring Framework – Вікіпедія. (загол. з екрана). URL: https://uk.wikipedia.org/ wiki/Spring_Framework (дата звернення 04.04.2023).

12. Johnson R., Hoeller J., Arendsen F., Risberg T., Sampaleanu C. Professional Java™ Development with the Spring Framework. Published by Wiley Publishing, Inc. URL: https://iamgodsom.files.wordpress.com/2014/08/wrox-professional-java-development-with-the-spring-framework.pdf (дата звернення 05.05.2023).
13. Apache Maven – Вікіпедія. (загол. з екрана). URL: https://uk.wikipedia.org/ wiki/Apache_Maven (дата звернення 04.04.2023).
14. Павловський В.І., Петрашенко А.В., Победа Д.В. Бази даних та засоби управління. Практикум. Київ: КПІ ім. Ігоря Сікорського, 2021. 112 с. URL: https://ela.kpi.ua/bitstream/123456789/46193/1/Bazy_danykh_ta_zasoby_upravlinnia_Praktykum.pdf (дата звернення 04.04.2023).
15. Дудзяний І.М. Обєктно-орієнтоване моделювання програмних систем: Навчальний посібник. Львів: Видавничий центр ЛНУ імені Івана Франка, 2007. 108 с. URL: https://fis.tntu.edu.ua/ data/elibrary/3/oop_uml.pdf (дата звернення 04.04.2023).

