

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ БРОНЮВАННЯ ІНДИВІДУАЛЬНИХ
ПРОЦЕДУР У СФЕРІ КРАСИ
SOFTWARE FOR BOOKING INDIVIDUAL BEAUTY TREATMENTS

Виконав: студент(ка) 4 курсу, групи КНТз-218

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

Зінченко К.Д.

(прізвище та ініціали)

Керівник Голуб Т.В.

(прізвище та ініціали)

Рецензент Терещенко Е.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Зінченко Катерини Дмитрівни
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення бронювання індивідуальних процедур у сфері краси
Software for Booking Individual Beauty Treatments

керівник проєкту (роботи) Голуб Тетяна Василівна, к.т.н.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від «27» квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) червень 2022 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Розробка програми. 4. Експлуатація та тестування програми.

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Голуб Т.В., доцент		
Нормоконтролер	Камінська Ж.К., асистент		

7. Дата видачі завдання «11» квітня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 тиждень	Розділ 1
3	Розробка архітектури програми	3 тиждень	Розділ 2
4	Розробка програми	4-5 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	6 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	7 тиждень	Додатки
7	Захист роботи	8 тиждень	

Студент(ка)

(підпис)

Зінченко К.Д.

(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Голуб Т.В.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 81 с., 21 рис., 2 табл., 3 дод., 10 джерел.

СФЕРА КРАСИ, ІНДИВІДУАЛЬНІ ПРОЦЕДУРИ, МАЙСТЕР, БРОНЮВАННЯ ПРОЦЕДУР, ВЕБЗАСТОСУНОК.

Об'єктом роботи є процес надання індивідуальних процедур у сфері краси. Предметом роботи є програмне забезпечення бронювання індивідуальних процедур у сфері краси.

Мета роботи – розробити програмне забезпечення бронювання індивідуальних процедур у сфері краси для підтримки роботи індивідуальних майстрів з клієнтами.

Матеріали, методи та технічні засоби: мова програмування Python, мультипарадигмне програмування, система керування базами даних MySQL.

Результати. Під час виконання роботи презентовано набір результатів, що характеризують виконані завдання, включаючи порівняння розробки з аналогом, вибір мови програмування, визначені вимоги до програми, створену базу даних, визначений порядок роботи програми та створені класи програми і як підсумковий результат – саму розроблену програму.

Висновки. Розроблено вебзастосунок для пошуку клієнтами майстрів, які здатні надати процедури у сфері краси у власних приміщеннях або індивідуально на умовах клієнта, оформлювати замовлення, а майстрам організовувати власний бізнес без участі великих гравців, керуючи розкладом власної роботи за допомогою керування замовленнями.

Галузь використання. Сфера краси, надання процедур індивідуальними майстрами як в своїх приміщеннях, так і у приміщеннях клієнта.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 81 pages, 21 figures, 2 tables, 3 appendixes, 10 sources.

SPHERE OF BEAUTY, INDIVIDUAL PROCEDURES, MASTER, BOOKING OF PROCEDURES, WEB APPLICATION.

The object of the work is the process of providing individual beauty treatments.

The subject of the work is the software for booking individual procedures in the field of beauty.

The purpose of the work is to develop software for booking individual beauty treatments to support the work of individual masters with clients.

Materials, methods and technical means: Python programming language, multi-paradigm programming, MySQL database management system.

Results. During the work a set of results is presented. The results characterize the performed tasks, including comparison of the development with analogue, choice of programming language, defined program requirements, created database, defined program operation procedure and created program classes and the developed program as a final result.

Conclusions. A web application was developed allowing clients to find masters who are able to provide beauty treatments in their own buildings or individually on the client's terms, to place orders, and masters to organize their own business without participation of big players, managing their own work schedule by order management.

Field of use. Sphere of beauty, provision of procedures by individual masters both in their own buildings and in the client's buildings.

ЗМІСТ

	С.
Перелік скорочень та умовних познач	7
Вступ	8
1 Аналіз предметної області	10
1.1 Індустрія краси	10
1.2 Аналіз аналогів	11
1.3 Висновки за розділом 1	15
2 Розробка архітектури програми	16
2.1 Вибір засобів розробки програми	16
2.2 Функціональні вимоги до програмного забезпечення	17
2.3 Проектування бази даних	19
2.4 Висновки за розділом 2	31
3 Розробка програми	32
3.1 Порядок роботи користувача з програмою	32
3.2 Опис реалізації програми	33
3.3 Висновки за розділом 3	39
4 Експлуатація та тестування програми	40
4.1 Призначення програми	40
4.2 Умови виконання програми	40
4.3 Тестування і опис принципів експлуатації програми	40
4.4 Висновки за розділом 4	44
Висновки	45
Перелік джерел посилання	46
Додаток А Технічне завдання	48
Додаток Б Текст програми	52
Додаток В Слайди презентації	75

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

БД – база даних;

ПЗ – програмне забезпечення.

