МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

Кваліфікаційна робота бакалавра

на тему: Розробка веб-сервісу для лізингу медичного обладнання на базі технологій Django DRF та Vue.js

Виконав студент групи К-18
спеціальності 122 «Комп'ютерні науки»
Дударев Віталій Ігорович

Керівник к.геогр.н., доцент
Кузніченко Світлана Дмитрівна

Консультант _____

Рецензент к.техн.н., доцент
Гнатовська Ганна Арнольдівна

Одеса 2022

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

Впровадження будь-яких IT технологій, полегшують процеси, які раніше робилися вручну або неефективно.

У житті трапляються різні ситуації, одна з таких пов'язана зі здоров'ям. Буває так, що людина хворіє, але може перебуває вдома і при цьому не мати спеціального обладнання для цього.

У всіх цих випадках купувати обладнання не потрібно: крім витрат на покупку, доведеться ще витрачатися на обслуговування, замість цього можна взяти обладнання на спеціальній майданчик для оренди у людей, які займаються цим. Тим самим це дозволить скоротити не тільки грошові витрати і обслуговуванню, а також вирішить проблему пов'язана з зберіганням даного обладнання після одужання оскільки, взяте в оренду обладнання можна буде повернути в будь-який час. Таким чином розборки в даній сфері є необхідними.

Метою кваліфікаційної роботи бакалавра є розробка веб-сервіс для лізингу медичного обладнання. Цей веб-сервіс допоможе людям, у яких є медичне обладнання, заробити на ньому та успішно вести свій бізнес, а також допоможе людям, які потребують медичне обладнання, знайти його швидко та без зайвих
клопіт.

Для досягнення мети необхідно вирішити ряд завдань:

- Проаналізувати аналогічні системи для лізингу та виявити функціонал, який буде доречно запозичити;
- Вибрати інструменти та технології для реалізації;
- Спроектувати систему;
- Реалізувати систему;
- Провести тестування системи.

Структура кваліфікаційної роботи складається з вступу, трьох розділів, висновків, переліку посилань на 22 найменування. Повний обсяг роботи становить 70 сторінок і містить 26 рисунків та 3 таблиці.

1 ОГЛЯД ІСНУЮЧИХ СИСТЕМ-АНАЛОГІВ ДЛЯ ЛІЗИНГА МЕДИЧНОГО ОБЛАДНАННЯ

1.1 Аналіз предметної області

Ми живемо у світі де кожен день це несподіванка. Інколи трапляються неприємні, а можливо навіть плачевні випадки. Одними з таких є завдання шкоди здоров'ю людини. Трапляються випадки, які обмежують пересування людини або завдають незручності при яких особі потрібне спеціальне обладнання. Зазвичай таке вузькоспеціалізоване обладнання необхідно на невеличкий період, а також воно коштує великих грошей. Тому його придбання недоцільне. Більш ефективний спосіб – взяти обладнання в лізинг.

Розглянемо іншу ситуацію. Людина, яка з певних причин в минулому придбала медичне обладнання, але більше його не використовує. Зазвичай воно знаходиться у гаражі та займає простір. Можливо ця персона хоче повернути вартість коштів, які вона витратила на придбання обладнання. Гарним варіантом буде здавати це обладнання в оренду людям, які його потребують. Таким чином людина виступає у ролі орендодавця. У орендодавців може виникнути інша проблема – це проблема при веденні свого бізнесу. Тому було вирішено в межах кваліфікаційної роботи бакалавра розробити веб-сервіс, який вирішить вище описані проблеми.

Даний проект призначений для автоматизації бізнесу з оренди медичного обладнання, який передбачає веб-сервіс, де може зареєструватися користувач, який у свою чергу може надавати обладнання для лізингу. Також проект має за мету полегшити знаходження медичного обладнання для лізингу звичайним користувачам, які в ньому потребуються. Таким чином цей проект можна вважати мостом, який сполучує підприємця та клієнта та задовольняє їх потреби.

1.2 Аналіз існуючих рішень та аналогів

У нас час бізнес здачі в оренду або прокат різних речей набирає все більше обертів. Тому було вирішено створити веб-додаток, який би допоміг людям у цій справі. Але для початку необхідно провести порівняння аналогів, щоб підкреслити недоліки та удосконалити даний сервіс.

Для порівняння були обрані наступні кандидати: EasyPro, Rent in Hand та Rentman.

1.2.1 EasyPro

EasyPro – це сервіс, що допоможе знайти для лізингу обладнання різного типу. EasyPro представлено у вигляді веб-сайту, який має зрозумілу структуру. Цей сервіс підходить для людей, які хочу взяти обладнання в лізинг. Сайт має простий та зручний інтерфейс. Серед переваг можна виділити підтвердження замовлення та автоматичний підрахунок цін з урахуванням поточних знижок або акцій. Незважаючи на ці плюси, сервіс має жахливий каталог обладнання та занадто простий пошук. Користуватися сервісом можна безкоштовно. На рис. 1.1 показано інтерфейс EasyPro.

1.2.2 Rent in Hand

Rent in Hand – це програма для автоматизації лізингу різних типів прокатного бізнесу, які займаються лізингом та орендою такого інвентаря, як: прокат велосипедів, автівок або будівельної техніки. Цей сервіс підходить лише для орендодавців. Даний сервіс представлений у вигляді веб-сайту з більш-менш сучасним дизайном. Але, незважаючи на це, він має незручну структуру сайту – не завжди можна зрозуміти, що робить та чи інша кнопка. Також сайт має проблеми з оптимізацією, деякі сторінки дуже повільно завантажуються. У свою чергу сервіс має зручний каталог обладнання, e-mail розсилку та систему нагадувань. Зважливим аспектом виступає те, що даний

сервіс не є безкоштовним. На рисунку 1.2 зображено інтерфейс веб-сайту Rent in Hand.

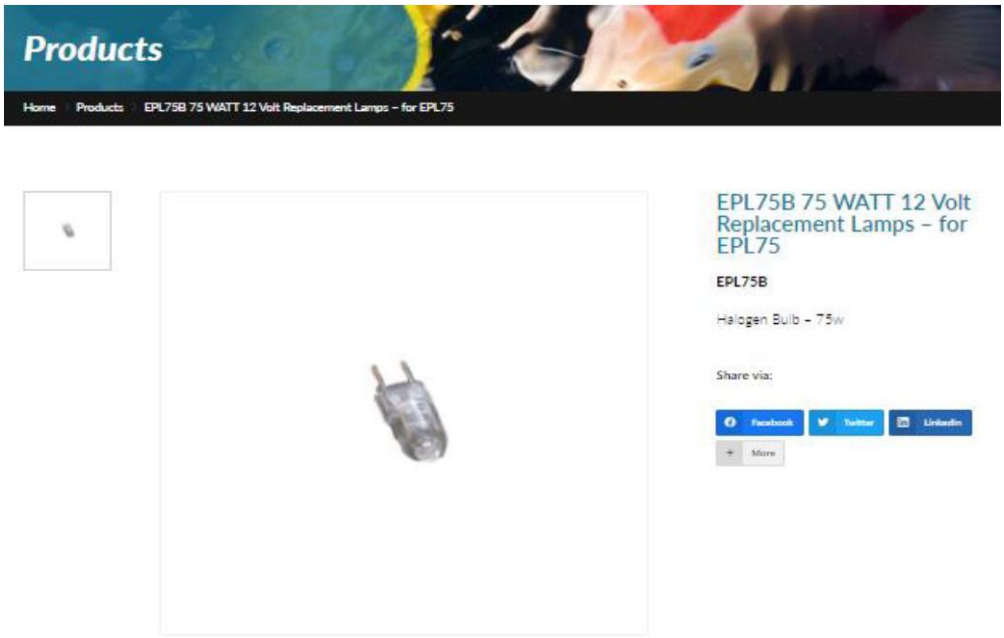


Рисунок 1.1 – Інтерфейс веб-сайту EasyPro

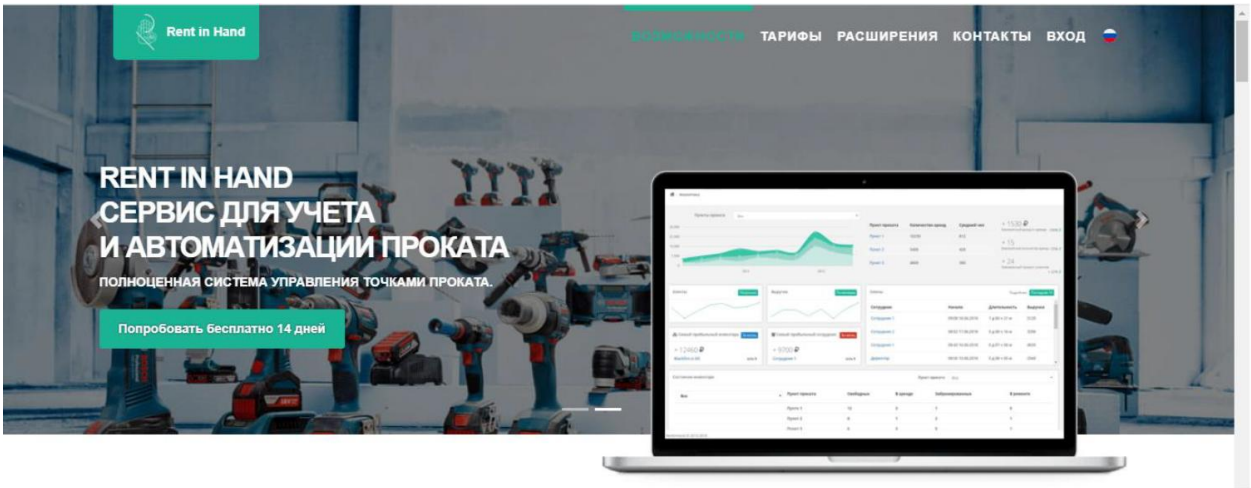


Рисунок 1.2 – Інтерфейс веб-сайту Rent in Hand

1.2.3 Rentman

Rentman – це програмне забезпечення призначене для організації процесу оренди у сфері подій, щоб вести більш ефективно бізнес. Він підходить лише орендодавцям. Сервіс представлено у вигляді веб-сайту з інтуїтивно зрозумілим інтерфейсом. Як і Rent in Hand, веб-сайт має e-mail розсилку, систему нагадувань та за його використання необхідно платити. Однак Rentman має гнучкий підрахунок цін з урахуванням усіх знижок та акцій. Rentman не може похвалитися зручним каталогом, на відміну від Rent in Hand. Також сервіс не має швидкого пошуку по обраним датам. На рис. 1.3 зображено інтерфейс веб-сайту Rentman.

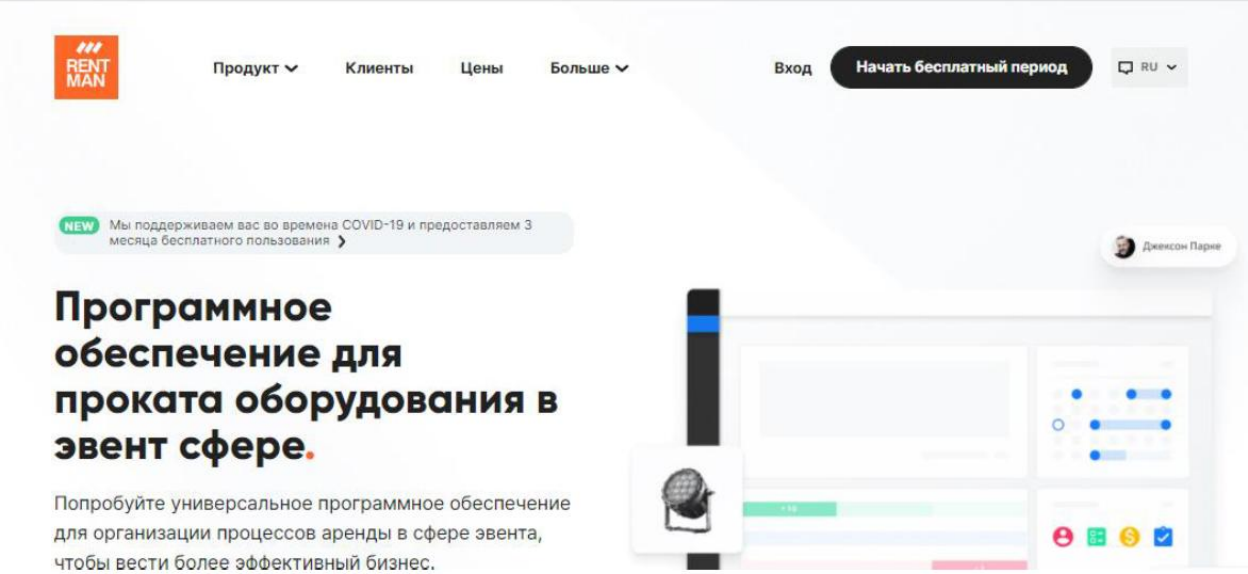


Рисунок 1.3 – Інтерфейс веб-сайту Rentman

1.2.4 Висновки щодо порівняння аналогів

Порівнявши можливі аналоги з нашим додатком, був переведений перелік переваг та недоліків систем, що допомогло удосконалити даний додаток.

У табл. 1.1, записані всі загальні критерії аналізу аналогів та проведено їх порівняння.

Таблиця 1.1 – Порівняння аналогів

Опис функціоналу	EasyPro	Rent in Hand	Rentman	Наш додаток
Простота та зручність використання	+	–	+	+
Гнучкий підрахунок цін	+	–	+	+
E-mail розсилка	–	+	+	+
Можливість бути і клієнтом і орендодавцем	–	–	–	+
Підтвердження замовлення	+	–	+	+
Нагадування	+	+	+	+
Швидкий пошук товарів на обрані дати	–	+	–	+
Зручний каталог обладнання	–	+	–	+
Безкоштовний	+	–	–	+

По таблиці видно, що основними недоліками аналогів є односторонність системи, тобто особа не може бути і клієнтом, і орендодавцем. Також недоліками деяких систем є відсутність підтвердження замовлення, незручність вибору обладнання та платне використання.

1.3 Вибір архітектурного шаблона та шаблонів проектування

Більшість архітектур побудовані на основі систем, які використовують схожі між собою набори інтересів. Цю схожість можна визначити як архітектурний стиль, який у свою чергу можна розглядати як особливий вид шаблонів. Цей вид шаблонів зазвичай носить назву архітектурні шаблони. Вони часто бувають складними та складеними. Конкретна інформаційна сис-

тема може демонструвати більш ніж один архітектурний шаблон. Дамо визначення терміну архітектурний шаблон.

Архітектурний шаблон – це шаблони програмного забезпечення (ПЗ), що являють собою звіт «належних практик» вирішення архітектурних проблем розробки програмного забезпечення. Розглянемо їх детальніше.

N-ярусний архітектурний шаблон допомагає провести структуризацію ПЗ, шляхом його розбиття на певну кількість абстрактних рівнів (ярусів), які у свою чергу виконують характерну для кожного рівня підмножину задач. Кількість ярусів необмежена, але зазвичай інформаційні системи в бізнес-застосунках часто використовують двоярусну архітектуру – більш відому під назвою клієнт-серверна архітектура (рис. 1.4).

Даний шаблон має такі переваги: повторне використання ярусів, підтримка стандартизації, залежності знаходяться локально, замінність. Недоліки: каскадне змінювання поведінки (зміна в одному ярусі приводить до зміни інших), низька ефективність, дублювання сервісів.

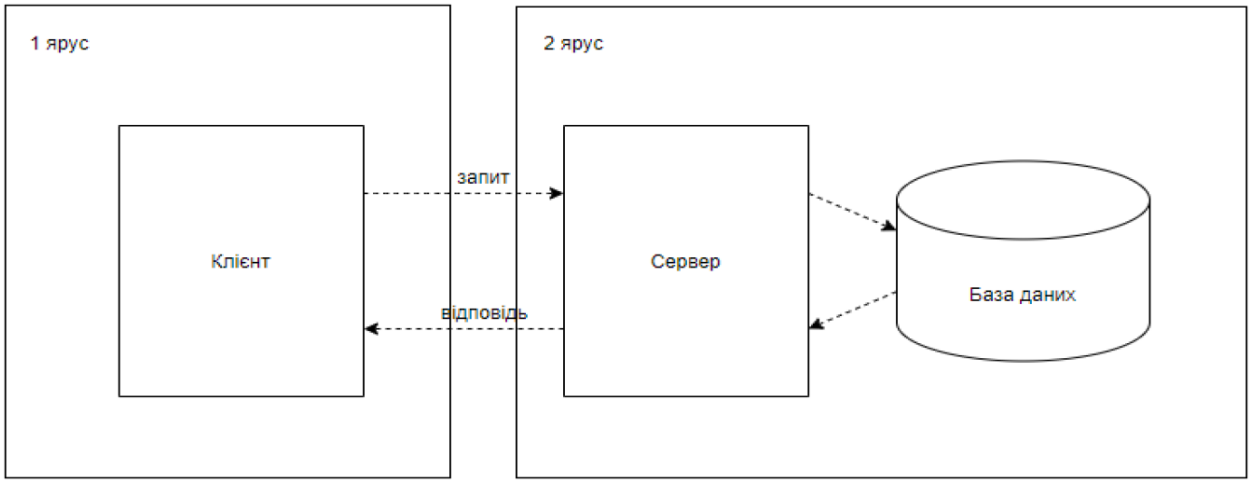


Рисунок 1.4 – Структура двоярусної архітектури

Архітектурний шаблон MVC передбачає поділ системи на три складові частини (рис. 1.5). Модель реалізує основні функціональні можливості та містить дані. Вид відображає інформацію для користувачів. Контролер

обробляє введені користувачем данні. Вид і контролер разом складають інтерфейс користувача, а механізм розповсюдження змін забезпечує узгодженість між інтерфейсом користувача та моделлю.

Переваги шаблона: синхронізованість даних, більшість фреймворків реалізують та використовують даний шаблон для створення ПЗ, багаточисленні види для однієї моделі. Недоліки шаблона: тісний зв'язок між видом і контролером, труднощі використання MVC з сучасними інструментами інтерфейсами користувача.