Д О Д А Т К И

ДОДАТОК А

Діаграма класів

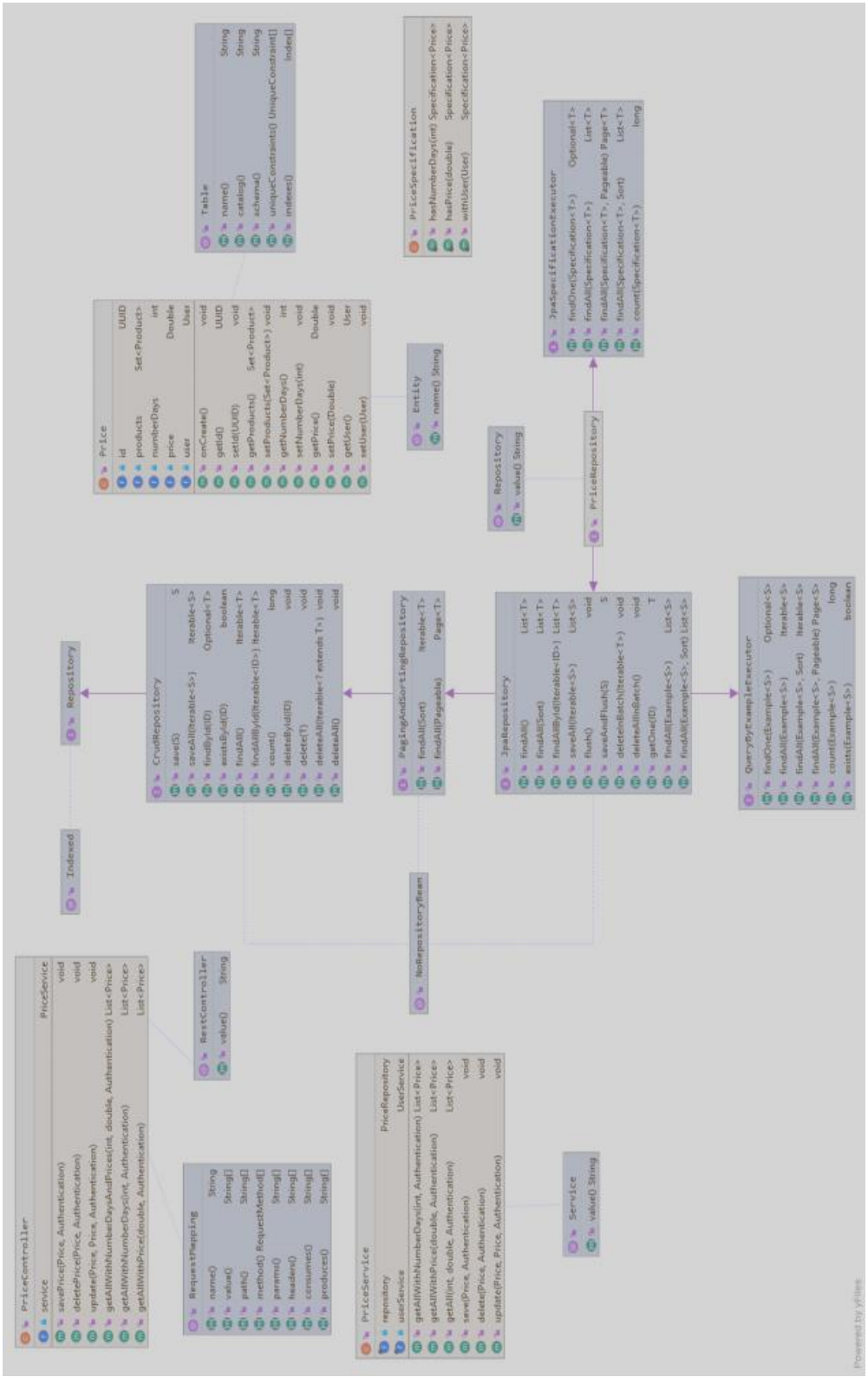

Рисунок А.1 – Діаграма класів

ДОДАТОК Б

Програмний код

AppConfig.java

```
@Component

public class AppConfig implements WebMvcConfigurer {
     @Override

     public void addCorsMappings(CorsRegistry registry) { regis-
          try.addMapping("/**")
               .allowedMethods("GET", "POST", "PUT",

"DELETE");
     }
}
```

SecurityConfig.java

```
@Configuration

@EnableWebSecurity

public class SecurityConfig extends
WebSecurityConfigurerAdapter {

     public static final long EXPIRATION_TIME =
31_536_000_000L;

     public static final String TOKEN_PREFIX = "Bearer "; public
     static final String SECRETE = "SomeSecret"; public static
     final String HEADER_STRING =

"Authorization";

     private UserDetailsServiceImpl userDetailsService; pri-
     vate BCryptPasswordEncoder encoder;

     @Autowired
     public SecurityConfig(UserDetailsServiceImpl

userDetailsService, BCryptPasswordEncoder encoder) {
          this.userDetailsService = userDetailsService;
          this.encoder = encoder;

     }

     @Override

     protected void configure(HttpSecurity http) throws Ex-
ception {
          http
```

```
                    .cors()
                    .and()
                    .csrf().disable()
                    .authorizeRequests()
                    .antMatchers( "/sign-up").permitAll()
                    .anyRequest().authenticated()
                    .and()
                    .addFilter(new
JwtAuthenticationFilter(authenticationManager()))
                    .addFilter(new
JwtAuthorizationFilter(authenticationManager()))

.sessionManagement().sessionCreationPolicy(SessionCreati
onPolicy.STATELESS);

     }

     @Override
     protected void
configure(AuthenticationManagerBuilder auth) throws
Exception {

auth.userDetailsService(userDetailsService).passwordEnco
der(encoder);

     }

}
```

UserController.java

```
@RestController
public class UserController {

     private UserService userService;
     private UserDetailsServiceImpl userDetailsService;

     @Autowired

     public   UserController(UserService   userService)   {
          this.userService = userService;

     }

     @RequestMapping(value  =  "sign-up",  method  =
RequestMethod.POST)

     public void signUp(@RequestBody User user) { Sys-
          tem.out.println(user);
          userService.save(user);

     }
     @RequestMapping(value  =  "admin",  method  =
RequestMethod.GET)
     public String test() {
          return "Wellcome, admin";
```

```
	}

	@RequestMapping(value = "user", method =
RequestMethod.GET)

	public User findByUsername(@RequestParam("username") String
username) {

		return userService.findByUsername(username);
	}
}
```