ВСТУП

Індустрія краси – це величезний гігант, який не має жодних ознак уповільнення зростання загалом: загальний обсяг продажів у світі в 2020 році склав 483 мільярди доларів, очікується, що при річному темпі зростання 4,75% загальний дохід до 2025 року перевищить 716 мільярдів доларів, найбільша частка ринку індустрії краси в Азіатсько-Тихоокеанському регіоні – 46 %, далі йде Північна Америка з 24 % і Західна Європа з 18 %, за географічним розташуванням домінують Азіатсько-Тихоокеанський регіон та Північна Америка, на які припадає понад 70 % загального розміру ринку разом узятих [1]-[2].

Таким величезний обсяг ринку, його активний розвиток, яким характеризується зокрема і Україна, визначають те, що сфера краси потребує відповідних засобів для свого існування. З одного боку це засоби, які використовуються в процедурах, а з іншого – це засоби, необхідні для організації виконання цих процедур. До другої групи засобів належить відповідне програмне забезпечення (ПЗ), яке має підтримувати інформування клієнтів про процедури, бронювання роботи майстрів, а відповідно майстрам надавати можливість планування власної роботи. В умовах розвитку сфери загалом, тобто тоді, коли кількість компаній, майстрів збільшується, існує постійна потреба як мінімум нових гравців у використанні відповідного адекватного засобу.

Однак, окрім цього варто також відзначити, що в Україні в поточний момент сфера краси, як і багато інших сфер, дещо змінюється. Багато людей переміщується, компанії виявляються розділеними частково в межах країни, тобто одні працівники часто знаходяться в одних містах, інші – в інших, в цей час працівники можуть починати працювати з клієнтами, але не мають засобів виявлення своїх вмінь. До таких засобів у такому випадку можна віднести сторінки в соціальних мережах і повністю ручний індивідуальний підхід. Проте ПЗ, яке орієнтовано саме на таких індивідуальних майстрів, які

не зважаючи на обставини, продовжують працювати, є важливим інструментом для підтримки і подальшого розвитку сфери краси в Україні, а загалом і в інших країнах. Таке ПЗ може створити інструмент для пошуку майстрів і демонстрування майстрами власних вмінь у сфері послуг краси, що дозволить цим сторонам взаємодіяти без необхідності участі салонів краси.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Індустрія краси

Люди є не тільки соціальними, але і візуальними істотами, тому має сенс те, що люди історично були колективно зайняті красою: чиясь зовнішність – це перше, що люди помічають при зустрічі, сама природа краси була в центрі уваги естетики, основної галузі вивчення західної філософії, і займала уми багатьох філософів, від Платона до Канта, і слід додати, що люди користувалися косметикою з давніх часів: використання темної підводки єгиптянами присутнє в уяві кожного через їхні картини та божества, тоді як більшість знає, що вікторіанські жінки часто втрачали свідомість через використання кремів на основі свинцю для покращення кольору обличчя разом із звуженням поясів [3]-[4].

Індустрія краси – це сфера, що включає догляд за шкірою, косметичні процедури, догляд за волоссям, ароматичні та засоби особистої гігієни, проте за останній період вона, як не дивно це в розвитку за останнє десятиліття, притерпала певне падіння, спричинене пандемією в світі: так у 2020 році продажі засобів кольорування впали на 33 % в усьому світі, тоді як загальні роздрібні продажі в сфері краси впали на 15 %, але в минулому галузь була стійкою, і експерти прогнозують повернення до зростання протягом всього 2022 року [5].

Тільки в Сполучених Штатах Америки у сфері послуг краси працює понад 670 000 людей, і прогнози зростання робочих місць є швидшими за середній рівень, згідно з деякими дослідженнями зріст за 2016-2026 роки має складати 13 %, у 2017 році його вартість сфери становила 532,43 мільярди доларів, а до 2023 року очікується, що сфера досягне ринкової вартості в 805,61 мільярдів доларів. Звичайно, що це величезні цифри, а тому створення програмних інструментів, які дозволять активніше розвиватися сфері в певній країні або дозволять активно мати зиск з розвитку сфери, є важливим і актуальним, зважаючи на обсяги індустрії [3]-[6].

Індустрія краси є особливо антициклічною, настільки, що з неї випливає відомий економічний феномен, так званий ефект помади, – це явище, яке лише частково підтверджено, стверджує, що споживачі все ще будуть купувати предмети розкоші в умовах економічного спаду, віддаючи перевагу при цьому меншим продуктам, таким як помада, проте окрім цього останніми роками сфера краси загалом стрімко розвивається, і багато хто пов'язує це з ширшою тенденцією покоління приділяти увагу фізичному здоров'ю, міленіалів часто називають основними рушійними силами стрімкого зростання сегменту краси, деякі дослідники взагалі виділяють, що догляд за собою для покоління є подоланням складного політичного та економічного клімату [3].