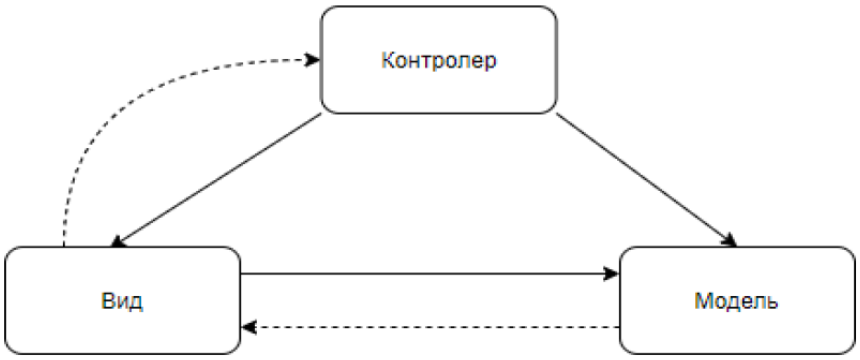


Рисунок 1.5 – Структура MVC

Архітектурний шаблон MVVM – це шаблон, який полегшує відокремлення розробки інтерфейсу користувача (UI) від бізнес-логіки (рисунок 1.6). Як і в класичному MVC, Модель містить дані, а Вид представляє інтерфейс користувача і підписується на події зміни значень властивостей і команд, що надаються Моделлю представленням. Сутність Модель представлення є частиною, яка відповідає за перетворення даних для їх подальшої підтримки і використання. В ній реалізується бізнес-логіка застосунка. Також містить у собі команди за допомогою яких Представлення може впливати на Модель.

Переваги шаблона: легко зрозуміти логіку Представлення, легше тестувати, менше коду. Серед недоліків можна виділити один – даний шаблон підходить для великих проектів, у яких велика кількість складних UI.

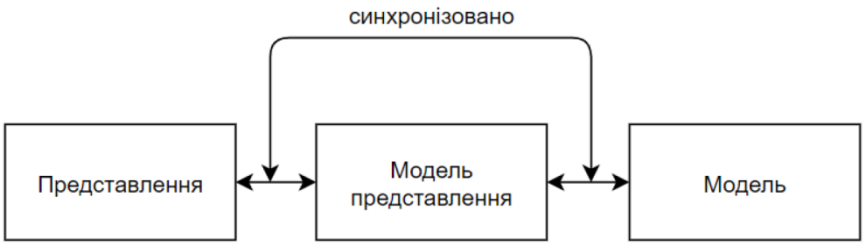


Рисунок 1.6 – Структура MVVM

Вище було сказано, що зазвичай інформаційна система складається з більш ніж одного архітектурного шаблона. Тому описувана в даній роботі інформаційна система не є винятком. Вона поєднує вище описані шаблони певним чином. За основу взято клієнт-серверну архітектуру, тобто ПЗ буде розділене на 2 яруси: клієнт та сервер з базою даних. Взаємодія між клієнтом і сервером буде за допомогою протоколу HTTPS, тобто REST взаємодія. Сервер в свою чергу буде використовувати шаблон MVC, але оскільки передбачається REST взаємодія, тоді шар Представлення буде реалізовано на клієнті у вигляді окремого односторінкового web-додатка. Оскільки клієнт це SPA, то він буде реалізовувати шаблон MVVM, оскільки завдяки ньому з легкістю можна будувати UI різної складності. Детально архітектуру зображено на рис.1.7, однак слід зауважити, що на рисунку також позначені application server та web server. Це означає, що написаний сервер буде виконуватися на сервері додатків (англ. application server).

Для полегшення розробки було обрано 3 найбільш доцільні для даної системи шаблони проектування.

Декоратор – це структурний шаблон проектування, який дозволяє динамічно додавати об’єктам нову функціональність, шлях їх загортання в корисні «обгортки». Даний шаблон корисний в випадках, коли неможливо або небажано використовувати наслідування для розширення функціоналу. Також плюсом виступає те, що об’єктам не тільки можна давати обов’язки (новий функціонал) але й знятих їх при непотрібності.

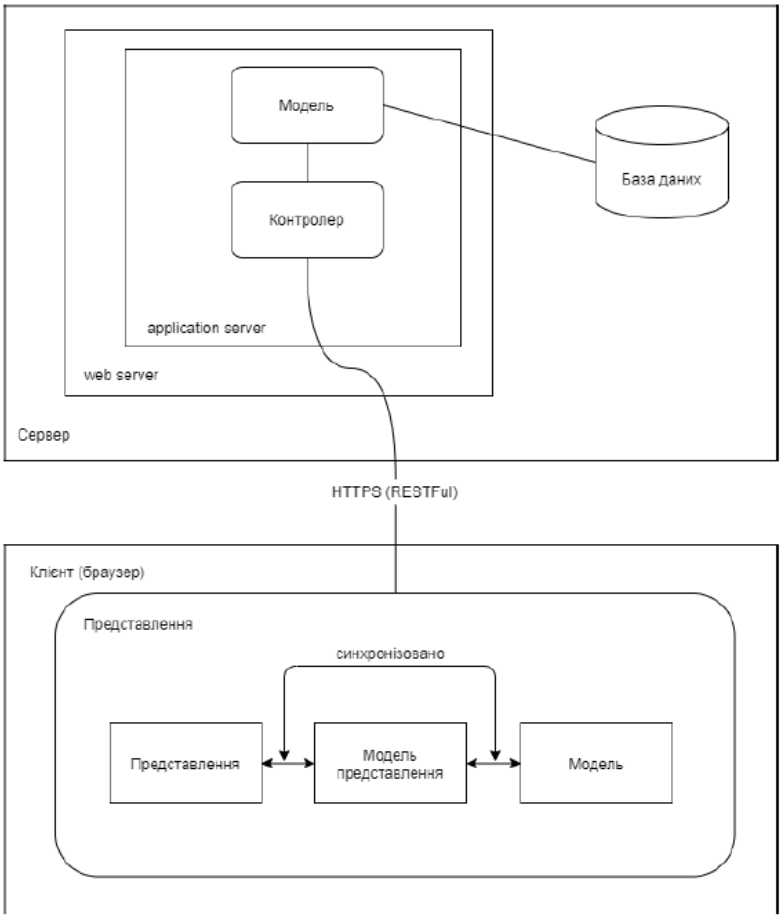


Рисунок 1.7 – Архітектура проекту

Ітератор – це шаблон проектування, який належить до класу шаблонів поведінки. Він дозволяє послідовно обходити елементи складених об’єктів, не розкриваючи їх внутрішнього представлення.

Шаблонний метод – це шаблон проектування, який належить до класу шаблонів поведінки. Він визначає скелет алгоритму, перекладаючи відповідальність за деякі його кроки на підкласи. Шаблон дозволяє підкласам перевизначити кроки алгоритму, не змінюючи його загальної структури.

1.4 Вибір мови та середовища програмування

1.4.1 Вибір мови програмування

Спочатку визначимось з мовою програмування на клієнті. Оскільки клієнтський додаток буде працювати у браузері в якості web-додатка, то

вибір мови програмування стає очевидним – це буде JavaScript або мова, яка транслюється в нативний JavaScript. Серед мов, що транслюються в JavaScript, розглянемо найбільш відомі TypeScript та CoffeeScript та порівняємо їх з JavaScript. На основі цих порівнянь виберемо найбільш придатну мову для розробки клієнта.

JavaScript – це динамічна, інтерпретована мова програмування високого рівня з підтримкою декількох парадигм програмування. JavaScript це мова програмування web-мережі. Переважна більшість web-сайтів використовує JavaScript, а всі сучасні web-браузери – в персональних комп'ютерах (ПК), планшетах та смартфонах – містять у собі інтерпретатор JavaScript, роблячи її найбільш розповсюдженою мовою програмування за всю історію [1].

JavaScript має динамічну і слабку типізацію, автоматичне керування пам'яттю та прототипне наслідування. Значна особливість JavaScript – функції виступають у якості об'єктів першого роду, тобто функція є об'єктом над яким можна проводити усі операції, що і над іншими об'єктами: передавати функцію у якості аргумента в іншу функцію, повертати функцію в якості результата з іншої функції, присвоювати функції атрибути.

В JavaScript відсутнє паралельне виконання коду з використанням потоків або процесів, але це не проблема, оскільки в JavaScript відмінно реалізована асинхронна обробка даних без якої важко уявити написання більш-менш складного проекту.

Одна з багатьох переваг JavaScript полягає у великій спільноті розробників, а також наявність великої кількість готових плагінів, інструментів та готових рішень для будь-якої задачі.

CoffeeScript – це невеличка мова програмування, яка компілюється в JavaScript. CoffeeScript – це спроба простим способом розкрити усі переваги JavaScript. Код написаний на CoffeeScript компілюється один в один в еквівалентний код на JavaScript, і не підтримує інтерпретації під час виконання. Скомпільований код працює так само швидко, а іноді навіть швидше, ніж еквівалентний код на JavaScript. Також можна використовувати будь-яку

існуючу бібліотеку JavaScript із CoffeeScript (і навпаки) [2]. Безумовно перевага CoffeeScript над JavaScript у більш компактному, лаконічному та зрозумілому синтаксисі.

TypeScript – це мова програмування з відкритим кодом, яка базується на JavaScript, додаючи статичну типізацію. Типи надають спосіб описання форми об'єктів, забезпечуючи кращу документацію та дозволяючи TypeScript перевірити чи правильно працює написаний код на етапі компіляції [3]. Код TypeScript перетворюється в код JavaScript за допомогою компілятора TypeScript або Babel. Цей JavaScript код представляє собою чистий та простий код, який запускається всюди, де працює JavaScript, наприклад в браузері. TypeScript дуже економить час на знаходження помилок та їх виправлення.

У табл. 1.2 узагальнено вище сказане про кожну мову програмування на основі порівняння їх можливостей. Звідси бачимо, що незважаючи на свою ефективність, зручність у роботі та інших перевагах, TypeScript та CoffeeScript не можуть зрівнятися з JavaScript. Тому розборка клієнтської частини буде проходити з використанням JavaScript в якості основної мови програмування.

Таблиця 1.2 – Порівняння мов програмування для написання клієнта

	JavaScript	TypeScript	CoffeeScript
Компактний та лаконічний синтаксис	–	–	+
Висока продуктивність	+	+/–	+/–
Статична типізація	–	+	–
Велика кількість готових рішень	+	–	–
Велика спільнота	+	+/–	–
Висока швидкість розробки	+/–	+/–	+

Далі виберемо мову програмування для написання серверної частини. Для цього розглянемо 3 найбільш уживаних для серверної розробки мов про-

грамування. Із рейтингу мов програмування на сайті dou.ua можна побачити, що за 2021 рік найбільш популярні мови для розробки серверної частини є Java, C# та Python [4]. Отже, порівняємо ці мови та виберемо найбільш доцільну для вирішення поставлених задач.

Java – це об’єктно-орієнтована мова програмування високого рівня, яка має простий але багатослівний синтаксис. Мова має широкий спектр використання, але здебільшого її використовують в ентерпрайз розробці (англ. enterprise development) та для написання мобільних додатків на Android. Програми написанні на Java компілюються в байт-код та інтерпретується віртуальною машиною (JVM). Код на Java переносимий, не залежить від вибору операційної системи (ОС) та має високу швидкість виконання завдяки JIT-компіляції, яка перетворює найбільш уживані частини коду в нативний код ОС.

Java позиціонується як безпечна мова програмування завдяки відсутності вказівників, автоматичному управлінні пам’яті та збирачу сміття. Також мова має чудову реалізацію паралельного виконання коду та статичну типізацію.

За всі роки існування Java об’єднала навколо себе велику спільноту розробників. Завдяки цьому існує багато готових рішень для різних задач на Java.

C# – це об’єктно-орієнтована мова програмування високого рівня з статичною типізацією для платформи .NET. Мова використовується здебільшого для ентерпрайз розробки. В свій час C# зазнала впливу з боку Java, тому велика кількість можливостей, що реалізована в Java також наявна в C#. Однак, C# має ряд особливостей, такі як підтримка динамічного виведення типів, вказівників та перезавантаження операторів. Ці особливості C# ввібрав від мови програмування C++. Тому й не дивно, що синтаксис C# являє собою комбінацію синтаксистів C++ та Java.

Як і Java, за роки свого існування C# зібрав навколо себе вражаючу кількість розробників.

Python – це динамічно типізована, об’єктно-орієнтована мова програмування з відкритим кодам. Python має різні сфери використання, але здебільшого його використовують в web-розробці та Data Science. Він має лаконічний, легко читаємий та приємний синтаксис. Підходить для початківців, оскільки дуже легкий у вивченні. Python можна запустити під управлінням будь-якої операційної системи, тому код на Python є переносимим.

Python має динамічну типізацію, а також усе в ньому є об’єктом. Функції виступають у якості об’єктів першого роду. Також для Python притаманні качина типізація, множинне наслідування, замикання та елегантно реалізоване перевантаження операторів. Python з перших кроків навчає гарному стилю програмування, оскільки в його реалізації входять такі шаблони проектування, як декоратор, ітератор, проксі. Дана мова програмування підтримує декілька парадигм програмування.

Python має потужні елементи функціонального стилю програмування. Навіть такі речі, як наслідування (відноситься до ООП), реалізовано в функціональній манері через дескриптори та розширення функціоналу оператора доступу до атрибутів об’єкта (крапки “.”).

Одним з недоліків Python вважається його повільна робота у порівнянні з мовами зі статичною типізацією. Однак, є декілька способів вирішення цієї проблеми і найлегший с них – використовувати у якості інтерпретатора PyPy. З ним швидкість виконання коду зростає майже в 100 разів і наближається до показників C та C++.

Python має велику спільноту розробників, а отже – й велику кількість фреймворків, готових бібліотек та рішень для різних задач.

Отже, у табл. 1.3 порівняно вище розглянуті мови програмування для написання серверної частини. Звідси можна побачити, що найкращий варіант – це Python.

Таблиця 1.3 – Порівняння мов програмування для написання серверної частини

	Java	C#	Python
Статична типізація	+	+	–
Висока швидкість розробки	–	–	+
Компактний та лаконічний синтаксис	–	–	+
Висока швидкість виконання коду	+	+	+/–
Велика спільнота	+	+	+
Велика кількість готових рішень	+	+	+
Переносимість програм	+	+	+
Безпечне управління пам’яттю	+	+	+

1.4.2 Вибір середовища програмування

Важко уявити ефективну розробку будь-якого продукта без необхідних інструментів. Найбільш важливим інструментом у розробці виступає середовище програмування, оскільки завдяки йому вдається мінімізувати витрати часу на повсякденні задачі, такі як профілювання та відлагодження програм, та побудувати ефективний робочий процес.

Розглянемо найбільш придатні для розробки середовища програмування вище описаним мовам програмування. Лідерами на ринку виступають Visual Studio Code, PyCharm та WebStorm. Отже, розглянемо їх ближче.

Visual Studio Code (VS Code) – це легкий, але потужний редактор з відкритим кодом, який працює під управлянням Windows, macOS та Linux [5]. Редактор постачається з вбудованою підтримкою JavaScript, TypeScript та Node.js і має багату екосистему розширень для середовищ виконання (таких як .NET та Unity) та майже усіх відомих мов програмування. В VS Code інтегрована підтримка різних систем контролю версій та робота з терміналом. Також редактор має інструментарій для профілювання та відлагодження програм. Одна з найбільших переваг редактора – він постачається безкоштовно.

PyCharm – це IDE для професійної розробки на Python. Редактор можна встановити на Windows, macOS та Linux. PyCharm позиціонує себе як розум-

ний редактор коду. Він автоматично доповнює і аналізує код, підсвічує помилки та надає підказки щодо їх усунення [6]. PyCharm має вбудовані інструменти для розробників, такі як налагодження, тестування та профілювання, підтримку систем контролю версій, наявність інструментів для роботи з базами даних, а також велику кількість плагінів, що забезпечують підтримку найбільш популярних фреймворків для web-розробки не тільки на Python, а і на JavaScript.

WebStorm – це розумна IDE для JavaScript [7]. Оскілки WebStorm та PyCharm розробляються однією компанією JetBrains, то редактор WebStorm має такі ж самі можливості, що і PyCharm, з поправкою на те, що WebStorm спеціалізується на роботі з JavaScript та його екосистемою, а також має інструменти збірки проектів, такі як Grunt, Gulp та npm. Найбільшим недоліком редакторів WebStorm та PyCharm виступає їх велика вартість.

Отже, у таблиці 1.4 порівняно розглянуті вище середовища програмування. Звідси можна побачити, що VS Code найбільш придатний редактор коду для розробки завдяки тому, що він безкоштовний і має однакові можливості у порівнянні з платними аналогами.

Таблиця 1.4 – Порівняння середовищ програмування

	PyCharm	WebStorm	VS Code
Підтримка систем контролю версій	+	+	+
Підтримка інструментів для відлагодження програм	+	+	+
Підтримка іншим мов програмування	+	+	+
Автокорекція та аналіз коду	+	+	-
Безкоштовна	-	-	+

1.5 Вибір технологій для розробки інформаційної системи

1.5.1 Вибір технологій для сервера

Оскільки Python одна з найпопулярніших мов для написання backend, він має велику кількість фреймворків, як великих за розміром та функціона-

лом, так і малих, як популярних, так і набираючих популярність, які в тій чи іншій мірі полегшають даний процес. Тому, розглянемо найбільш популярні феймворки та оберемо серед них той, що задовільнить усім умовам, а саме: гнучкість, швидкість виконання, наявність повної та якісно написаної документації, наявність готових архітектурних рішень.

Із офіційної документації, Django – це високорівневий web-фреймворк на Python, який сприяє швидкій розробці та чистому, прагматичному дизайну [8]. Він бере на себе більшу частину клопіт під час створення продукту, тому можна зосередитись на вирішенні конкретних бізнес-задач, не вдаючись у написанні повсякденних задач, таких як обробка запитів або обробка URL адресів. Також фреймворк дозволяє в найближчий час перейти від ідеї до конкретного прототипу.