UserDto.java

```
@JsonIgnoreProperties(ignoreUnknown = true)
public class UserDTO {
	private UUID id;
	private String username;
	private String password;
	private String email;
	private Set<String> roles;

	public User toUser(){

		User user = new User();
		user.setId(id);
		user.setUsername(username);
		user.setPassword(password);
		user.setEmail(email);
		user.setRoles(roles);

		return user;
	}

	public static UserDTO fromUser(User user) {
		UserDTO userDTO = new UserDTO();
		userDTO.setId(user.getId());
		userDTO.setUsername(user.getUsername());
		userDTO.setPassword(user.getPassword());
		userDTO.setEmail(user.getEmail());
		userDTO.setRoles(user.getRoles());
		return userDTO;
	}

	public UserDTO() {
	}

	public UUID getId() {

		return id;
	}

	public void setId(UUID id) {
		this.id = id;
	}
```

```
    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;

    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getEmail() {
        return email;

    }

    public void setEmail(String email) {
        this.email = email;

    }

    public Set<String> getRoles() {
        return roles;

    }

    public void setRoles(Set<String> roles) {
        this.roles = roles;
    }
}
```

User.java

```
@Entity
@Table(name = "application_user")
public class User {
    @Id
    private UUID id;

    @NotNull
    @Column(name = "username")

    private String username;

    @NotNull
    @Column(name = "password")
    private String password;

    @Email
    @NotNull
```

```
    @Column(name = "email")
    private String email;

    @NotNull
    @Column(name = "role")

    @ElementCollection(targetClass = String.class, fetch
= FetchType.EAGER)
    private Set<String> roles;

    @Column(name = "id_product")
    @OneToMany(mappedBy = "user")
    @JsonManagedReference
    private Set<Product> products;

    public User() {
    }

    public  User(UUID  id,  @NotNull  String  username,
                  @NotNull String password,

                  @Email @NotNull String email,
                  @NotNull Set<String > roles) {
        this.id = id;
        this.username = username;
        this.password = password;
        this.email = email;
        this.roles = roles;
    }

    @PrePersist
    public void onCreate() {

        this.id = UUID.randomUUID();
    }

    public UUID getId() {
        return id;
    }

    public void setId(UUID id) {
        this.id = id;
    }

    public String getUsername() {
        return username;

    }

    public  void  setUsername(String  username)  {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }
```

```
    public void setPassword(String password) {
        this.password = password;

    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;

    }

    public Set<String> getRoles() {
        return roles;

    }

    public void setRoles(Set<String> roles) {
        this.roles = roles;

    }

    public Set<Product> getProducts() {
        return products;
    }

    public void setProducts(Set<Product> products) {
        this.products = products;
    }
}
```

Product.java

```
@Entity
@Table(name = "products")
public class Product {
    @Id
    private UUID id;

    @Column(name = "name")
    private String name;

    @Column(name = "type")
    private String type;



    @Column(name = "price")
    private double price;

    @Size(max = 512)
    @Column(name = "description")
    private String description;
```

```
@Column(name = "photo_link")

@ElementCollection
private List<String> photoLinks;

@JoinColumn(name = "id_user")
@ManyToOne(fetch = FetchType.EAGER)
@JsonBackReference

private User user;

public Product() { }

@PrePersist
public void onCreate() {
     this.id = UUID.randomUUID();

}

public UUID getId() {
     return id;
}

public void setId(UUID id) {
     this.id = id;
}

public String getName() {
     return name;
}

public void setName(String name) {
     this.name = name;
}

public String getType() {
     return type;
}

public void setType(String type) {
     this.type = type;

}

public double getPrice() {
     return price;
}

public void setPrice(double price) {
     this.price = price;
}

public String getDescription() {
     return description;
}
```

```
    public void setDescription(String description) {
        this.description = description;
    }

    public List<String> getPhotoLinks() {
        return photoLinks;

    }

    public void setPhotoLinks(List<String> photoLinks) {
        this.photoLinks = photoLinks;
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

}
```

Role.java

```
public final class Role {
    public static final String USER = "USER";

    public static final String ADMIN = "ADMIN";
}
```

ProductSpecification.java

```
public class ProductSpecification {
    public static Specification<Product> withId(UUID id)
{

        return (root, criteriaQuery, criteriaBuilder) ->
criteriaBuilder.equal(root.get("id"), id);
    }

    public static Specification<Product> hasType(String type)
{

        return (root, criteriaQuery, criteriaBuilder) ->
criteriaBuilder.equal(root.get("type"), type);
    }

    public static Specification<Product>
withPrice(Double price) {

        return (root, criteriaQuery, criteriaBuilder) ->
criteriaBuilder.equal(root.get("price"), price);
    }
```

```
     public static Specification<Product> withUser(User user)
{

          return (root, criteriaQuery, criteriaBuilder) ->
criteriaBuilder.equal(root.get("user"), user);
     }
}
```