1.2 Аналіз аналогів

Основним прототипом програми, розробка якої реалізується, є вебзастосунок Barb (рис. 1.1) [7]. Цей застосунок дає можливість пошуку процедур краси, при чому надавати ці процедури можуть як салони краси загалом, так і окремі майстри. Це є основною відмінністю цієї програми від інших, що дозволило саме її розглядати в якості певного прототипу розробки.

Вебсайт Barb надає можливість пошуку процедур за назвою, пошуку за назвою салону, за іменем майстра. Також надається перелік всіх процедур, розділених за категоріями. Обираючи певну процедуру, задаючи додатково фільтри, пов'язані з районом міста, станцією метро найближчою (для міст, в яких є метро) можна виконати пошук процедур, які виконуються саме в заданому місці. Окремо задається місто, в якому має бути виконаний пошук процедур. Всі міста українські. Усі позиції в результатах пошуку представлені оголошеннями, що включають інформацію про салон або майстра, інформацію про процедуру, пошук якої виконується, з можливістю переходу до повного переліку процедур та вартості цих процедур. За майстром визначаються його спеціальності, а після відкриття сторінки

майстра можна переглянути відгуки про цього майстра, які також формують середню оцінку, виставлену майстру.

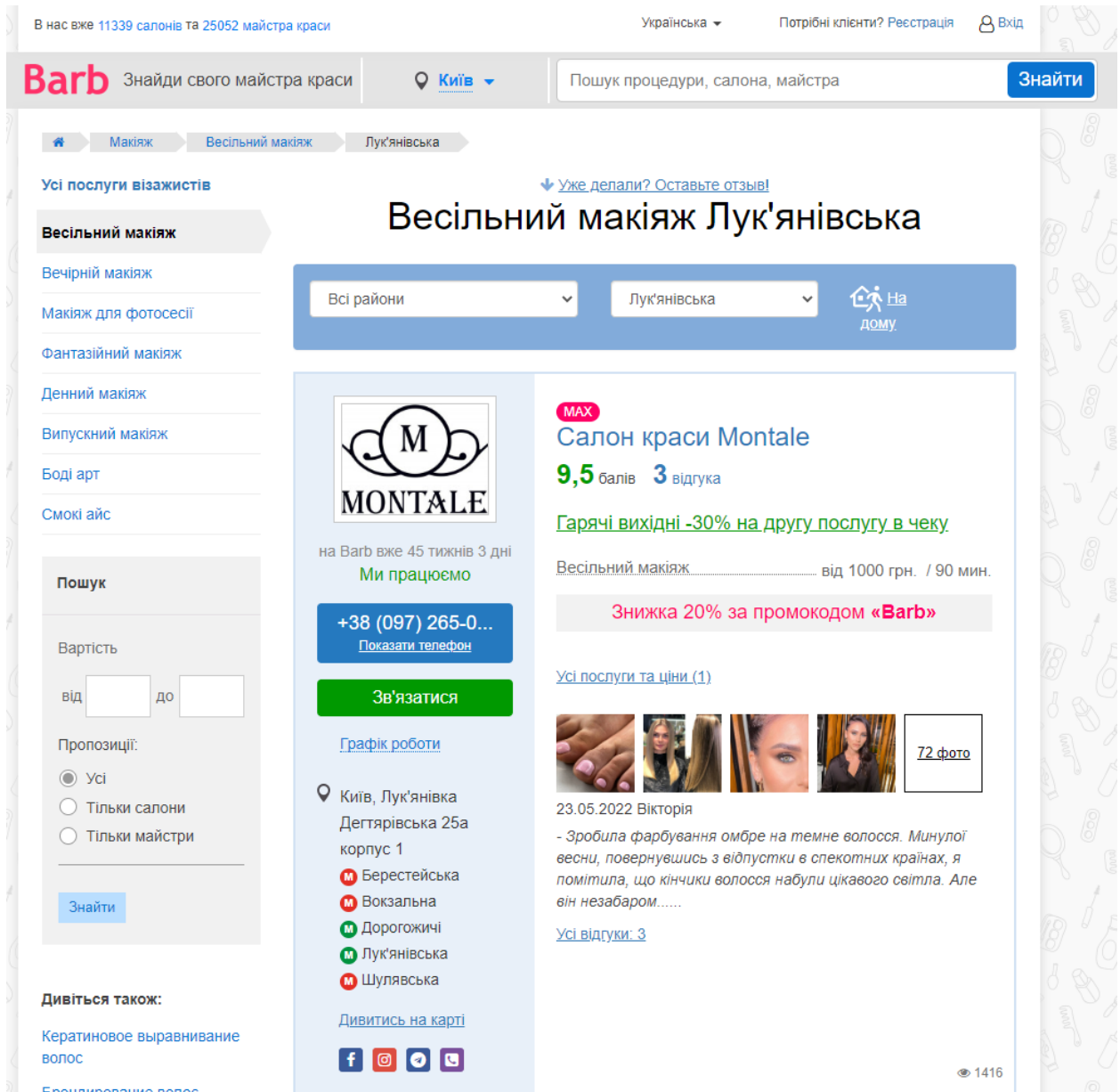


Рисунок 1.1 – Вебсайт Barb

Дана програма дозволяє зв'язатися з майстром, відправляючи повідомлення, але сформувавши замовлення на сайті не можна. Тобто він носить загалом інформаційні функції, а подальше ведення бізнесу реалізується або через інші програми, або через інші засоби комунікації і зустріч в подальшому.