Django робить акцент на безпеку, гнучкість та масштабування. Також він намагається дати усе із коробки, тобто надати розробнику усі необхідні інструменти, такі як ORM, шаблонізатор, роутинг, сигнали, тестовий клієнт та інше. Фреймворк має вбудований WSGI, який можна використовувати під час розробки і тестування. Однак він не підходить для реального використання, тому необхідно використовувати один з виробничих серверів додатків, наприклад Gunicorn або uWSGI.

Django підтримує концепцію повторного використання додатків – це стало причиною існування великої кількісті готових модулів та бібліотек для даного фреймворка.

Django побудований з використанням великої кількості шаблонів проектування, як архітектурних, так і GoF. Наприклад, Django використовує архітектурний шаблон MVT, що являє собою переосмислений MVC. Як і в MVC, літера M означає модель або model. Однак за контролер (літера C у MVC) відповідає в'ю або view (V у MVT), а представлення (V у MVC) – шаблон або template (T у MVT).

Django заохочує використання різних принципів написання якісного коду, наприклад DRY, KISS.

Діла розглянемо наступний менш популярний у порівнянні з Django, але достатньо популярний фреймворк Twisted.

Згідно офіційній документації Twisted – це подійно-орієнтований (асинхронний) мережевий web-фреймворк на Python [9]. Він підтримує велику кількість протоколів передачі даних та дозволяє з легкістю розгорнути сервер на будь-якому з них. Також фреймворк відомий своєю продуктивністю та швидким виконанням коду завдяки асинхронному виконанні програм.

Однією з важливих можливостей Twisted виступає використання event loop не тільки написаних на Python, а і на інших мовах програмування.

Як і Django, Twisted розроблено з використанням різних шаблонів проектування, наприклад реалізація шаблонів фабричний метод та абстрактна фабрика, які допомагають використовувати в коді різні типи протоколів.

Ще у часи, коли Python не підтримував асинхронність та не мав модуля asyncio, розробники Twisted взялися за розробку власних механізмів асинхронного виконання коду. Тому популярність Twisted зумовлена наявністю асинхронності, а саме відсутності її у Python в минулому. Однак це не означає, що Twisted погано реалізує асинхронність і скоро кане в Лету. Не зважаючи на те, що фреймворк був одним із перших, він на рівні змагається з більш новими асинхронними фреймворками, а в деяких випадках випереджає їх як у швидкості та продуктивності, так і в частоті появі нових можливостей та функцій.

Отже, зваживши усі переваги та недоліки описаних фреймворків, було вирішено використовувати Django.

Для розробки необхідна бібліотека Django Rest Framework (DRF). DRF – це бібліотека, яка призначена для полегшення розробки API. Вона надає безліч інструментів та дозволяє створити сервер довжиною в декілька строчок коду.

1.5.2 Вибір технологій для клієнта

Далі порівняємо фреймворки для JavaScript та оберемо серед них найбільш кращий.

Vue.js – це прогресивний фреймворк для написання web-додатків. На відміну від аналогічних фреймворків-монолітів, Vue можна поступово впроваджувати у проект в якості бібліотеки. Оскільки ядро Vue в першу чергу вирішує задачі рівня представлення (англ. view), тому це спрощує інтеграцію з іншими бібліотеками та існуючими проектами [10]. Однак, фреймворк також добре підходить для створення односторінкових web-додатків (SPA) різної складності. У цьому Vue допомагає інструмент CLI та велика кількість бібліотек, написаних для цього фреймворка.

Основні можливості Vue.js:

- віртуальний DOM;
- підтримка реактивності (завдяки реалізації архітектурного шаблону MVVM);
- має ядро з основними функціями, а реалізацію інших можливостей (наприклад, роутинг або управління глобальним станом додатку) виносить в додаткові бібліотеки;
- використовує мало пам'яті та має високу швидкість роботи;
- підтримка одно- та двостороннього зв’язувань, компонентного підходу, великої кількості директив та модульності;
- підтримує JSX як альтернативу компонентам;
- якісна документація;
- легко вивчити.

Серед недоліків фрейморка можна відзначити його надлишкову гнучкість. У Vue існує багато способів вирішення однієї задачі. Це нерідко стає перешкодою для новачків, оскільки не всі способи є доцільним та раціональними і вони можуть вплинути як на швидкість кода, так і на його якість.

Svelte.js – це дуже молодий фреймворк для розробки web-додатків на JavaScript.

Svelte приніс радикально новий підхід до побудови користувальницьких інтерфейсів. У той час як традиційні фреймворки, такі як React та Vue, виконують основну частину своєї роботи в браузері, Svelte перекладає цю роботу на етап компіляції, який відбувається під час створення програми [11].

Замість використання різності (англ. diff) віртуального DOM, Svelte пише код, який хірургічно (тобто саме ті елементи, які були змінені) оновлює DOM, коли стан додатка змінюється.

Основні переваги Svelte.js:

- відсутній віртуальний DOM;
- компонентний підхід;
- висока продуктивність у порівнянні з іншими фреймворками;
- компактний синтаксис;
- простий для вивчення ;
- краща реалізація реактивності;
- швидке зростання, як в плані популярності, так і в плані розробки нових функцій.

Незважаючи на вагомі переваги, Svelte має великі недоліки. Через всій молодий вік, він має мале ком'юніті та малу кількість бібліотек. Але головний недолік – нестабільна робота фреймворка, оскільки він знаходиться на фазі бета-версії.

React.js – це JavaScript бібліотека для створення інтерфейсу користувача [12]. Оскільки React – бібліотека, то вона не дає усіх можливостей, що дають вище описані фреймворки. React надає лише інструменти для візуалізації вмісту DOM та ефективного управління ним після цього. Він не включає вбудовану підтримку перевірки форм, маршрутизацію і не постачає власний HTTP-клієнт. Тому для розробки web-додатків, необхідно підключати сторонні бібліотеки.

Ось основні переваги React:

- використовує віртуальний DOM;
- підтримує компонентний підхід;
- швидкий;
- легкий у вивченні;
- має велику спільноту розробників;
- завдяки React Native може використовуватися для мобільної розробки.

Серед недоліків можна виділити погано структуровану документацію та нав’язування розробки з використанням JSX без надання будь-яких альтернатив.

Зважаючи на всі недоліки та переваги, було вирішено, що найбільш доцільним варіантом виступає Vue.js.

Для створення web-додатку необхідна бібліотека Axios. Axios – це HTTP клієнтів для браузерів, який мало споживає ресурси системи. У Axios є підтримка запитів, отримання відповідей від сервера, їх трансформація і автоматична конвертація в JSON. Однією з найбільших переваг є безпека – в бібліотеку вбудовано захист від підробки запитів між сайтами.

1.6 Вибір бази даних

Важливим аспектом, без якого не можливе існування складних систем, є збереження інформації. Найбільш придатними виступають бази даних. Зараз виділяють 2 основні види баз даних: SQL та NoSQL.

SQL бази даних – це різновид баз даних, які використовують декларативну мову програмування SQL. Зазвичай їх використовують, коли між сутностями є тісні зв’язки. Перевагами SQL баз даних є незалежність від конкретної системи керування баз даних (СКБД), наявність стандартів та опис намірів (декларативність), тобто програміст описує тільки ті дані, які

необхідно дістати або змінити. Серед недоліків можна зазначити, що SQL бази даних мають складності в роботі з ієрархічними структурами даних.

NoSQL бази даних – це різновид баз даних, які не використовують SQL. По суті, NoSQL дані не є реляційними, а NoSQL бази даних зазвичай не мають схем і вони мають більш узгоджену модель, ніж наявні в традиційних реляційних БД. Перевагами NoSQL баз даних виступають легкість у масштабуванні та висока продуктивність. Серед недоліків NoSQL баз даних можна зазначити низьку надійність, великий об’єм даних та залежність від СКБД.

Зважаючи на усі переваги та недоліки вище описаних типів баз даних, для написання проекту було обрано SQL бази даних, оскільки для проекту важливі структура та зв’язки між сутностями. Далі порівняємо та виберемо для проекту базу даних.

Oracle Database – об’єктно-реляційня система керування базами даних компанії Oracle. Була розроблена для корпоративних розподілених обчислень. Перевагами даної СКБД є висока швидкість обробки транзакцій, сумісність з ACID принципами, обробка великих даних, популярність у розробників, простота у використанні, зрозуміла документація, підтримка довгих найменувань, JSON, покращений тег списку і Oracle Cloud. Найбільший недолік Oracle – її висока вартість.

PostgreSQL – масштабна об'єктно-реляційна СКБД, що працює на Linux, Windows, OSX і деяких інших системах. У PostgreSQL є такі функції, як логічна реплікація, декларативне розбиття таблиць, поліпшені паралельні запити, більш безпечна автентифікація по паролю на основі SCRAM-SHA-256. Також у PostgreSQL є підтримка табличних просторів, збережених процедур, об'єднань, представлень і тригерів та асинхронна реплікація. У порівнянні с Oracle, Postgres має менший функціонал та продуктивність, але в той же час Postgres має найбільшу перевагу – безкоштовність.

Безкоштовність та функціонал PostgreSQL цілком задовольняє цілям проекту, тому розробка буде проходити з її використанням.

1.7 Висновки до першого розділу

Отже, в даному розділі кваліфікаційної роботи бакалавра було визначено та проаналізовано предметну область.

Було проведено аналіз аналогів. Таким чином отримали, що аналоги представлені у вигляді веб-сайтів. Кожний з них вирішує лише окремий вид проблем, тому вони не підходять для лізинга, на відміну від нашого додатка. Також аналоги мають не досить хорошу структуру та інтерфейс. Наш додаток пропонує зручний інтерфейс та хорошу структуру. Більшість з них потребують плату за використання, однак наш додаток цілком безкоштовний. Отже, можна побачити, що по усім характеристикам наш додаток кращий ніж відповідні аналоги.

Серед багаточисельних технологій було обрано декілька для написання проекту. Клієнт буде написаний на мові програмування JavaScript з використання фреймворка Vue.js та бібліотеки Axios. Сервер буде написаний на мові програмування Python з використання фреймокра Django та бібліотеки DRF. У якості бази даних виступатиме PostgreSQL. Також було обрано єдине середовище програмування – VS Code.

Для майбутнього веб-сервісу було вибрано архітектурний шаблон, який є комбінацією клієнт-серверної архітектури, шаблонів MVC та MVVM. Клієнт-серверна взаємодія відбувається за допомогою REST API. Також для проекту було обрано 3 шаблони проектування, які мають за мету полегшити проектування тривіальних задач.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Мета і задачі інформаційної системи

Інформаційна система, що розробляється, призначена для автоматизації бізнесу з лізингу медичного обладнання, а також полегшити пошук даного виду обладнання простим людям. Система розроблена у вигляді web-додатку.

Мета ІС: створити інтернет-майданчик, на якому зможуть зустрітися ті люди, кому потрібно медичне обладнання, і ті люди, у кого воно наявне і можуть передати його в лізинг. Даний процес має бути зручним і швидким для обох типів людей.

Цільова аудиторія: лізингодавці та лізингоодержувачі (люди, які хочуть взяти обладнання у лізинг).

Головні функції ІС:

– надати можливість зареєструватися та авторизуватися;

– розподіл обладнання по категоріям;

– можливість додати обладнання;

– можливість стежити за обладнанням та керувати ним;

– збереження історії лізингу;

– нагадування про завершення або прострочення термінів лізингу;

– можливість забронювати обладнання;

– пошук обладнання по містах, датах і категоріях;

– відправка статистики та договору про лізинг на пошту користувача;

– відправка різного виду повідомлень на пошту користувача.

Опишемо вхідні та вихідні інформаційні потоки.

Вхідні:

– ПІБ, поштова адреса, телефон користувача;

– назва обладнання, опис, тип, вага, завантажувальна вага, матеріал, габарити, виробник, ціна, знижки, розмір застави, місце розташування;

– місто, дати початку та кінця лізингу, категорія обладнання. 30

Вихідні:

– повідомлення про завершення бронювання, про прострочення термінів бронювання, про успішне додання обладнання;

– повідомлення з детальною статистикою;

– результати пошуку;

– договір лізингу.

На рисунку 2.1 зображено діаграму вхідних та вихідних інформаційних потоків.

Рисунок 2.1 – Діаграма вхідних та вихідних інформаційних потоків

2.2 Типи користувачів

Гість – незареєстрований у системі користувач, який має можливість переглядати головну сторінку, каталог та інформацію про обладнання, але не має можливості взяти або здати обладнання у лізинг. Також гість може зареєструватися у системі.

Користувач – цей тип користувачів об’єднує 2 типи людей: люди, які мають намір здавати обладнання у лізинг та автоматизувати ведення бізнесу

(лізингодавці), а також люди, які хочуть взяти обладнання в лізинг (лізингоодержувачі). Користувач може війти в систему, додати та редагувати обладнання, редагувати профіль, переглядати каталог та обладнання, отримати статистику на пошту (якщо він здає обладнання).

Адміністратор – цей тип користувачів виступає у ролі модераторів, тобто адміністратори можуть видаляти користувачів та усю інформацію, які порушують або не відповідають правовим нормам та правилам системи.

Після визначення користувачів було побудовано діаграму прецедентів (рисунок 2.2 та рисунок 2.3).

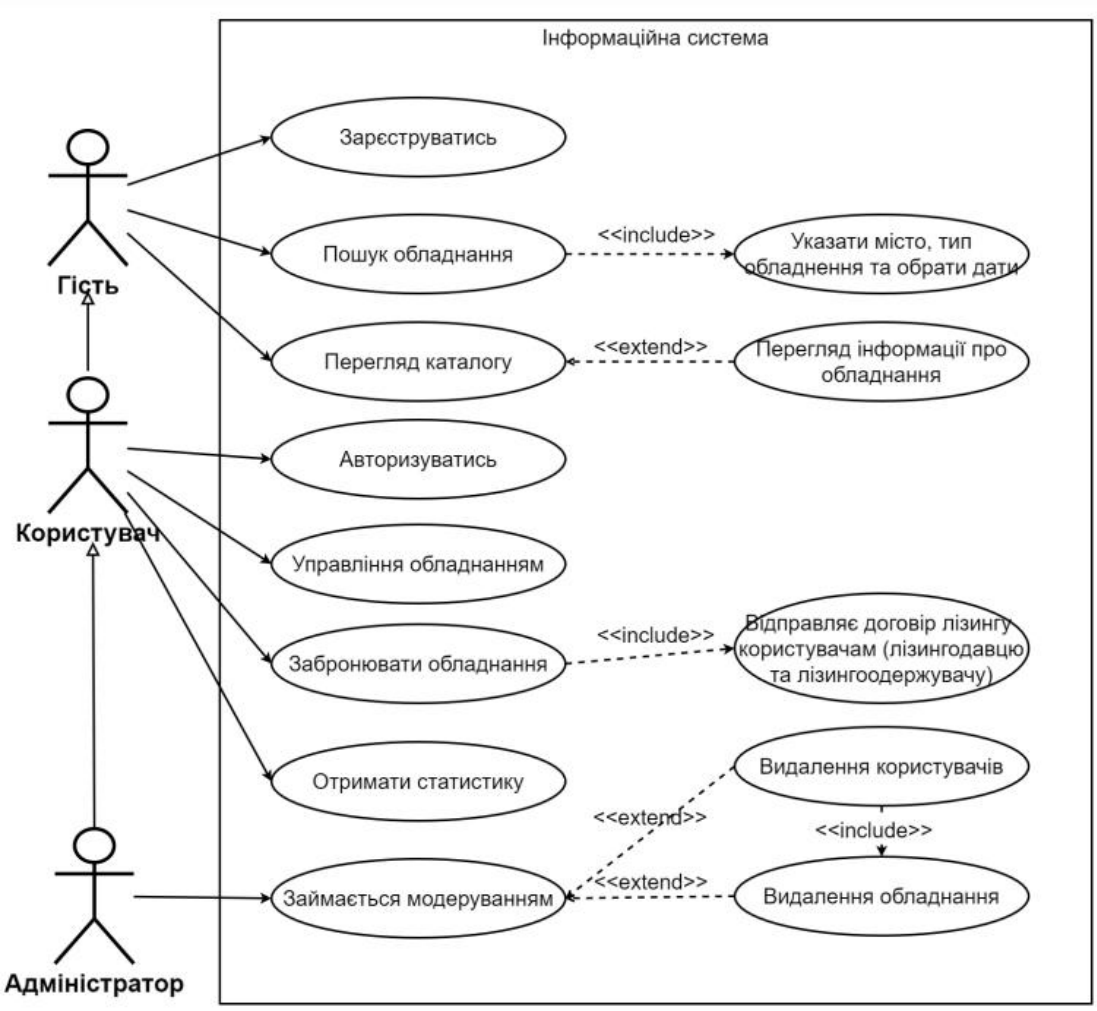


Рисунок 2.2 – Діаграма прецедентів ІС

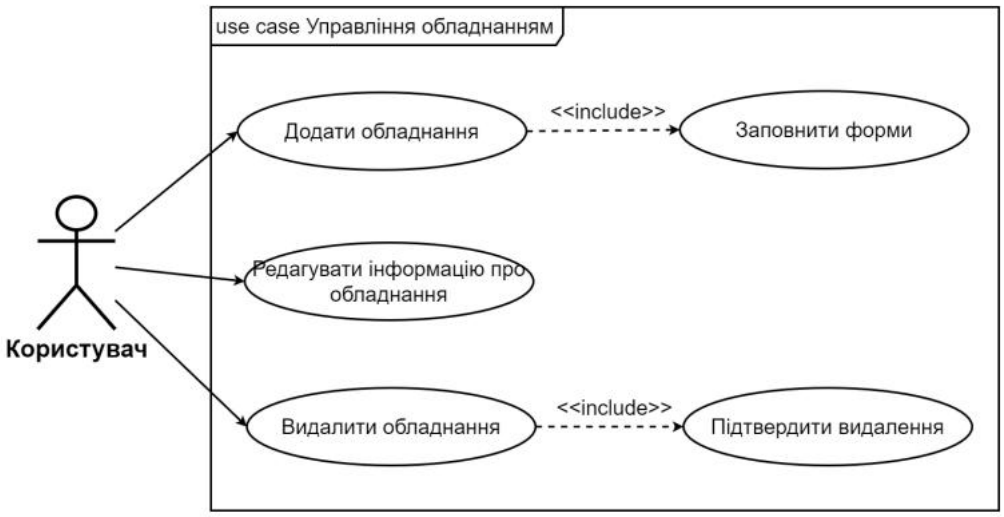


Рисунок 2.3 – Діаграма прецедентів для use case «Управління обладнанням»

2.3 Представлення інтерфейсу користувача (UI View)

Розглянемо мокапи графічного інтерфейсу. Спочатку користувач потрапляє на головну сторінку (рис. 2.4).