ProductRepository.java

```
public interface ProductRepository extends
JpaRepository<Product,                  UUID>,
JpaSpecificationExecutor<Product> { }
```

UserRepository.java

```
@Repository

public interface UserRepository extends
JpaRepository<User, UUID> {
     User findByUsername(String username);
}
```

JwtAuthenticationFilter.java

```
public class JwtAuthenticationFilter extends
UsernamePasswordAuthenticationFilter {
     private AuthenticationManager authenticationManager;

     public        JwtAuthenticationFilter(AuthenticationManager
authenticationManager) {

          this.authenticationManager  =
authenticationManager;

          //setFilterProcessesUrl("/login");
     }

     @Override
     public Authentication
attemptAuthentication(HttpServletRequest request,
HttpServletResponse response) throws
AuthenticationException {
          try {
               UserDTO credential = new
ObjectMapper().readValue(request.getInputStream(),

UserDTO.class);

                return authenticationManager.authenticate( new
                          UsernamePasswordAuthenticationToken(
                            credential.getUsername(),
                            credential.getPassword()
                  )
               );
```

```
            } catch (IOException e) {
                throw new RuntimeException(e);
            }

      }

      @Override
      protected void
successfulAuthentication(HttpServletRequest request,
HttpServletResponse response,
                                                        FilterChain

chain,   Authentication   authResult)   throws   IOException,
ServletException {
            User user = ((User) authResult.getPrincipal());

            String token = JWT.create()
                        .withSubject(user.getUsername())
                        .withArrayClaim("roles",

user.getAuthorities().stream()

.map(GrantedAuthority::getAuthority)
                                    .toArray(String[]::new))
                        .withExpiresAt(new
Date(System.currentTimeMillis() +
SecurityConfig.EXPIRATION_TIME))

   .sign(Algorithm.HMAC256(SecurityConfig.SECRETE.getBytes( )));
//response.addHeader(SecurityConfig.HEADER_STRING,
SecurityConfig.TOKEN_PREFIX + token);

            response.setContentType("application/json");  re-
            sponse.setCharacterEncoding("UTF-8");         re-
            sponse.getWriter().write(new

   JSONObject().put("token", token).toString());
            response.getWriter().flush();

            response.getWriter().close();
      }
}
```

JwtAuthorizationFilter.java

```
public   class   JwtAuthorizationFilter   extends
BasicAuthenticationFilter {

      public        JwtAuthorizationFilter(AuthenticationManager
authenticationManager) {

            super(authenticationManager);
      }

      @Override
```

```
     protected void doFilterInternal(HttpServletRequest re-
quest, HttpServletResponse response, FilterChain chain)
throws IOException, ServletException {

          String header =
request.getHeader(SecurityConfig.HEADER_STRING);

           if(header == null ||
!header.startsWith(SecurityConfig.TOKEN_PREFIX)) {

                chain.doFilter(request,    response);
                return;

           }

           UsernamePasswordAuthenticationToken
authentication = getAuthentication(request);



SecurityContextHolder.getContext().setAuthentication(aut
hentication);
           chain.doFilter(request, response);
     }

     private      UsernamePasswordAuthenticationToken
getAuthentication(HttpServletRequest request) {
          String token =
request.getHeader(SecurityConfig.HEADER_STRING);

          if (token != null &&
token.startsWith(SecurityConfig.TOKEN_PREFIX)) {
               try {
                    DecodedJWT jwt =

JWT.require(Algorithm.HMAC256(SecurityConfig.SECRETE.get
Bytes()))
                              .build()

.verify(token.replace(SecurityConfig.TOKEN_PREFIX, ""));

                    String user = jwt.getSubject();

                    List<SimpleGrantedAuthority> authorities
=

jwt.getClaims().get("roles").asList(SimpleGrantedAuthori
ty.class);

                    if (user != null) {
                         return new
UsernamePasswordAuthenticationToken(user, null,
authorities);
                    }
               } catch (Exception e) {
                    throw new RuntimeException(e);
               }
```

```
            }
            return null;
      }
}
```