Окрім того існують велика кількість вебсайтів окремих салонів. Такі вебсайти зазвичай містять інформацію про виконувані процедури, про їх вартість, але не мають засобів замовлення цих процедур. Однак в цьому розрізі окремо стоять вебсайти мереж салонів, вони зазвичай мають засоби роботи з більш широкою функціональністю. Для прикладу розглянемо вебсайт мережі салонів краси КАФО (рис. 1.2) [8].

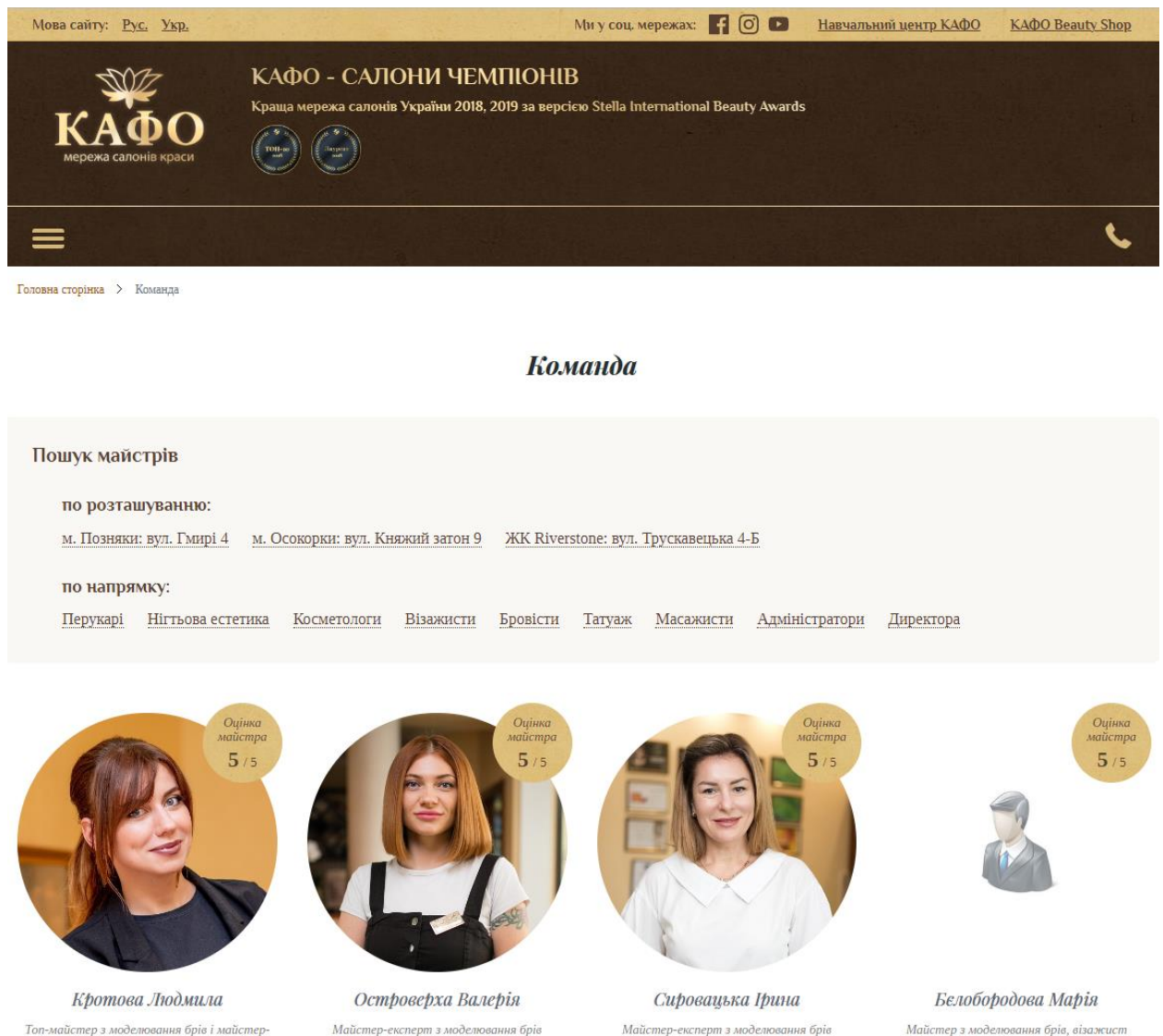


Рисунок 1.2 – Вебсайт мережі салонів краси КАФО

Даний вебсайт дозволяє переглядати дані за різними салонами, містить інформацію про майстрів, в тому числі про рівень їх вмінь, створюючи при цьому певний рейтинг майстрів, надає інформацію про освіту

майстрів, а також містить перелік відгуків клієнтів. Проте по суті створення відгуків – єдина активна функція, яку може виконувати клієнт, клієнт не може створювати замовлення.