З головної сторінки можна потрапити на сторінку реєстрації (рисунок 2.5 б) або на сторінку авторизації (рисунок 2.5 а). Також з головної сторінки можна потрапити на сторінку каталогу натиснувши кнопку «Пошук» або обравши категорію. Натиснувши кнопку «Розпочати лізинг» в нижній секції, можна потрапити на сторінку додавання обладнання, якщо користувач не здавав досі обладнання в лізинг. Якщо користувач раніше додав обладнання, то він потрапить на сторінку управління обладнанням (рисунок А.1). Оскільки сторінка управління обладнанням подана у Додатку А, тоді коротко опишемо її. З цієї сторінки можна потрапити на головну сторінку, натиснувши на логотип, на сторінку додавання обладнання, натиснувши кнопку «Створити оголошення». При натисканні кнопки «Редагування» система перенаправить користувача на сторінку додавання обладнання, де будуть заповнені поля відповідно його оголошенню. Користувач може змінити значення будь-якого поля та зберегти зміни.

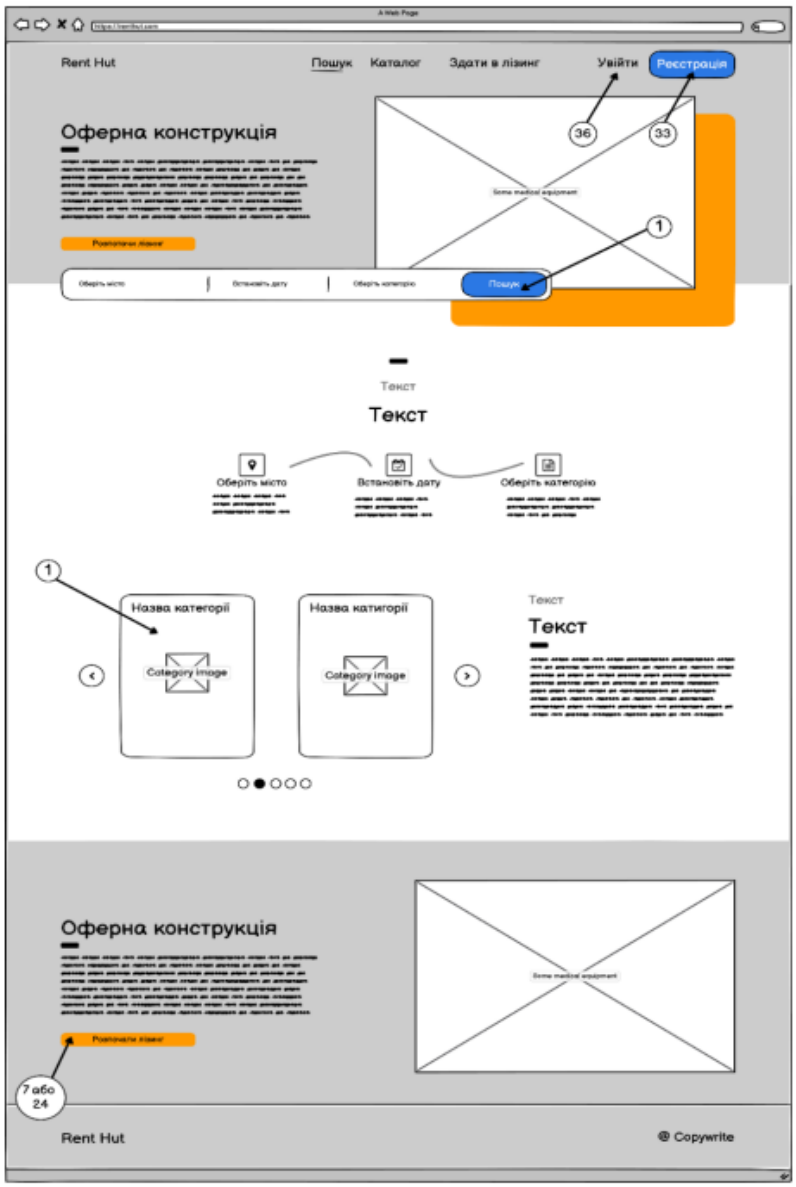
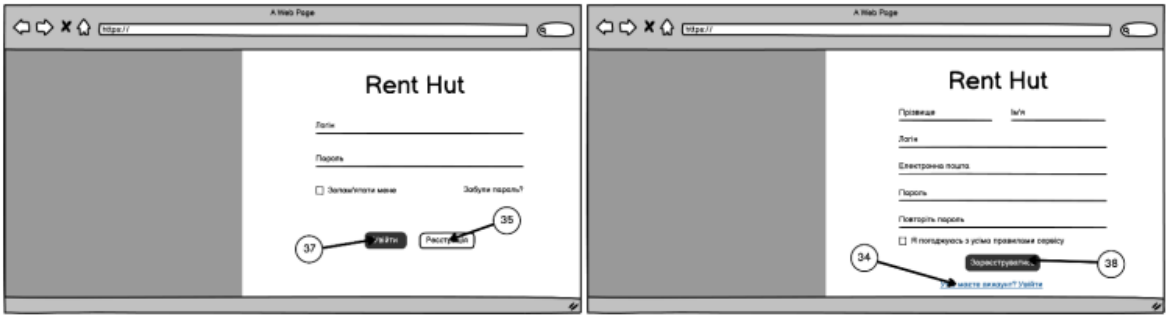


Рисунок 2.4 – Мокап головної сторінки



а) б)

Рисунок 2.5 – Мокапи графіного інтерфейсу: а) сторінка авторизації; б) сторінка реєстрації

Зі сторінки авторизації (рисунок 2.5 а) можна потрапити на сторінку реєстрації (Реєстрація) або на головну сторінку (Увійти) у якості авторизованого користувача. Зі сторінки реєстрації (рисунок 2.5 б) можна потрапити на сторінку авторизації (Уже маєте аккаунт? Увійти) або на головну сторінку (Зареєструватися). На сторінці каталогу (рисунок 2.6) відображаються результати пошуку або усі обладнання за обраною категорією. З даної сторінки можна потрапити на головну сторінку, натиснувши на логотип, а також – на сторінку з детальною інформацією про конкретне обладнання (рисунок А.2).

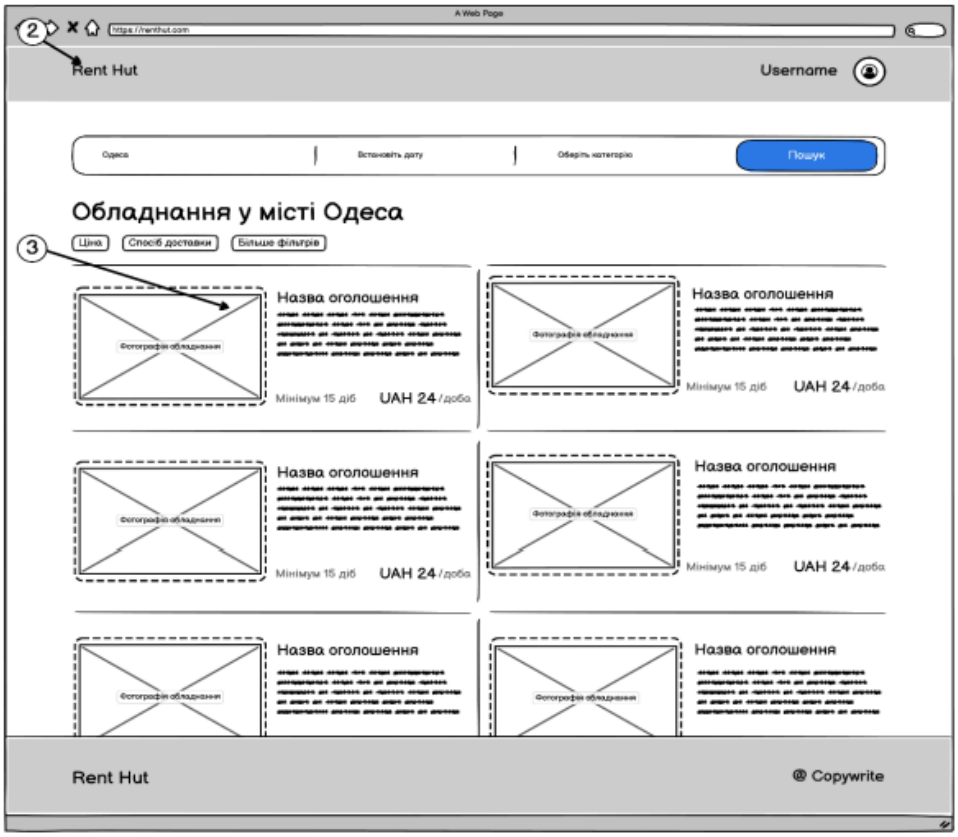


Рисунок 2.6 – Мокап сторінки каталогу

2.4 Логічне представлення ІС (Logical View)

Для отримання повної картини над усією ІС побудуємо діаграму логічного представлення системи (рис. 2.7). На діаграмі зображено web-додаток з яким взаємодіють користувачі. Він виконується у браузері та пев-

ним чином реагує на дії користувачів. Web-додаток взаємодіє з користувачами шляхом відображення необхідної інформації та наданням інтерфейсу для взаємодії з системою. Також web-додаток надсилає запити серверу та у відповідь отримує дані у форматі JSON. Сервер реалізує бізнес логіку, приймає запити від web-додатку, обробляє їх та повертає результат. База даних реагує на запити серверу та повертає йому дані. Слід розглядати сервер та базу даних як одну складену логічну одиницю, оскільки вони будуть ділити один простір.

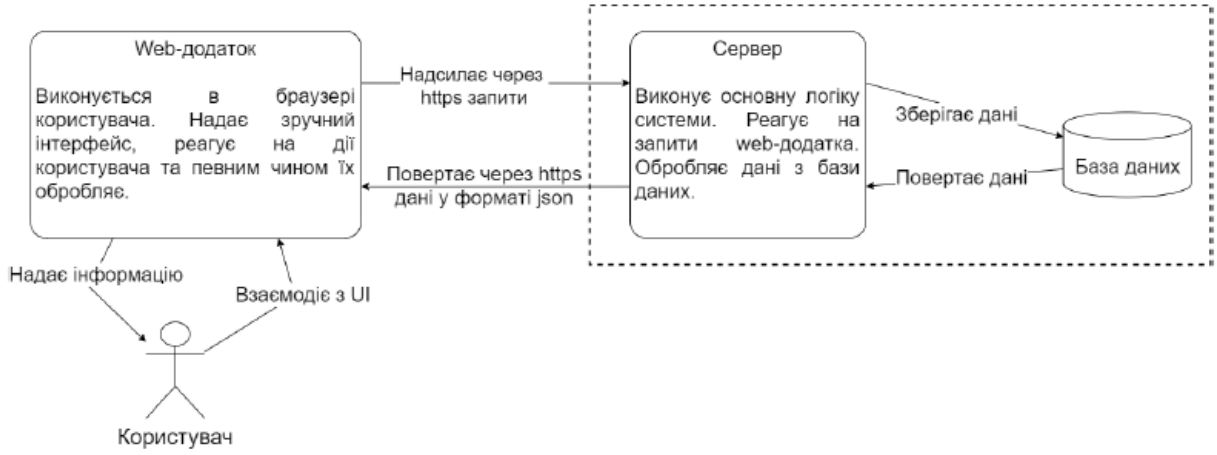


Рисунок 2.7 – Логічне представлення системи

2.5 Представлення розгортання ІС (Deployment View)

На рис. 2.8 зображена діаграма розгортання системи. Звідси модна побачити, що розгортання буде проводитися в 3 етапи. Спочатку буде розгорнуто back-end сервер на локальній машині або хостингу. Він містить 2 компоненти: сервер написаний на DRF та база даних (у даному випадку PostgreSQL). Сервер на DRF буде виконуватися завдяки серверу додатків Gunicorn. Наступним кроком розгортається front-end сервер також на локальній машині або хостингу. Він відповідатиме за віддачу статичних файлів, балансуванні навантаження та на ньому буде міститися web-додаток

на Vue.js. Нарешті сам web-додаток буде розгорнуто на ПК користувача та буде працювати у браузері.

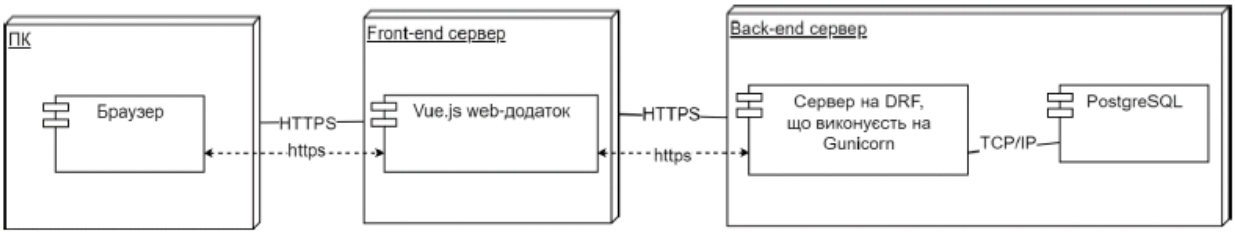


Рисунок 2.8 – Діаграма розгортання ІС

2.6 Представлення процесів ІС (Process View)

Виділимо два основні процеси для інформаційної системи, опишемо їх та побудуємо для них діаграми послідовності та активності.

Сценарій 1 – Додавання обладнання.

Для додання обладнання користувач має обрати відповідний розділ. Додавання проходить у декілька етапів: користувач має поетапно заповнити форму вибору типу обладнання, форму вказання адреси, форму встановлення характеристик, форму вказання назви та опису, форми встановлення цін та застави, форму додавання фотографій та форму вибору способу доставки. Після заповнення всіх форм користувач має підтвердити додавання. Після підтвердження web-додаток відправить запит до сервера на збереження даних. Сервер зберігає дані та повертає повідомлення про успішно виконану операцію, а web-додаток відобразить користувачу це повідомлення. На рис. 2.9 зображено діаграму активності для даного сценарію. Діаграма послідовності зображена на рисунку Б.1 та подана у Додатку Б.

Сценарій 2 – Бронювання (взяття у лізинг) обладнання.

Спочатку користувач користується пошуком, таким чином він запитує цікаві для нього пропозиції. Web-додаток підтягне дані з сервера та відобразить їх користувачу.

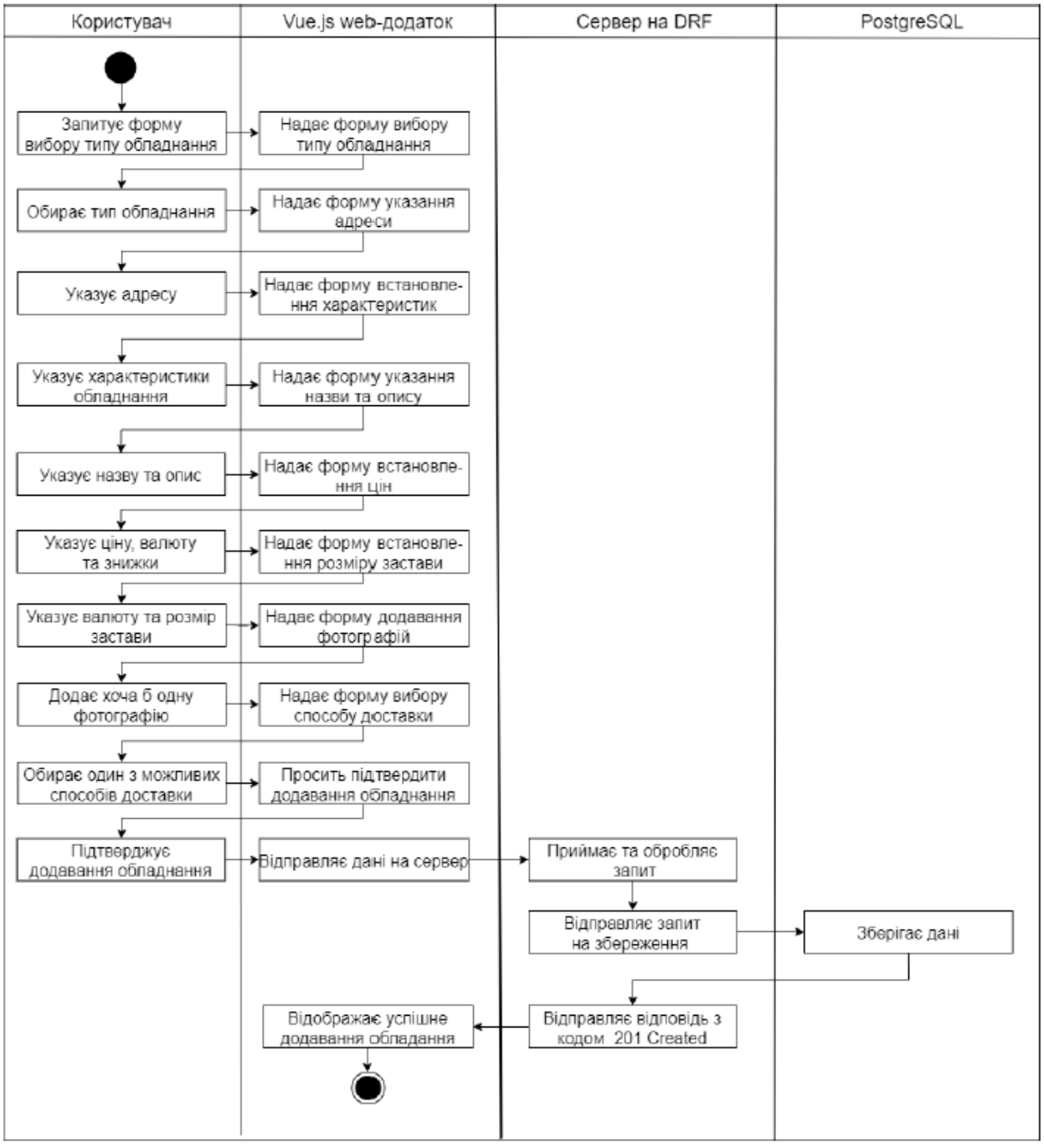


Рисунок 2.9 – Діаграма активності для першого сценарію

Далі користувач обирає будь-яке обладнання, а web-додаток відобразить детальну інформацію про нього. Щоб взяти обладнання у лізинг користувач має обрати терміни лізингу. Система розрахує йому вартість лізингу з урахуванням знижок та тривалості. Користувач має підтвердити бронювання. Останнім кроком система відправить на пошту користувачам (лізингодавцю та лізингоодержувачу) заповнений договір лізингу, а користувачі у свою чергу мають самостійно завірити його у нотаріуса. Діаграма послідовності зображена на рисунку Б.2 та розміщена у Додатку Б.