UserService.java

```
@Service
public class UserService {

      private   UserRepository   userRepository;
      private BCryptPasswordEncoder encoder;

      @Autowired

      public    UserService(UserRepository    userRepository,
BCryptPasswordEncoder encoder) {
            this.userRepository = userRepository;
            this.encoder = encoder;
      }

      public  User  findByUsername(String  username)  {  return
            userRepository.findByUsername(username);

      }

      public void save(User user) {

user.setPassword(encoder.encode(user.getPassword()));
            //user.setRoles(new
HashSet<>(Arrays.asList(Role.USER, Role.ADMIN)));
            user.setRoles(new
HashSet<>(Collections.singletonList(Role.USER)));
            userRepository.save(user);
      }
}
```

UserDetailService.java

```
@Service

public   class   UserDetailsServiceImpl   implements
UserDetailsService {

      private UserService userService;

      @Autowired

      public     UserDetailsServiceImpl(UserService
userService) {

            this.userService = userService;
      }

      @Override
```

```
     public    UserDetails    loadUserByUsername(String
username) throws UsernameNotFoundException {

          User user =
userService.findByUsername(username);

          if (user == null)

               throw    new    UsernameNotFoundException("Used
with username: " + username + "not found");

          return new
org.springframework.security.core.userdetails.User(user.
getUsername(),
                    user.getPassword(),
                    user.getRoles().stream()

.map(SimpleGrantedAuthority::new)
                              .collect(Collectors.toSet())
          );

     }
}
```

ProductService.java

```
@Service

public class ProductService {

     private   final   ProductRepository   repository;
     private final UserService userService;

     @Autowired

     public    ProductService(ProductRepository    repository,
UserService userService) {

          this.repository   =   repository;
          this.userService = userService;
     }

     public   List<Product>   getAll(Authentication
authResult) {

          String username = (String)
authResult.getPrincipal();
          User user =
userService.findByUsername(username);
          return
repository.findAll(where(withUser(user)));
     }

     public Optional<Product> getOneById(UUID id, Au-
thentication authResult) {

          String username = (String)
```

```
authResult.getPrincipal();

          User user =
userService.findByUsername(username);
          return
repository.findOne(where(withId(id)).and(withUser(user))
);
     }
     public List<Product> getAllWithPrice(double price, Au-
thentication authResult) {
          String username = (String)
authResult.getPrincipal();
          User user =
userService.findByUsername(username);

          return
repository.findAll(where(withPrice(price)).and(withUser(
user)));
     }

     public List<Product> getAllWithType(String type, Au-
thentication authResult) {

          String username = (String)
authResult.getPrincipal();
          User user =
userService.findByUsername(username);
          return
repository.findAll(where(hasType(type)).and(withUser(use

r)));
     }

     public    List<Product>    getAllWithPriceAndType(double
price, String type, Authentication authResult) {

          String username = (String)
authResult.getPrincipal();
          User user =
userService.findByUsername(username);
          return
repository.findAll(where(withPrice(price)).and(hasType(t
ype)).and(withUser(user)));

     }

     @Transactional

     public void save(Product product, Authentication
authResult) {
          String username = (String)
authResult.getPrincipal();
          User user =
userService.findByUsername(username);
          product.setUser(user);
          repository.save(product);
```

```
    }
    @Transactional

    public void updateProduct(Product productFromDB, Prod-
uct newProduct, Authentication authResult) {

        String username = (String)
authResult.getPrincipal();
        User user =

userService.findByUsername(username);

        if
(Objects.equals(productFromDB.getUser().getUsername(),
user.getUsername())) {

            BeanUtils.copyProperties(newProduct,
productFromDB, "id", "user");

            repository.save(productFromDB);
        }
    }

    @Transactional

    public void delete(Product product, Authentication
authResult) {
        String username = (String)
authResult.getPrincipal();
        User user =
userService.findByUsername(username);

        if
(Objects.equals(product.getUser().getUsername(),
user.getUsername())) {
            repository.delete(product);
        }
    }
}
```