Враховуючи те, що основним прототипом було обрано саме програму Barb, то було виконано її порівняння з програмою, яка розробляється в роботі (табл. 1.1).

Таблиця 1.1 – Порівняння аналогів

Критерій	Barb	Дипломна програма
Пошук процедур у сфері краси	+	+
Пошук майстрів	+	+
Пошук салонів	+	–
Перегляд процедури за категоріями	+	+
Відгуки	Про майстрів	Про спеціальності майстрів та про процедури
Замовлення процедур	–	+
Планування роботи майстра	–	+
Визначення засобів виконання процедур	–	+
Відправлення повідомлення майстру	+	+

Розробка повинна відрізнятися тим, що вона надає можливість створення відгуків не просто про майстра загалом, а про його професійні навички за певною професією або про процедури, які ним надаються. Також відмінністю є те, що програма повинна підтримувати можливість оформлення замовлень і пов'язане з цим планування роботи майстра, а також деталізувати процедури у виконанні майстра до засобів, які

використовуються для них, щоб клієнт точніше розумів, на скільки це відповідає його вимогам, якщо такі вимоги є. При цьому програма не надає можливості пошуку салонів краси, а є спеціалізованою саме на роботі з індивідуальними майстрами.

1.3 Висновки за розділом 1

Виконано огляд предметної області, розглянуто аналоги, виділено основний прототип розробки, яким є застосунок Barb, який дозволяє виконувати пошук не тільки салонів краси, але і майстрів. Користуючись логікою роботи цієї програми, було додано певні особливості розробки, що виконується, роблячи основною її ідеєю роботу саме з індивідуальними майстрами.

Поставлено наступні завдання в роботі:

- аналіз предметної області;
- проєктування бази даних;
- проєктування ПЗ бронювання індивідуальних процедур у сфері краси;
- програмна реалізація ПЗ бронювання індивідуальних процедур у сфері краси.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір засобів розробки програми

Під час вибору мови, якою мала розроблятися програма, було виконано порівняння мов програмування Python [9]-[10] і Ruby (табл. 2.1). Підсумком цього порівняння став вибір мови програмування Python для реалізації програми, зважаючи на те, що вона підтримує множинне наслідування, відповідно дозволяє гнучкіше працювати з готовим кодом, має більше бібліотек для повторного використання, окрім того засоби цієї мови простіше вивчати і сприймати, що важливо для ситуації, коли потрібне вивчення додаткових можливостей під час реалізації програми. Загалом можна відзначити, що обидві мови є достатньо якісними рішеннями для реалізації вебзастосунків на поточний момент, проте враховуючи обмеженість ресурсів під час реалізації дипломної роботи, саме Python було обрано.

Таблиця 2.1 – Вибір мови програмування

Критерій	Python	Ruby
Об'єктно-орієнтована мова програмування	Так	Так
Простота вивчення мови	Вища	Менша
Високорівнева мова	Так	Так
Підтримка множинного наслідування	Так	Ні
Вебфреймворк	Django	Ruby on Rails
Готові бібліотеки для повторного використання	Більше	Менше

2.2 Функціональні вимоги до програмного забезпечення

Функціональні вимоги до програми були визначені за результатами аналізу предметної області. Серед користувачів програми виділено клієнта (рис. 2.1) і майстра (людина, що надає процедури в сфері краси).