На рисунку 2.10 зображено діаграму активності для другого сценарію.

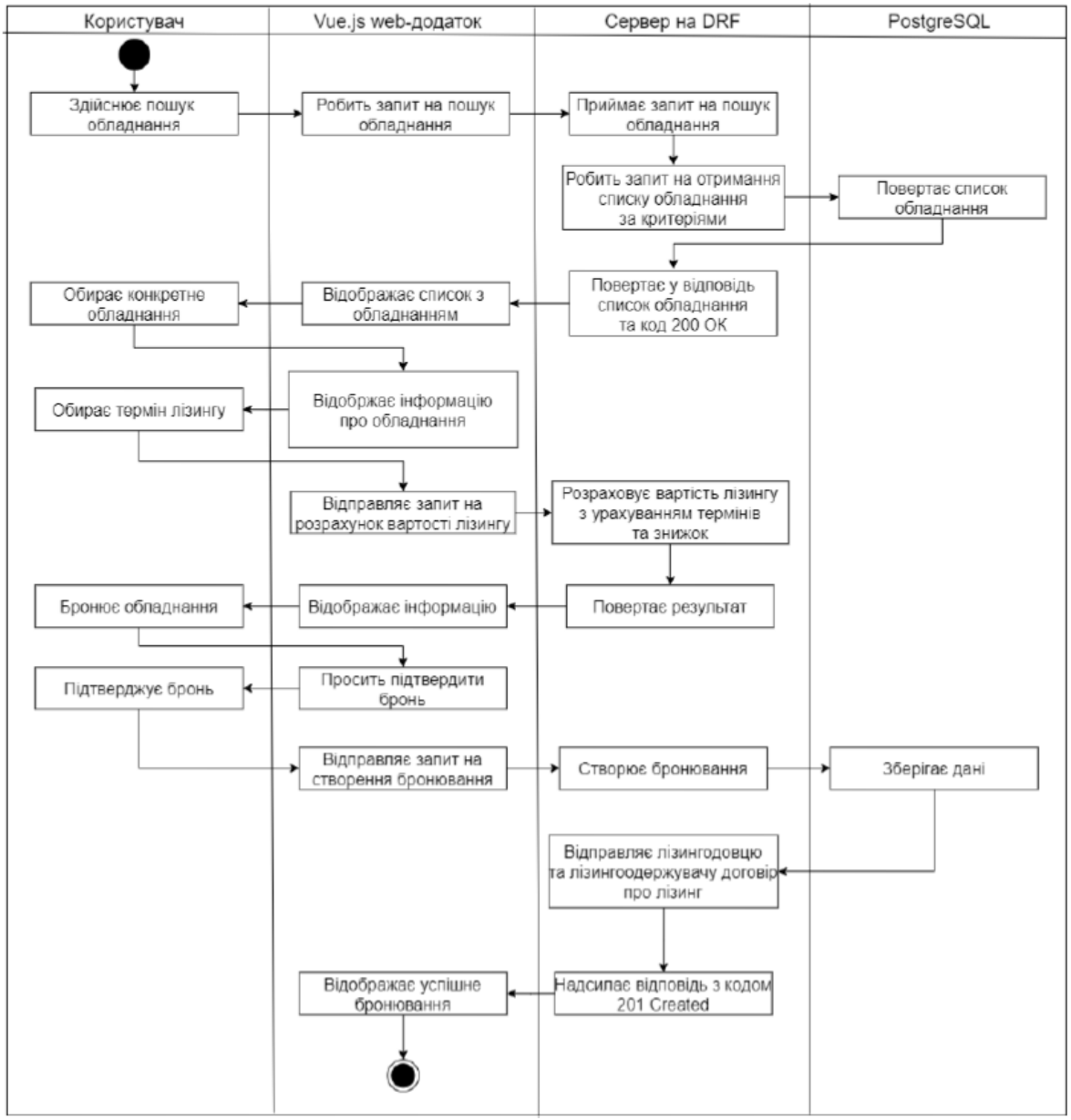


Рисунок 2.10 – Діаграма активності для другого сценарію

2.7 Представлення даних ІС (Data View)

Дані будуть зберігатися в реляційній базі даних. У якості системи керування базами даних виступає PostgreSQL. СКБД використовує транзакційний рівень ізоляції Read Commited. Взаємодія з базою даних (БД) проходить завдяки TCP/IP протоколу. Для встановлення зв'язку з БД необхідно вказати хост, порт, логін та пароль, а також назву бази даних.

На рис. 2.11 зображена схема бази даних з вказанням сутностей та їх взаємозв'язками. Звідси видно, що система має 6 сутностей:

– Equipment – сутність, яка зберігає інформацію про обладнання. Ця сутність пов'язана з сутностями Category та User зв'язком один-до-багатьох. У якості первинного ключа виступає поле id. Поля category_id та user_id виступають зовнішніми ключами на сутності Category та User відповідно;

– Image – це сутність, що представляє фотографії обладнання. Сутність буде зберігати шлях до зображення, а саме зображення буде зберігати на диск. Image пов'язана з сутністю Equipment зв'язком один-до-багатьох. Поле id виступає первинним ключем, а equipment_id – зовнішнім ключем;

– Category – це сутність, яка представляє доступні категорії у системі. У ній поле id виступає первинним ключем;

– User – це сутність, що представляє зареєстрованих користувачів системи. У ній поле id виступає первинним ключем;

– Order – ця сутність, яка представляє бронь користувача. Вона пов'язана з сутністю Equipment зв'язком один-до-багатьох. Поле id виступає первинним ключем, а equipment_id – зовнішнім ключем;

– Statistic – ця сутність історію статистики користувача. Вона пов'язана з сутністю User зв'язком один-до-багатьох. Поле id виступає первинним ключем, а user_id – зовнішнім ключем.

Для полегшення взаємодії з базою даних, використовується вбудована Django ORM. Тому домені об’єкти будуть відповідати та мати однакові атрибути з сутностями бази даних.

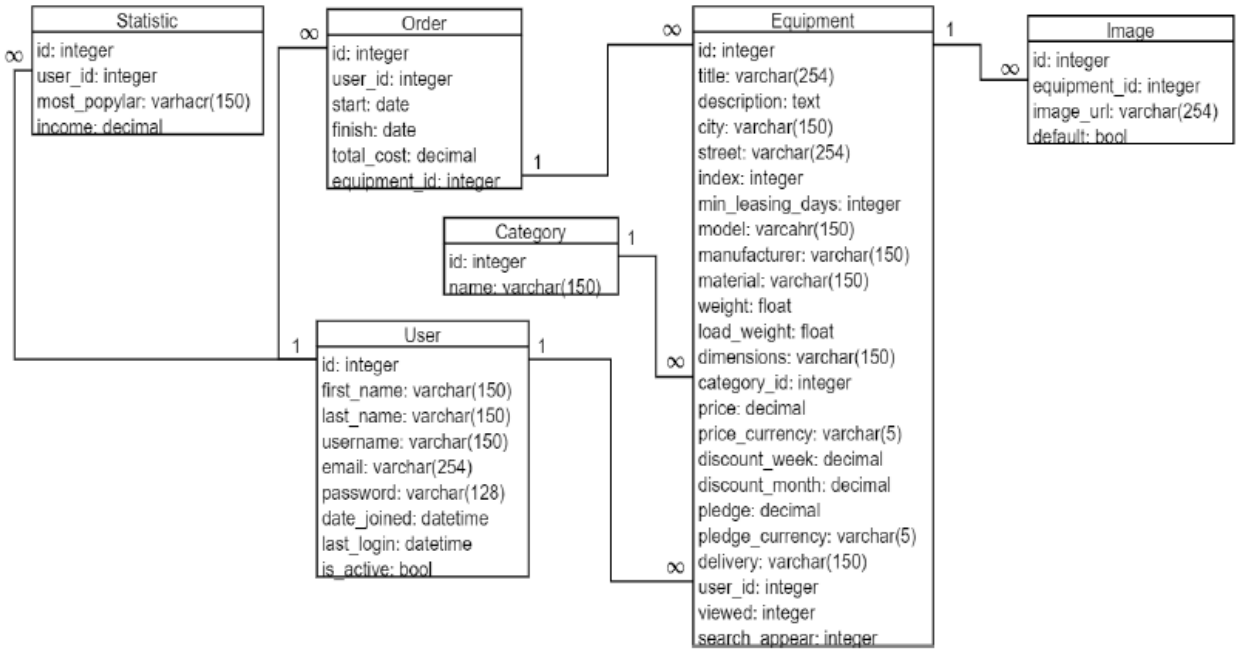


Рисунок 2.11 – Схема бази даних

2.8 Висновки до другого розділу

Отже, в даному розділі кваліфікаційної роботи були сформульовані мета та головні задачі інформаційної системи. Таким чином мета полягає у створенні інтернет-майданчика, на якому зможуть зустрітися ті люди, кому потрібно медичне обладнання, і ті люди, у кого воно наявне і можуть передати його в лізинг. Даний процес має бути зручним і швидким для обох типів людей.

Було низку діаграм, які представляють слої ІС, процеси в ній, дані, комунікацію, логічне уявлення системи та її розгортання.

Було визначено архетип системи та визначено вимоги забезпечення безпеки системи. Для уявлення графічного інтерфейсу було розроблено мо-

капи для кожної сторінки та побудовано діаграму переходів між ними. Останнім кроком було визначено технології, за допомогою яких буде проводитись розробка.

3 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Уявлення про структуру проекту

Розглянемо структуру web-додатку, яка зображена на рисунку 3.1. З рисунку можна побачити, що проект складається з таких основних директорій та файлів:

− node_modules – директорія, яка містить усі встановлені бібліотеки;

− public – директорія, яка містить головний файл HTML та іконку проекту;

− .gitignore – файл, який указує git не включати у коміти обрані файли або папки/директорії;

− src – директорія, яка виступає у якості головної папки проекту у якій знаходиться код проекту.

Директорія src містить у собі інші директорії та файли:

− api – ця директорія містить файл index.js, у якому описано класи для взаємодії з API сервера;

− assets – ця директорія містить у собі усі картинки, логотипи та іконки;

− components – ця директорія містить однофайлові компоненти, складається додаток. Дані компоненти можна використовувати повторно в будь-якому місці проекту;

− models – ця директорія містить файл index.js, який містить визначення класів-моделей проекту. Ці класи необхідні для полегшення взаємодії різних частин проекту;

− router – дана директорія містить файл index.js, який визначає URI по яким можна звернутися до конкретного компоненту або сторінки;

− store – ця директорія містить файл index.js, який визначає об’єкт store. Даний об’єкт надається бібліотекою Vuex та призначений для зберігання стану додатку та синхронізації даних;

− App.vue – головний однофайловий компонент, який задає дочірнім компонентам загальні стилі, частину поведінки, управляє переходами між компонентами та зв’язує їх з store, що надається завдяки Vuex;

− main.js – головний файл, який містить скрипт створення екземпляру Vue.js додатка та його налаштування.

Рисунок 3.1 – Структура web-додатку

Далі розглянемо структуру серверу, яка зображена на рис. 3.2. Видно, що структура сервера складається з наступних директорій та файлів:

− settings – це python пакет, що містить модулі налаштування проекту. Модуль settings.py містить загальні налаштування проекту. Модуль asgi.py містить створення та налаштування асинхронного серверу додатків (у випадку його використання). Модуль wsgi.py містить створення та налаштування синхронного серверу додатків (у випадку його використання). Модуль urls.py містить визначення маршрутів усього проекту;

− authentication – це пакет трактується, як django-додаток, і відповідає за представлення користувачів та реєстрацію/авторизацію;

− equipment – це пакет трактується, як django-додаток, і відповідає за представлення обладнання;

− notification – це пакет трактується, як django-додаток, і відповідає за відправку повідомлень користувачам;

− order – це пакет трактується, як django-додаток, і відповідає за створення бронювання обладнання;

− statistic – це пакет трактується, як django-додаток, і відповідає за представлення статистики;

− venv – ця директорія містить у собі встановлені фреймворки та бібліотеки;

− .gitignore – файл, який указує git не включати у коміти обрані файли або папки/директорії;

− manage.py – це утиліта, яка надається Django, для управління проектом (запуск серверу додатків, створення міграцій бази даних та інше);

− requirements.txt – це текстовий файл, який містить перелік усіх необхідних бібліотек та фреймворків з вказанням їх версій. Використовується для швидкого встановлення залежності при розгортанні серверу.

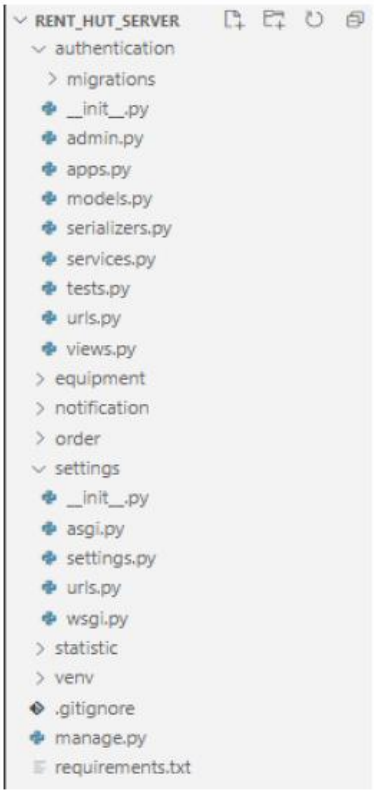


Рисунок 3.2 – Структура серверу

Пакети authentication, equipment, notification, order та statistic мають однакову структуру модулів:

− admin.py – цей пакет призначений для відображення моделей в адміністративній панелі Django;

− apps.py – цей пакет містить конфігурацію конкретного django-додатку;

− models.py – цей пакет містить визначення сутностей (моделей) бази даних пов’язаних з конкретним django-додатком;

− serializers.py – пакет, який містить серіалізатори;

− services.py – пакет, який визначає класи-сервіси для конкретної моделі;

− tests.py – пакет з модульними тестами;

− urls.py – пакет, що визначає маршрути конкретного додатку;

− views.py – пакет, що визначає класи-контролери.

3.2 Уявлення про класи ІС

На рис. 3.3 зображено діаграму класів серверу. Оскільки серверна частина складається з великої кількості класів, було прийнято рішення зобразити на діаграмі лише деяку частину.

З рисунку можна побачити, що для вибору алгоритму обробки помилки, було реалізовано шаблон проектування стратегія. Даний шаблон реалізують такі класи: базовий клас стратегії IHttpErrorHandlerStrategy, його підкласи Http400BadRequestHandlerStrategy, Http401Unauthenticated-HandlerStrategy, Http-403ForbiddenHandlerStrategy, Http404NotFound-HandlerStrategy, Http405Method-NotAllowedHandlerStrategy та Http503Service-UnavailableHandlerStrategy, які заміщують метод handle() та встановлюють свої алгоритми обробки помилок. Також шаблон допомагає реалізувати клас HttpErrorContext, який використо-вується для визначення який саме алгоритм необхідно виконати.

Класи EquipmentModel, ImageModel та AuthUserModel представляють відповідні таблиці з бази даних та надають можливість виконувати запити до бази даних. Увесь функціонал доступний завдяки тому, що перелічені вище класи наслідують функціонал класу Model, який у свою чергу надається Django ORM.

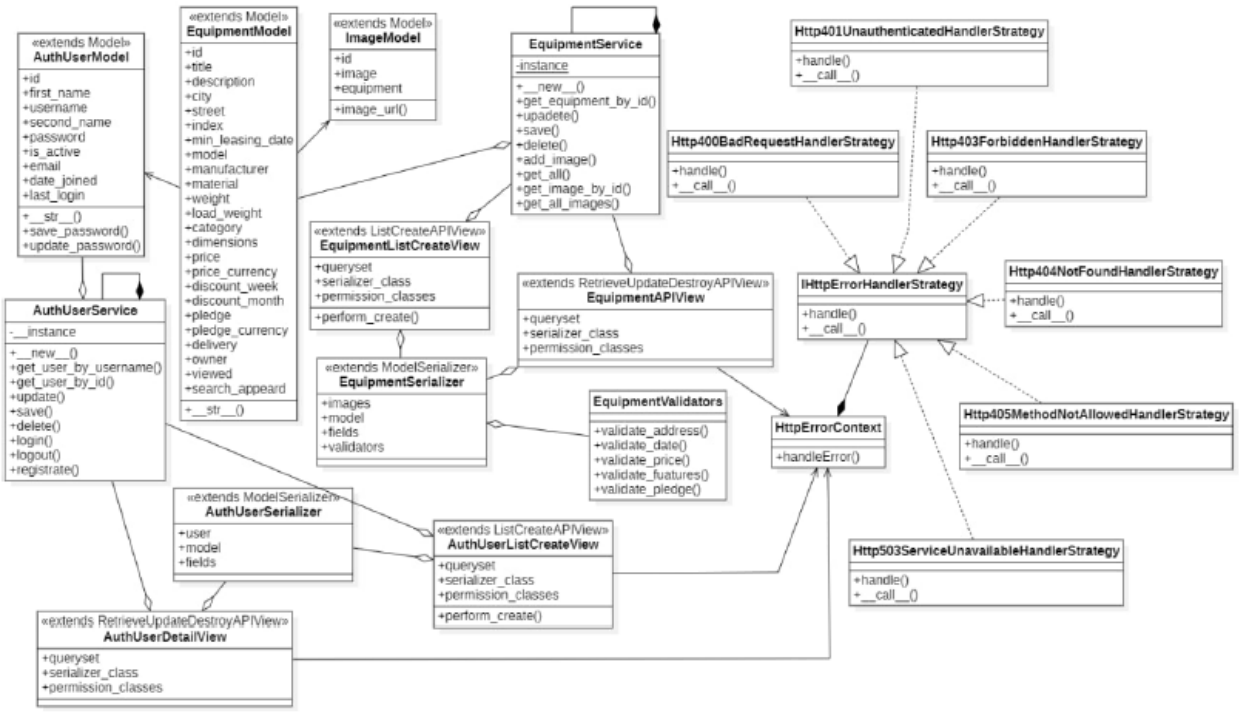


Рисунок 3.3 – Діаграма класів серверу

Клас AuthUserService реалізує бізнес логіку управління користувачами системи. Він реалізує шаблон сінглтон та має такі методи:

− __new__() – «магічний» метод, за допомогою якого реалізується шаблон сінглтон. Магічними називають ті методи, які автоматично викликаються інтерпретатором при виконанні відповідних умов. Наприклад, даний метод викликається, коли користувач намагається створити екземпляр класу. Метод працює наступним чином: якщо екземпляр існує, повертається посилання на нього, у іншому випадку екземпляр створюється, зберігаються на повертається користувачу;

− get_user_by_username(username) – метод, який повертає користувача з відповідним username;

− get_user_by_id(id) – метод повертає користувача з відповідним id;

− update(user) – метод оновлює інформацію про конкретного користувача;

− save(**kwargs) – метод зберігає в базу даних нового користувача;

− delete(user) – метод видаляє з бази даних конкретного користувача;

− login(username, password) – метод реалізує вхід користувача в систему;

− logout(username) – метод реалізує вихід користувача з системи;

− registrate(**kwargs) – метод реалізує реєстрацію користувача.

Клас EquipmentService відповідає за бізнес логіку управління обладнанням. Він реалізує шаблон проектування сінглтон. Має наступні методи:

− __new__() – метод допомагає реалізувати шаблон сінглтон;

− update(equipment) – метод оновлює інформацію про конкретне обладнання;

− save(equipment) – метод зберігає в базу даних нове обладнання;

− delete(equipment) – метод видаляє з бази даних конкретне обладнання;

− get_equipment_by_id(id) – метод повертає обладнання з відповідним id;

− get_image_by_id(id) – метод повертає фотографію обладнання за її id;

− add_image(**kwargs) – метод додає обладнанню нову фотографію;

− get_all() – метод повертає усе обладнання;

− get_all_images() – метод повертає усі фотографії обладнання.

Класи EquipmentSerializer та AuthUserSerializer відповідають за серіалізацію та десеріалізацію об'єктів відповідних класів. Вони наслідують функціонал базового класу ModelSerializer. Валідацію виконує клас EquipmentValidators.

Класи AuthUserDetailView та EquipmentAPIView наслідують функціонал класу RetrieveUpdateDestroyAPIView, який має багато методів, що забез-

печують автоматичну обробку відповідних запитів. Також перераховані класи відповідають за обробку запитів на оновлення, видалення та отримання інформації про окрему сутність бази даних.

Класи AuthUserListCreateAPIView та EquipmentListCreateAPIView наслідуються від базового класу ListCreateAPIView, який надає функціонал та допомагає підкласам обробляти запити без написання великої кількості коду. Наведені вище класи відповідають за обробку запитів на створення відповідних об’єктів та їх зберігання, а також отримання списку цих об’єктів.

Далі розглянемо алгоритм роботи деяких методів класу AuthUserService. На рис. 3.4 зображено алгоритм роботи методу __new__(). Даний метод допомагає реалізувати шаблон сінглтон. Він перевіряє наявність у класу поля instance, а у випадку його наявності – повертає значення цього поля. У випадку, коли поле не існує, тоді метод створює екземпляр цього класу, зберігає його та повертає.

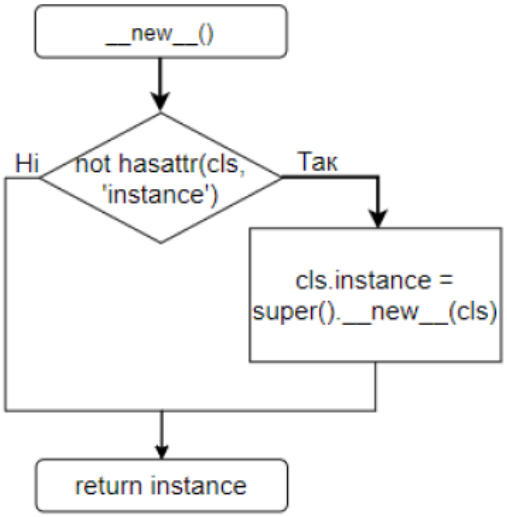


Рисунок 3.4 – Алгоритм роботи методу __new__()

На рис. 3.5 зображено роботу методу registrate(). Метод отримує усі необхідні дані та викликає метод save(), тим самим забезпечуючи реєстрацію у системі.

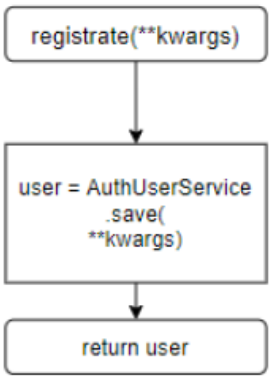


Рисунок 3.5 – Алгоритм роботи метода registrate()

На рис. 3.6 зображено алгоритм роботи метода login(). Даний метод отримує логін та пароль. За логіном він знаходить відповідного користувача та перевіряє співпадіння хешів паролів. Якщо вони співпали, то метод створює для користувача токен та повертає його. У іншому випадку повертає порожнє значення.

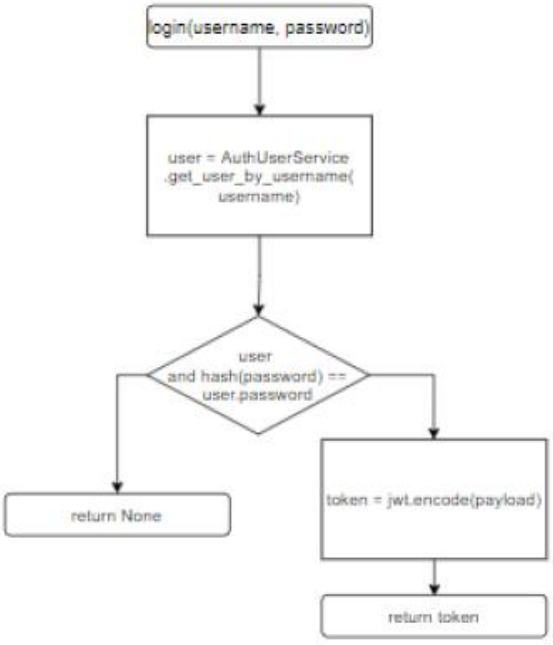


Рисунок 3.6 – Алгоритм роботи метода login()

На рис. 3.7 зображено алгоритм роботи метода get_user_by_username(). Метод викликає у класу AuthUserModel метод get() з параметром username. У

відповідь метод get() повертає необхідного користувача. Далі метод get_user_by_username() повертає отриманий результат.

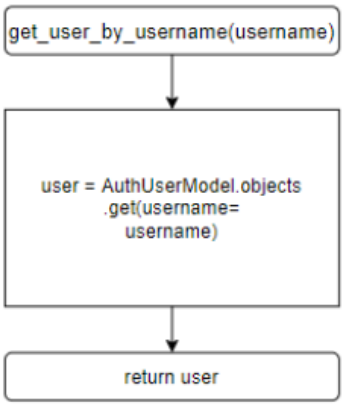


Рисунок 3.7 – Алгоритм роботи метода get_user_by_username()

На рис. 3.8 зображено роботу метода logout(). Даний метод отримує користувача за його username та встановлює полю token значення None, тим самим забезпечуючи вихід із системи.

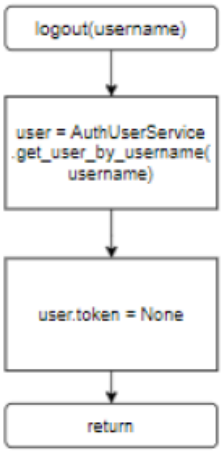


Рисунок 3.8 – Алгоритм роботи методу logout()

Далі розглянемо діаграму класів web-додатку. З рис. 3.9 видно, що було використано шаблон проектування фабричний метод. Даний шаблон використовується для створення та швидкої зміни формату валют. Базовий клас CurrencyFormatter визначає метод formatCurrency(), який відповідає за представлення валюти. Його підкласи UACurrencyFormatter, UECurrencyFormatter

та UACurrencyFormatter реалізують представлення відповідної валюти. Клас CurrencyFactory відповідає за створення конкретного формату валюти.

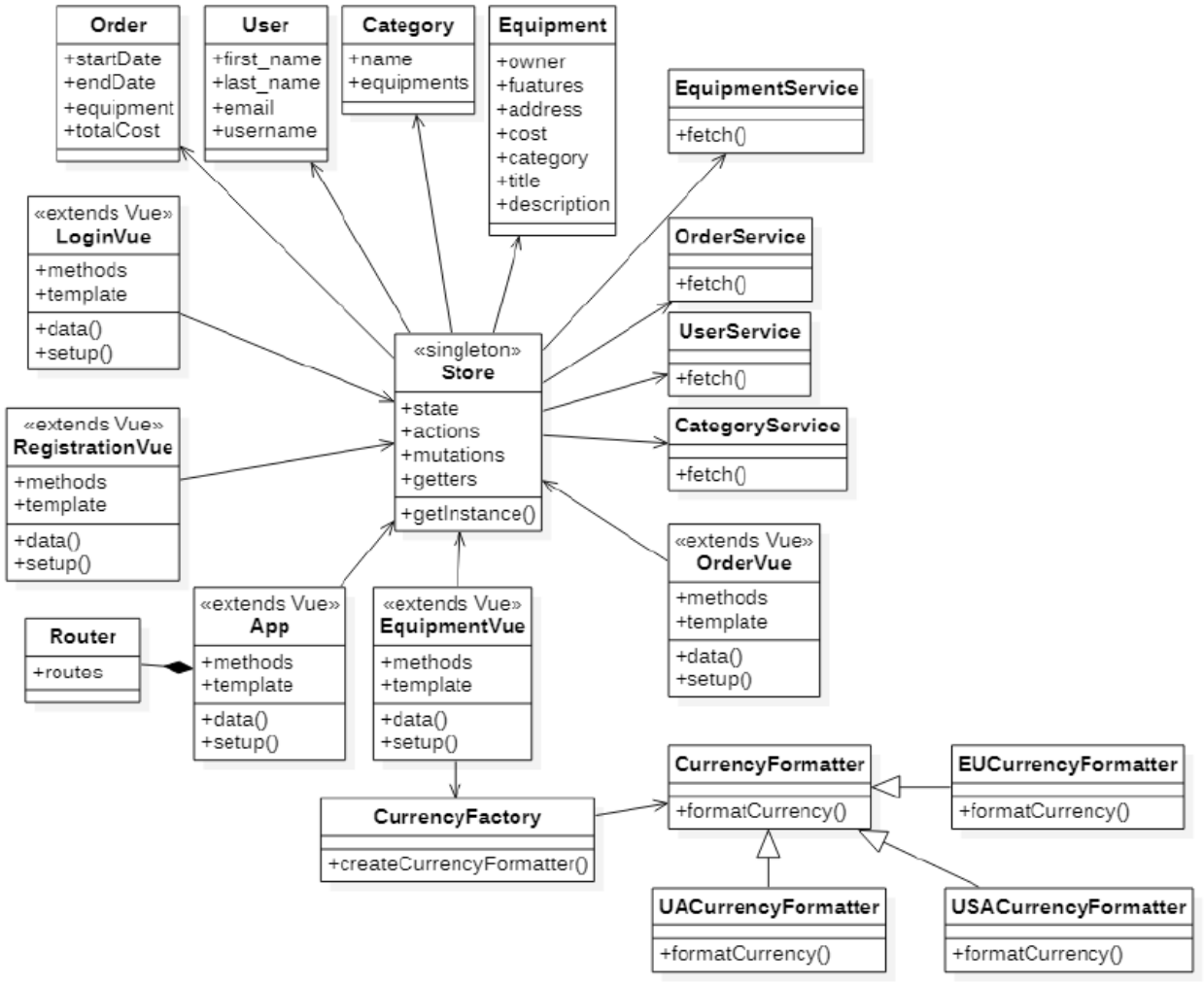


Рисунок 3.9 – Діаграма класів web-додатку

Клас Store реалізує шаблон сінглтон. Даний клас відповідає за зберігання стану додатку та виступає у якості моста, який з’єднує різні компоненти та сервіси. Клас має метод getInstance(), який повертає новий або вже існуючий екземпляр цього класу. Клас має 4 поля, які відповідають за конкретну взаємодію. Поле state представляє собою об’єкт у якому визначені усі загальні змінні та їх значення. Це поле відповідає за зберігання стану. Поле actions відповідає за асинхронне виконання операцій, взаємодію з сервісами

(UserService, OrderService та інші) та отримання даних з серверу. Дане поле представлене у вигляді об’єкту у якому визначені відповідні методи. Поле mutations представляє собою об’єкт у якому визначені методи для синхронної роботи та зміни state додатку. З цих методів не можна визивати асинхронні методи з actions. Поле getters представляє собою об’єкт, що визначає гетери, які повертають необхідні зміни зі state. Можна зауважити незвичний вигляд класу Store, а саме те, що за поведінку відповідають 2 поля, що визначають об’єкти з необхідними методами. Таке дивне архітектурне рішення надається ідеологією Vuex.

Класи Order, User, Equipment та Category представляють конкретну сутність з бази даних та призначені для зручної роботи з ними. Класи EquipmentService, UserService, OrderService та CategoryService мають один метод fetch(), який відповідає за відправку запитів на сервер по відповідним URL.

Клас App представляє собою головну компоненту, яка має у собі усі інші. Класи RegistrationVue, Login_View, Equipment_View та Order_View також представляють собою компоненти, які відповідають за відображення конкретної частини додатку та реалізацію його функціоналу. Усі перелічені в даному абзаці класи наслідують функціонал класу Vue. Вони мають по 2 основних поля та методи. Поле data у компонентів представлене у вигляді функції (тобто це метод), яка повертає об’єкт з визначенням усіх змінних компоненти. Це необхідно для того, щоб кожен екземпляр компоненти мав власну копію об’єкта з даними, що повертається функцією. Метод setup() викликається автоматично коли компонента створюється і відповідає за налаштування компонентів. Поле template містить визначення HTML-шаблону компоненти. Поле methods представляє собою об’єкт, у якому визначені методи управління шаблоном.

ВИСНОВКИ

У процесі написання кваліфікаційної роботи був, розроблений web-сервіс для лізингу медичного обладнання. Дана система складається з 2 частин – це сервер та web-додаток. Даний продукт був розроблений для людей, кому потрібно медичне обладнання, та людей, у кого воно наявне і можуть передати його в лізинг.

У першому розділі даної роботи було визначено та проаналізовано предметну область. Були розглянуті аналоги та проведене порівняння. Далі були обрані технології. Клієнт був написаний на мові програмування JavaScript. Сервер був написаний на мові програмування Python. Також були обрані архітектурні шаблони.

У другому розділі були сформульовані мета та головні задачі інформаційної системи. Було побудовано низку діаграм, які представляють слої ІС, процеси в ній, дані, комунікацію, логічне уявлення системи та її розгортання. Були розроблені мокапи для уявлення графічного інтерфейсу.

У третьому розділі була представлена структура кожної з частин проекту, а також побудовані до них діаграми класів. Загалом було здійснено реалізацію системи. Були розглянуті питання управління програмним кодом з використанням системи контролю версій Git.

Таким чином, мета та усі поставлені до даної роботи завдання були успішно виконані.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. EasyPro – Офіційний сайт. URL: https://easypro.com/ (дата звернення: 18.05.2022).
2. Rent in Hand – Офіційний сайт. URL: https://rentinhand.ru/ru (дата звернення: 18.05.2022).
3. Rentman – Офіційний сайт. URL: https://rentman.io/ru (дата звернення: 18.05.2022).
4. Самые важные архитектурные шаблоны, которые нужно знать : стаття. URL: https://bit.ly/3gkzOgU (дата звернення: 19.05.2022).
5. Многослойная архитектура : стаття. URL: https://bit.ly/2TkNRuI (дата звернення: 19.05.2022).
6. MVC – модель-представление-контроллер : стаття. URL: https://bit.ly/3542MMC (дата звернення: 19.05.2022).
7. Паттерн MVVM: стаття. URL: https://bit.ly/3g9bwYc (дата звернення: 19.05.2022).
8. Фабричний метод: стаття. URL: https://bit.ly/3zglfn5 (дата звернення: 20.05.2022).
9. Сінглтон: стаття. URL: https://bit.ly/3iwTy3E (дата звернення: 20.05.2022).
10. Стратегія: стаття. URL: https://bit.ly/2RFbD4f (дата звернення: 20.05.2022).
11. Флэнаган Дэвид. JavaScript. Полное руководство, 7-е издание. – СПб.: ООО «Диалектика», 2022. – с. 720.
12. Coffeescript – Офіційний сайт. URL: https://bit.ly/2Suv684 (дата звернення: 20.05.2022).
13. TypeScript – Офіційний сайт. URL: https://bit.ly/3bOzCET (дата звернення: 20.05.2022).