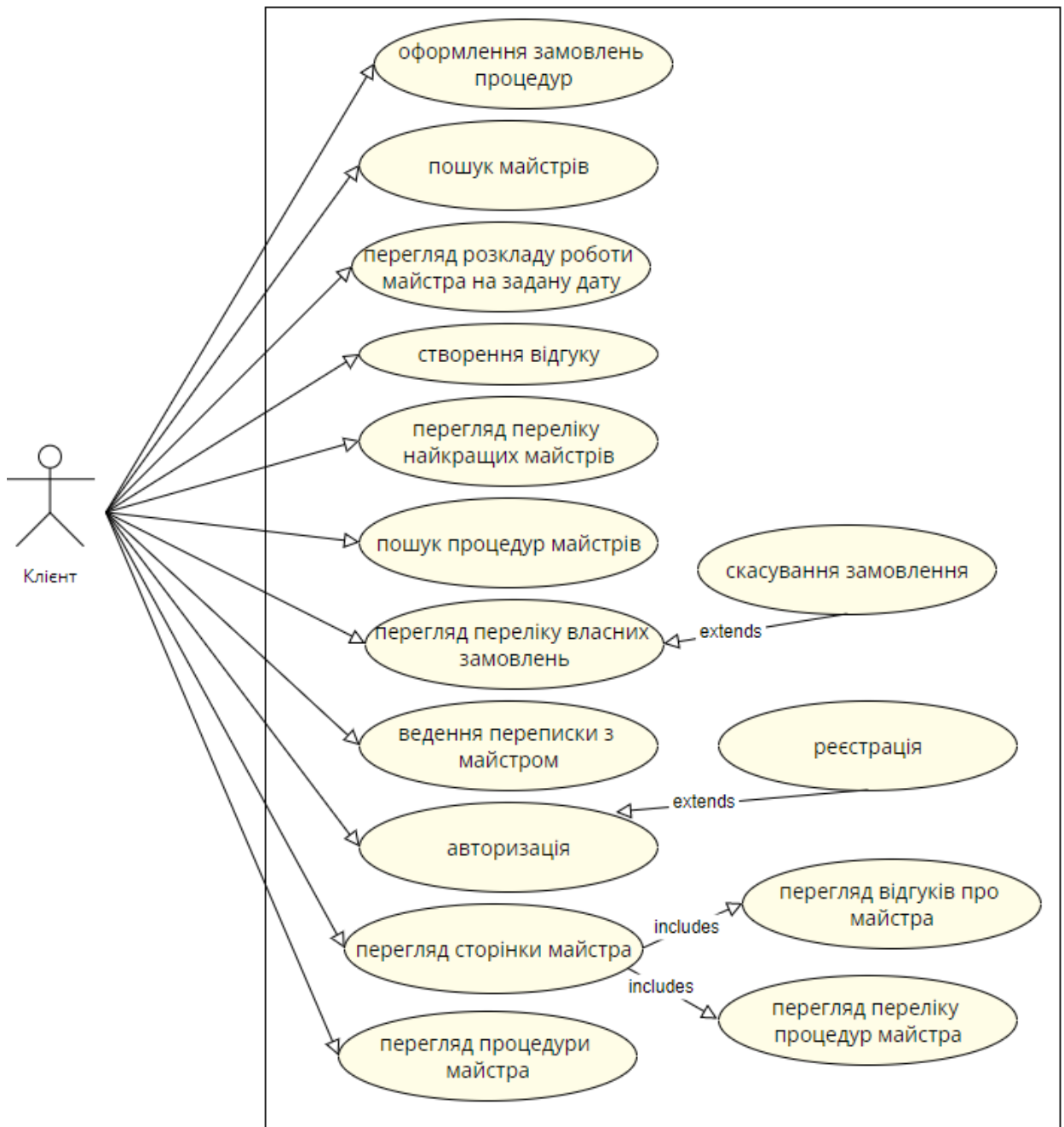


Рисунок 2.1 – Діаграма прецедентів клієнта

Відповідна діаграма прецедентів для майстра представлена на рис. 2.2.