14. Рейтинг мов програмування 2022 : стаття. URL: https://bit.ly/3i4TKa7 (дата звернення: 21.05.2022).
15. Visual Studio Code – Офіційний сайт. URL: https://code.visualstudio.com/docs (дата звернення: 21.05.2022).
16. PyCharm Features – Офіційний сайт. URL: https://bit.ly/3yA44MV (дата звернення: 21.05.2022).
17. WebStorm Features – Офіційний сайт. URL: https://bit.ly/3bSNpum (дата звернення: 21.05.2022).
18. Django Project – Офіційний сайт. URL: https://www.djangoproject.com/ (дата звернення: 22.05.2022).
19. Twisted FAQ – Офіційний сайт. URL: https://bit.ly/3ffz1hQ (дата звернення: 22.05.2022).
20. Vue 3 – Офіційний сайт. URL: https://v3.vuejs.org/ (дата звернення: 22.05.2022).
21. Svelte – Офіційний сайт. URL: https://svelte.dev/ (дата звернення: 22.05.2022).
22. React – Офіційний сайт. URL: https://ru.reactjs.org/ (дата звернення: 22.05.2022).

ДОДАТОК А
МОКАПИ ГРАФІЧНОГО ІНТЕРФЕЙСУ

Рисунок А.1 – Мокап сторінки управління обладнанням

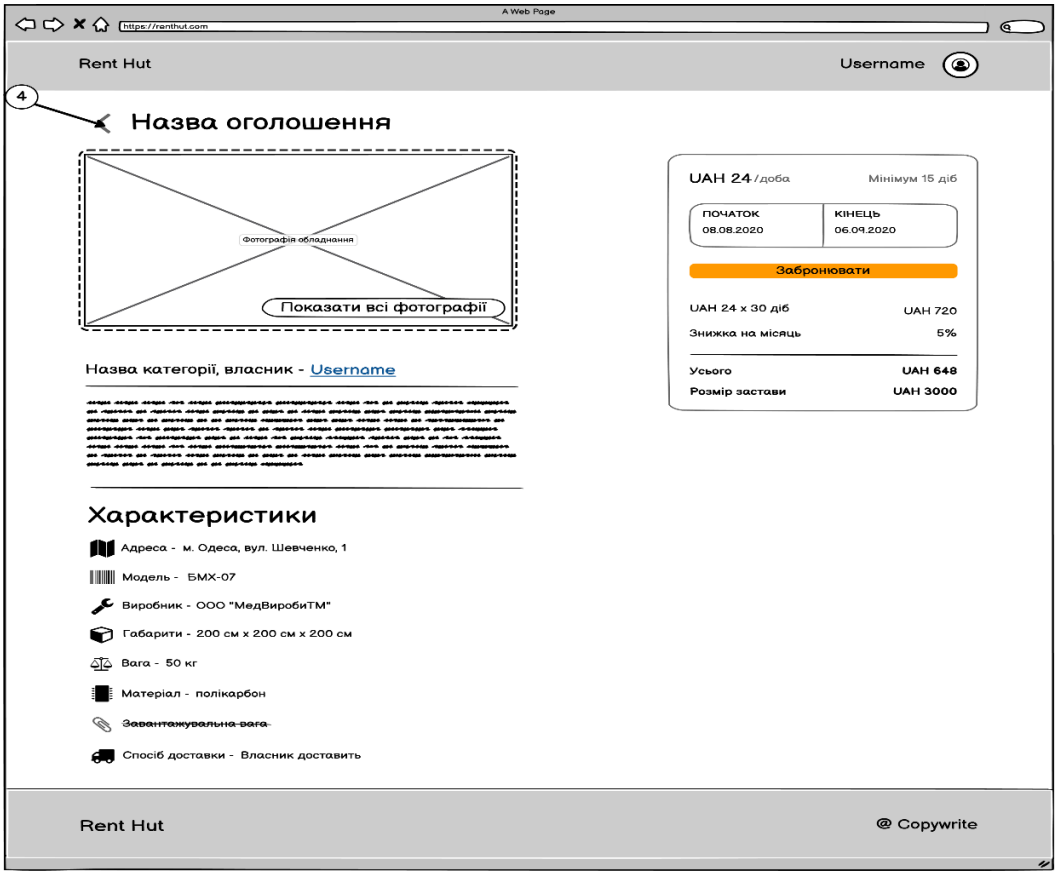


Рисунок А.2 – Мокап сторінки з детальною інформацією про обладнання

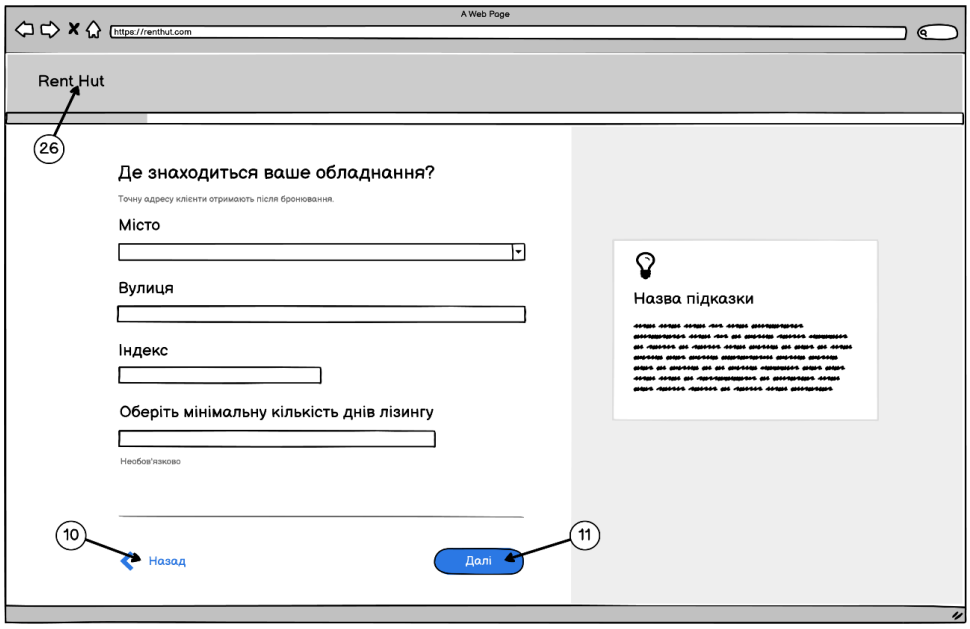


Рисунок А.3 – Мокап сторінки додавання обладнання з формою вказання адреси

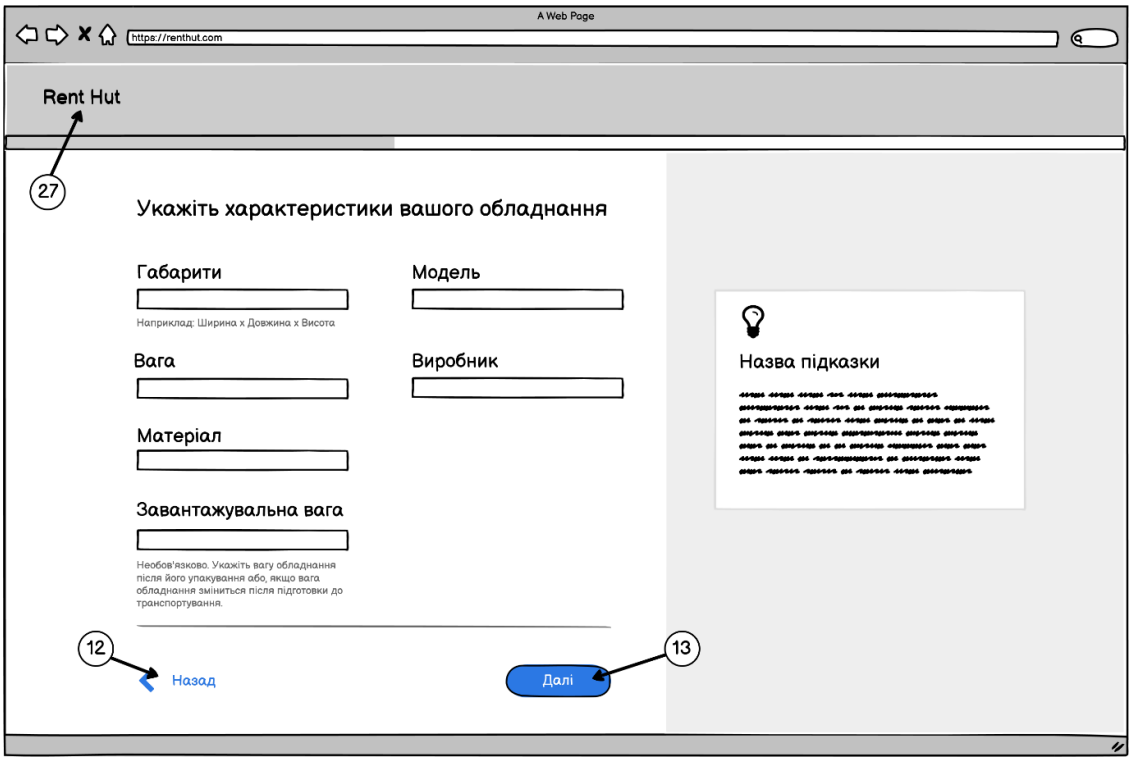


Рисунок А.4 – Мокап сторінки додавання обладнання з формою вказання характеристик обладнання

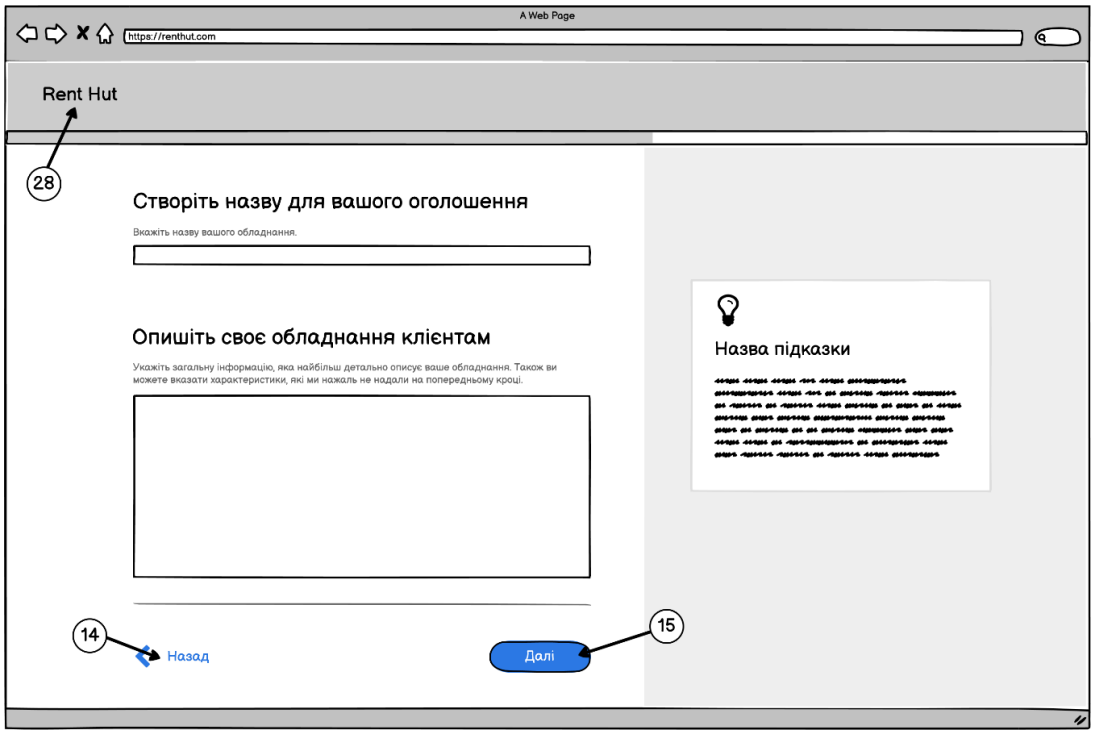


Рисунок А.5 – Мокап сторінки додавання обладнання з формою вказання назви та опису

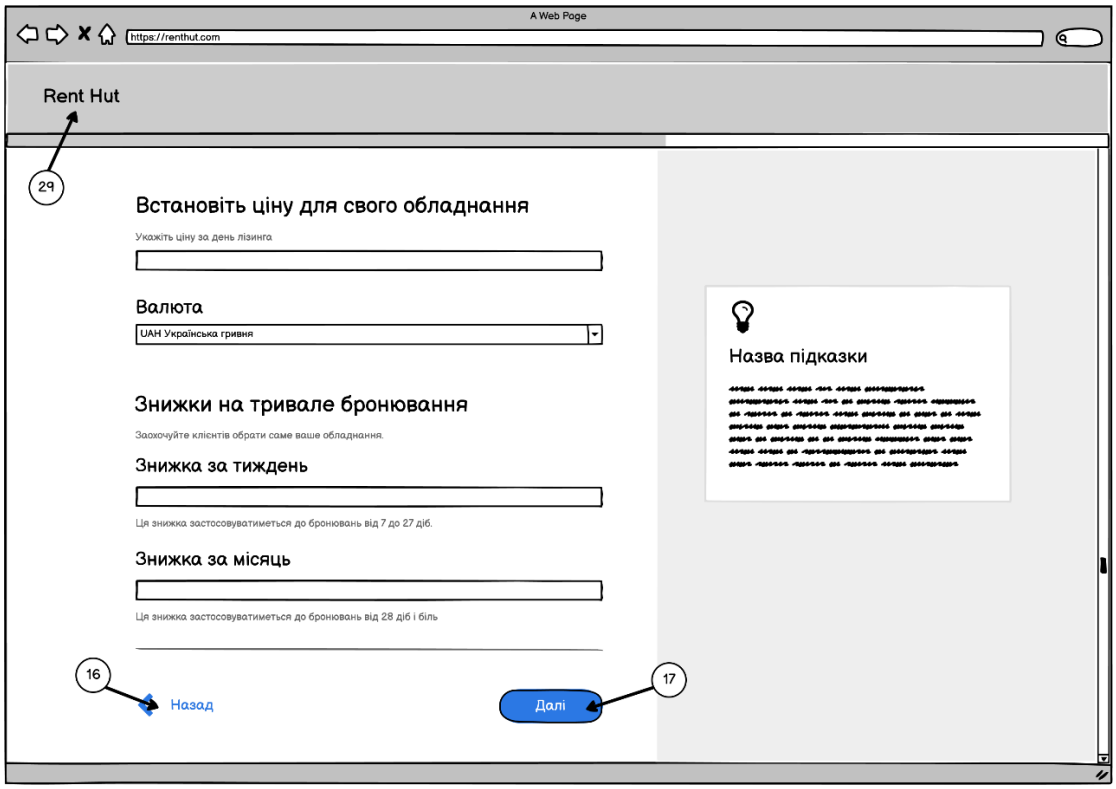


Рисунок А.6 – Мокап сторінки додавання обладнання з формою вказання цін та знижок

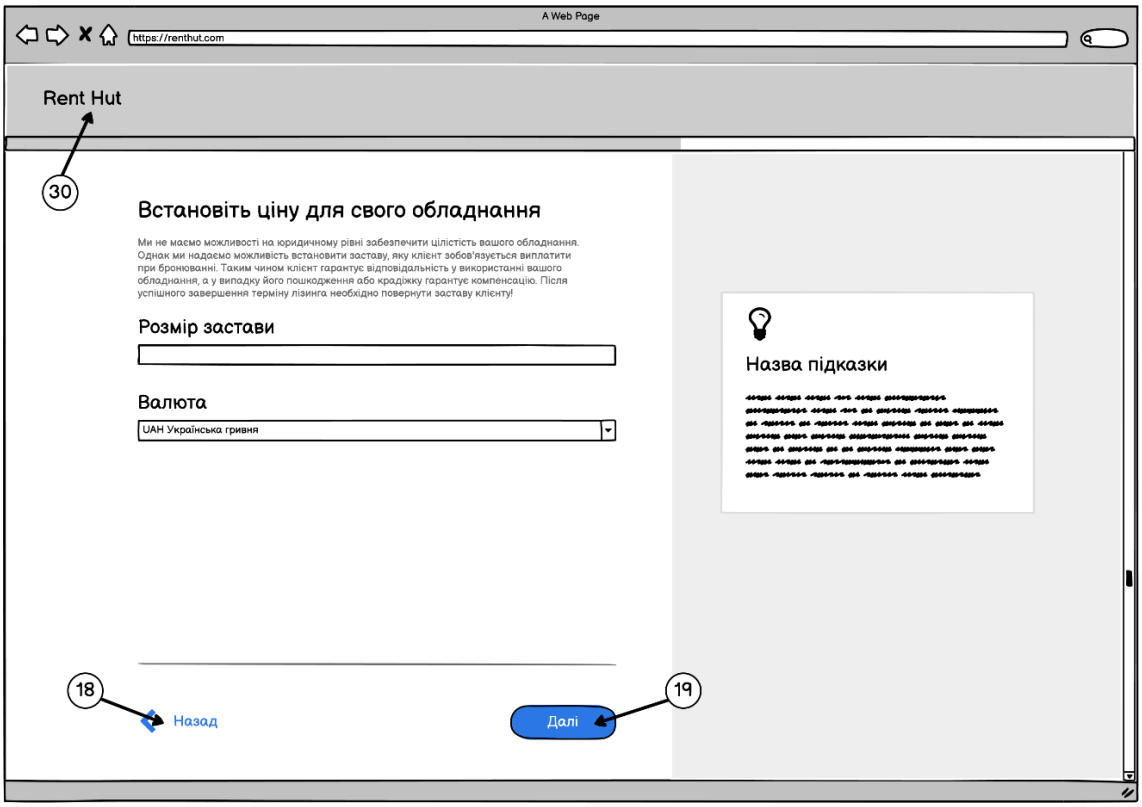


Рисунок А.7 – Мокап сторінки додавання обладнання з формою вказання розміру застави

Рисунок А.8 – Мокап сторінки додавання обладнання з формою додавання фотографій

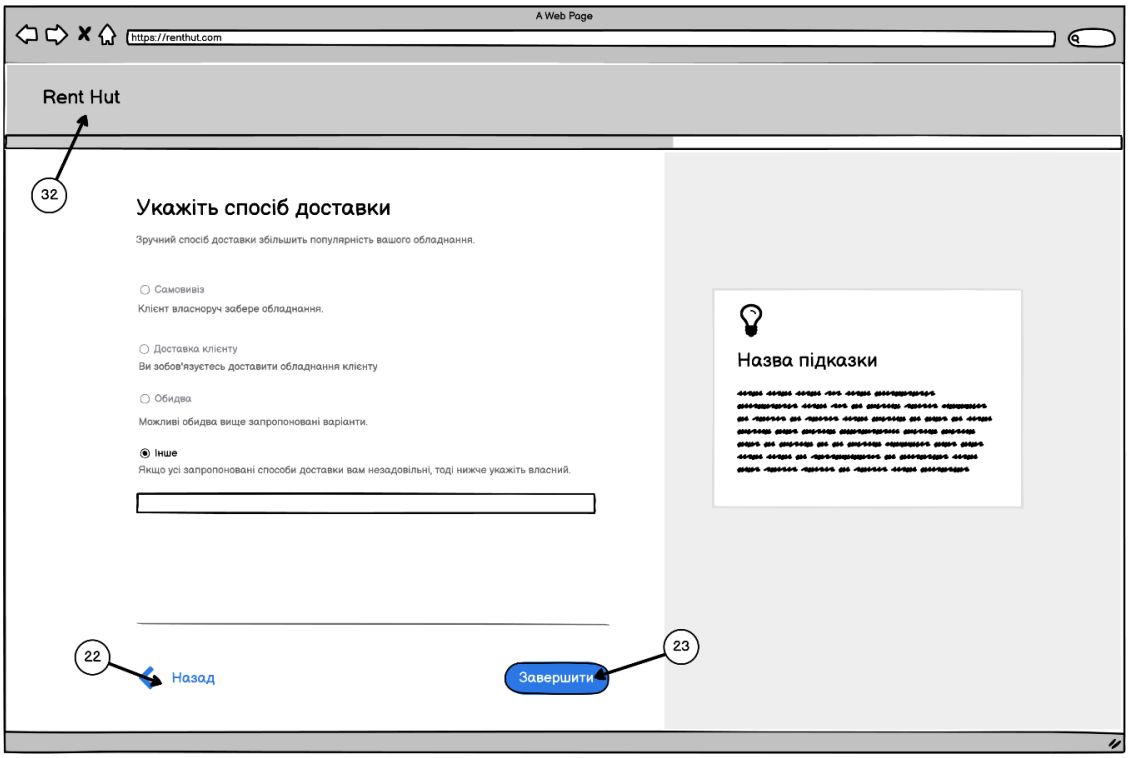


Рисунок А.9 – Мокап сторінки додавання обладнання з формою вказання способу доставки

ДОДАТОК Б
СКРИНШОТИ ДОДАТКУ

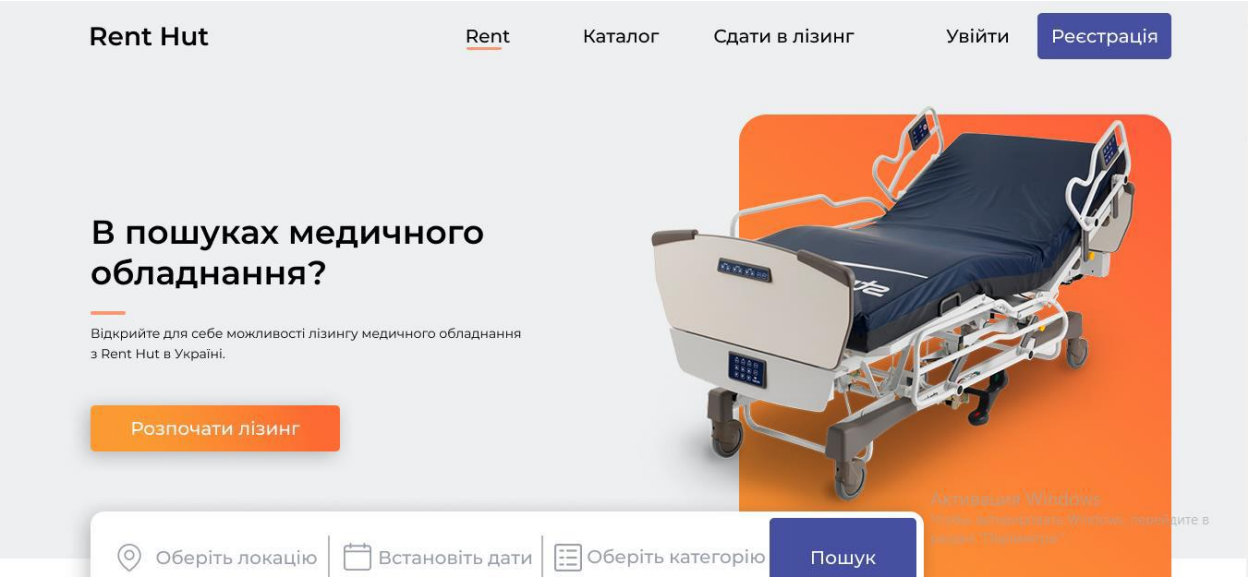


Рисунок Б.1 – Лендінг сторінка

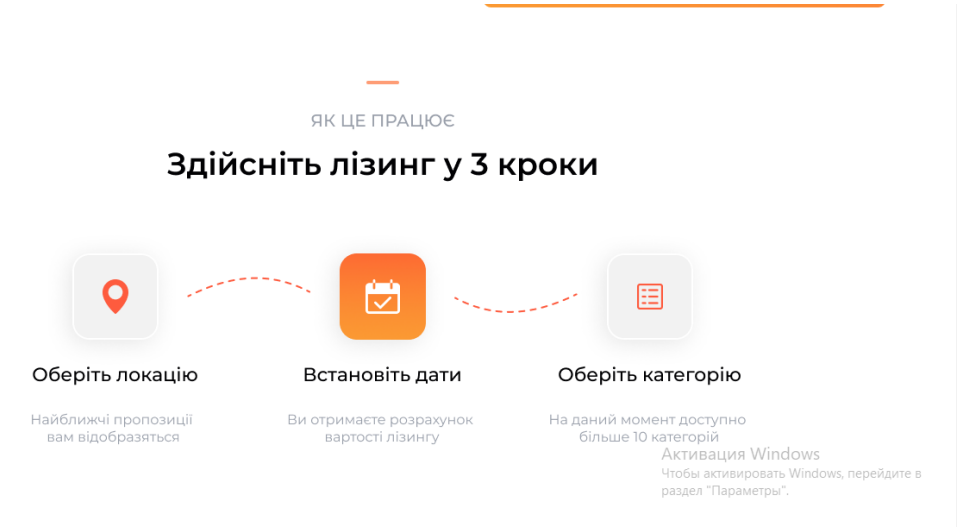


Рисунок Б.2 – Лендінг сторінка (продовження)

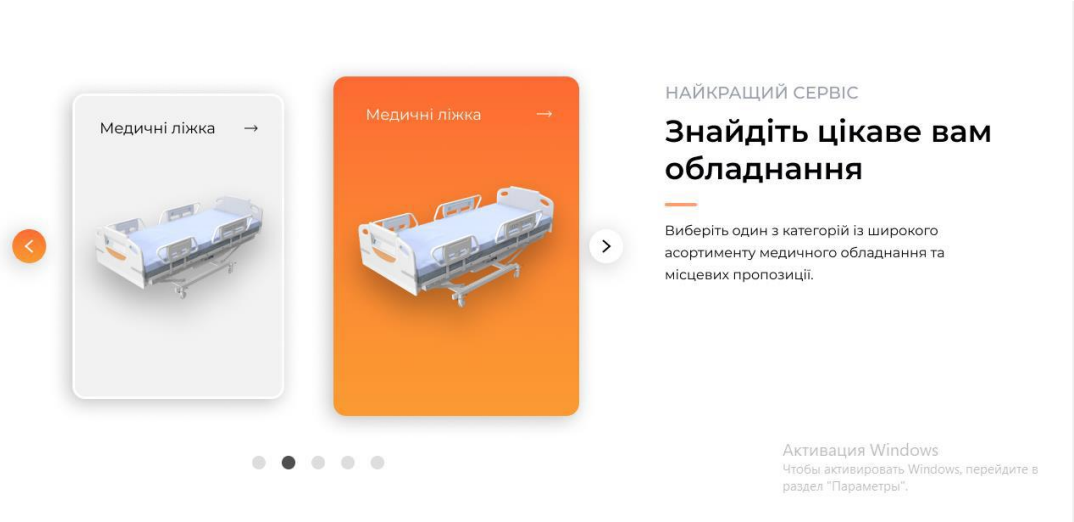


Рисунок Б.3 – Лендінг сторінка (продовження)

Рисунок Б.4 – Лендін сторінка (кінець)

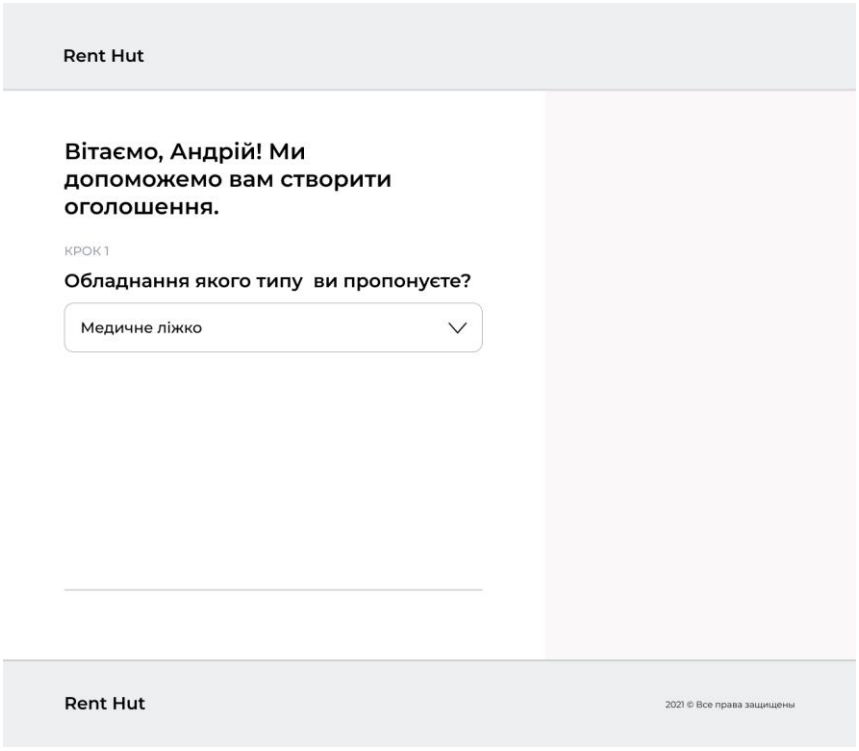


Рисунок Б.6 – Форма указання адреси

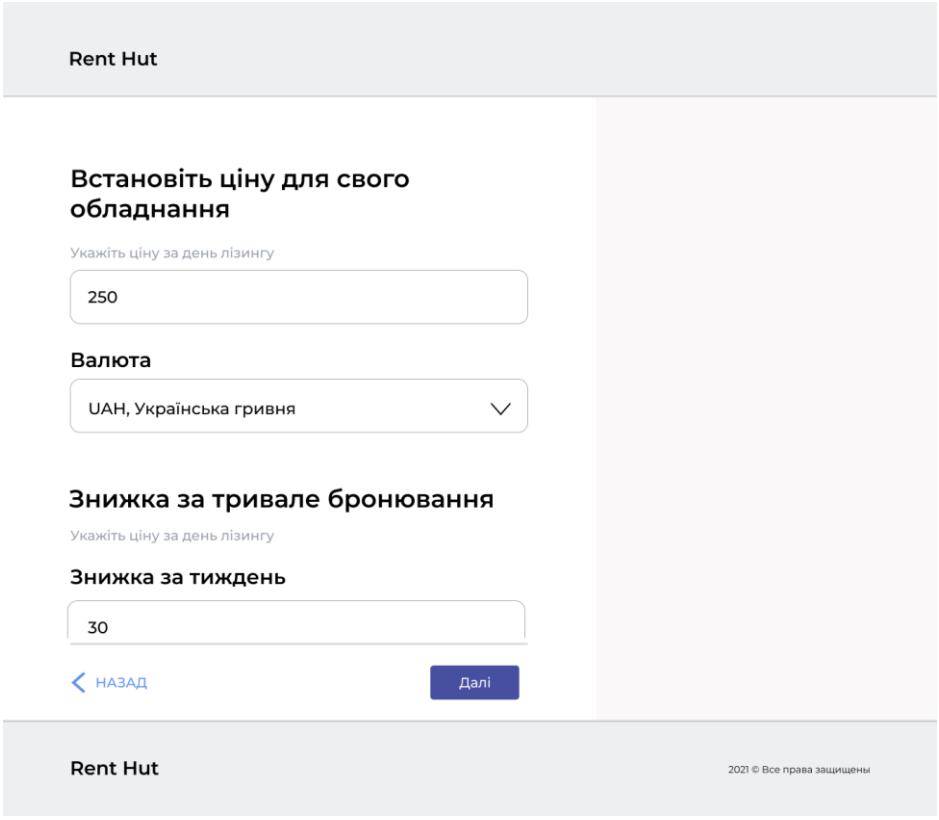


Рисунок Б.7 – Форма вказання ціни обладнання

Додаток В

КОД ПРОГРАМИ

Файл urls.py

```
from django.contrib import admin
from django.urls import path,include
urlpatterns = [
path('admin/', admin.site.urls),
#path to djoser end points
path('auth/', include('djoser.urls')),
path('auth/', include('djoser.urls.jwt')),
#path to our account's app endpoints
path("api/accounts/",include("accounts.urls"))
]
```

Файл models.py

```
from django.db import models
from django.contrib.auth.models import User
class AuthUserModel(models.Model):
us-
er=models.OneToOneField(User,on_delete=models.CASCADE,related_name=
"profile")
description=models.TextField(blank=True,null=True)
location=models.CharField(max_length=30,blank=True)
date_joined=models.DateTimeField(auto_now_add=True)
updated_on=models.DateTimeField(auto_now=True)
is_organizer=models.BooleanField(default=False)
def __str__(self):
return self.user.username
```

Файл signals.py

```
from django.contrib.auth.models import User 90
```

```
from django.db.models.signals import post_save
from django.dispatch import receiver
from .models import userProfile
@receiver(post_save, sender=User)
def create_profile(sender, instance, created, **kwargs):
if created:
userProfile.objects.create(user=instance)
from django.apps import AppConfig
class AccountsConfig(AppConfig):
name = 'accounts'
def ready(self):
import accounts.signals
```

Файл serializers.py

```
from rest_framework import serializers
from .models import userProfile
class userProfileSerializer(serializers.ModelSerializer):
user=serializers.StringRelatedField(read_only=True)
class Meta:
model=userProfile
fields=’__all__’
```

Файл views.py

```
from rest_framework.generics import
(ListCreateAPIView,RetrieveUpdateDestroyAPIView,)
from rest_framework.permissions import IsAuthenticated
from .models import userProfile
from .permissions import IsOwnerProfileOrReadOnly
from .serializers import userProfileSerializer 91
```

```
class UserProfileListCreateView(ListCreateAPIView):
queryset=userProfile.objects.all()
serializer_class=userProfileSerializer
permission_classes=[IsAuthenticated]
def perform_create(self, serializer):
user=self.request.user
serializer.save(user=user)
class userProfileDetailView(RetrieveUpdateDestroyAPIView):
queryset=userProfile.objects.all()
serializer_class=userProfileSerializer
permission_classes=[IsOwnerProfileOrReadOnly,IsAuthenticated]
from django.urls import include, path
from rest_framework.routers import DefaultRouter
from .views import UserProfileListCreateView, userProfileDetailView
urlpatterns = [
#gets all user profiles and create a new profile
path("all-profiles",UserProfileListCreateView.as_view(),name="all-
profiles"),
# retrieves profile details of the currently logged in user
path("profile/<int:pk>",userProfileDetailView.as_view(),name="profi
le"),
]
```

Файл license.py

```
from rest_framework.permissions import BasePermission,SAFE_METHODS
class IsOwnerProfileOrReadOnly(BasePermission):
def has_object_permission(self, request, view, obj):
if request.method in SAFE_METHODS:
return True
return obj.user==request.user 92
```

Файл test.py

```
class userProfileTestCase(APITestCase):
profile_list_url=reverse('all-profiles')
def setUp(self):
# создайте нового пользователя, отправив запрос к конечной точке
djoser
self.user=self.client.post('/auth/users/',data={'username':'mario',
'password':'i-keep-jumping'})
# получить веб-токен JSON для вновь созданного пользователя
re-
sponse=self.client.post('/auth/jwt/create/',data={'username':'mario
','password':'i-keep-jumping'})
self.token=response.data['access']
self.api_authentication()
def api_authentication(self):
self.client.credentials(HTTP_AUTHORIZATION='Bearer '+self.token)
# получить список всех профилей пользователей во время аутентифика-
ции пользователя запроса
def test_userprofile_list_authenticated(self):
response=self.client.get(self.profile_list_url)
self.assertEqual(response.status_code,status.HTTP_200_OK)
# получить список всех профилей пользователей, пока запрос пользо-
вателя не прошел проверку подлинности
def test_userprofile_list_unauthenticated(self):
self.client.force_authenticate(user=None)
response=self.client.get(self.profile_list_url)
self.assertEqual(response.status_code,status.HTTP_401_UNAUTHORIZED)
# проверьте, чтобы получить данные профиля аутентифицированного
пользователя
def test_userprofile_detail_retrieve(self):
response=self.client.get(reverse('profile',kwargs={'pk':1}))
# print(response.data)
self.assertEqual(response.status_code,status.HTTP_200_OK)
# заполнить профиль пользователя, который был автоматически создан
с использованием
```

```
сигналов
def test_userprofile_profile(self):
profile_data={'description':'I am a very famous game
character','location':'nintendo world','is_creator':'true',}
re-
sponse=self.client.put(reverse('profile',kwargs={'pk':1}),data=prof
ile_data)
print(response.data)
self.assertEqual(response.status_code,status.HTTP_200_OK)
```