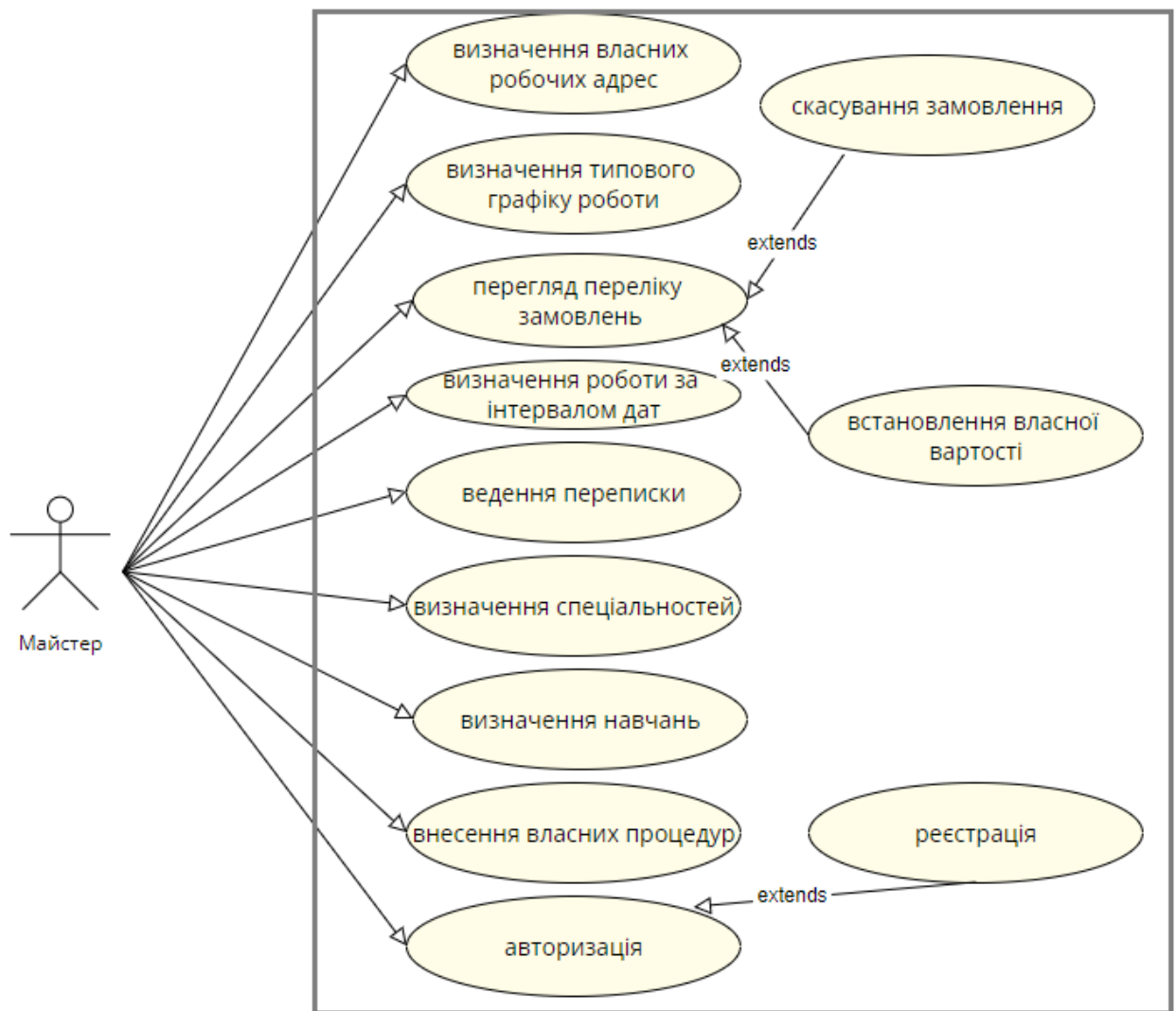


Рисунок 2.2 – Діаграма прецедентів майстра

ПЗ має задовольняти для клієнта і майстра наступні функціональні вимоги:

- оформлення замовлень процедур у сфері краси від майстра;
- пошук майстрів;
- пошук процедур майстрів;
- перегляд сторінки майстра;
- перегляд процедури майстра;
- внесення майстром власних процедур;
- визначення засобів, які майстер використовує для виконання процедури;
- перегляд переліку процедур, які виконує майстер;

- визначення навчань майстра;
- визначення спеціальностей майстра;
- перегляд розкладу роботи майстра на задану дату;
- створення відгуку про спеціальність майстра;
- перегляд відгуків про майстра;
- створення відгуку про процедуру у виконанні майстра;
- перегляд переліку власних замовлень;
- перегляд переліку замовлень на роботу майстра;
- встановлення власної вартості замовлення майстром;
- скасування замовлення;
- підтвердження замовлення;
- визначення типового графіку роботи майстра;
- визначення роботи за інтервалом дат майстром;
- визначення власних робочих адрес майстром;
- ведення переписки між майстром і клієнтом;
- визначення відсутності засобів для процедури;
- реєстрація і авторизація майстрів і клієнтів.

2.3 Проєктування бази даних

Оскільки програма направлена на роботу з індивідуальними майстрами в сфері краси, то однією з основних сутностей предметної області є майстер, якого представлено в базі даних (БД) таблицею Master, поля якої включають:

- id – ідентифікатор майстра;
- username – логін, який використовує цей майстер для роботи в програмі;
- mastername – повне ім'я майстра;
- password – пароль майстра;
- phone – контактний номер або номери телефонів майстра;

- city – місто, в якому мешкає і працює майстер;
- information – інформація про майстра;
- area_log – часова затримка, яку необхідно встановити між замовленнями, якщо вони реалізуються в клієнта вдома і переміщення між замовленнями відбувається в межах одного й того самого району міста;
- outside_area_log – часова затримка, яку необхідно встановити між замовленнями, якщо вони реалізуються в клієнта вдома і переміщення між замовленнями відбувається між різними районами міста;
- untabled – поле, яке визначає, чи працює майстер тільки за графіком, чи можливо сформулювати замовлення на будь-який час (не тільки час робочого графіка) і майстер його розгляне.

Наступною важливою сутністю предметної області є звичайно саме процедура у сфері краси. Оскільки процедур може бути в програмі достатньо багато, а переглядати постійно всі незручно, то процедури було розділено на категорії, а відповідна сутність фактично представлена двома одиницями.

Сутність категорії процедур у сфері краси представлена таблицею TreatmentCategory, поля якої включають:

- id – ідентифікатор категорії процедури у сфері краси;
- category_title – назва категорії процедури у сфері краси.

Сама процедура у сфері краси представлена таблицею Treatment, поля якої включають:

- id – ідентифікатор процедури у сфері краси;
- treatmentcategory – категорія, до якої належить процедура;
- treat_title – назва процедури у сфері краси.

Таблиця Treatment має зв'язок з таблицею TreatmentCategory, створений полем treatmentcategory, типу один до багатьох: кожна процедура у сфері краси належить до однієї конкретної категорії, а одна категорія процедур у сфері краси має різні процедури в сфера краси, які її представляють.

Окрім того потрібно визначити, які саме процедури виконуються конкретним майстром. Для цього створена таблиця MasterTreatment, поля якої включають:

- id – ідентифікатор процедури у сфері краси, яку виконує майстер;
- master – майстер;
- treatment – процедура у сфері краси, яку цей майстер виконує;
- specification – особливості процедури у сфері краси у виконанні цього майстра;
- score – середня оцінка, яку було виставлено даній процедурі майстра.

Кожна процедура може мати певні особливості виконання або певну особливу загалом реалізацію у виконанні майстра. Для цього створена таблиця MasterTreatmentSpecification, поля якої включають:

- id – ідентифікатор особливої реалізації процедури у виконанні майстра;
- mastertreatment – процедура у виконанні майстра, до якої належить ця особлива реалізація (або базова реалізація, але такий варіант один для кожної процедури);
- specification – визначення того, в чому полягає особливість;
- price – вартість виконання цієї реалізації процедури у даного майстра;
- duration – тривалість виконання цієї реалізації процедури у даного майстра.

Звичайно, що сама програма створюється заради взаємодії майстрів з клієнтами і навпаки, а основною цією взаємодією є замовлення процедур у сфері краси. Для цього створена таблиця TreatmentOrder, поля якої включають:

- id – ідентифікатор замовлення;
- client – клієнт, який виконав замовлення;
- order_date – дата створення замовлення;
- treatment_time – час початку виконання процедур;

- treatment_end – час завершення виконання процедур;
- price – сумарна вартість замовлення;
- state – стан замовлення;
- client_address – адреса клієнта, де має виконуватися замовлення (якщо клієнт бажає виконувати процедури в себе вдома, а якщо реалізація виконується у приміщенні, наданому майстром, то дане поле залишається порожнім);
- cityarea – район міста, в якому знаходиться клієнт, якщо замовлення оформлено за адресою клієнта;
- workingaddress – адреса роботи майстра для випадків, коли клієнт готовий сам приїхати до майстра.

Таблиця TreatmentOrder має зв'язок з таблицею Client, створений полем client, типу один до багатьох: кожен клієнт може створити багато замовлень в програмі, а одне замовлення повинно бути створено тільки одним конкретним клієнтом.

Таблиця TreatmentOrder має зв'язок з таблицею CityArea, створений полем cityarea, типу один до багатьох: кожен район міста може використовуватися в різних замовленнях, а одне замовлення може стосуватися тільки одного або жодного району міста (перший варіант, якщо замовлення виконується у клієнта вдома, а другий – якщо процедури реалізуються за адресою майстра).

Таблиця TreatmentOrder має зв'язок з таблицею WorkingAddress, створений полем workingaddress, типу один до багатьох: кожна процедура у сфері краси належить до однієї конкретної категорії, а одна категорія процедур у сфері краси має різні процедури в сфера краси, які її представляють.

Кожне замовлення може складатися з однієї або декількох процедур, тому потрібно певним чином зберігати склад замовлення. Для цього створена таблиця TreatmentOrderItem, поля якої включають:

- id – ідентифікатор одиниці замовлення;

- treatmentorder – замовлення, до якого відноситься процедура;
- mastertreatmentSpecification – конкретна процедура у виконанні майстра з певною особливістю, яку було замовлено.

Таблиця TreatmentOrderItem має зв'язок з таблицею TreatmentOrder, створений полем treatmentorder, типу один до багатьох. Аналогічного типу зв'язок є між таблицями TreatmentOrderItem і MasterTreatmentSpecification. Але для пояснення типів зв'язків слід визначити, що таблиця TreatmentOrderItem визначає зв'язок багато до багатьох між таблицями TreatmentOrder і MasterTreatmentSpecification, оскільки кожне замовлення може складатися з однієї або більше процедур у реалізації з певними особливостями у виконанні майстра, а кожна така реалізація процедури може бути задіяна в різних замовленнях.

Іншою стороною взаємодії в програмі є клієнт. Для збереження цих даних створено таблицю Client, поля якої включають:

- id – ідентифікатор клієнта;
- username – логін, який використовує клієнт для роботи в програмі;
- clientname – повне ім'я клієнта;
- password – пароль, який використовує клієнт для роботи в програмі;
- phone – номер або номери телефонів клієнта;
- cityarea – район міста, в якому проживає клієнт в основній визначеній адресі;
- base_address – основна визначена адреса клієнта.

Оскільки програма передбачає створення клієнтами відгуків, то було виділено дві окремі сутності для збереження відгуків клієнтів.

Першою такою сутністю є сутність, що відповідає відгуку про процедуру у виконанні майстра, для збереження якої використовується таблиця MasterTreatmentReview, поля якої включають:

- id – ідентифікатор відгуку про процедуру у виконанні майстра;
- mastertreatment – процедура у виконанні майстра, якої стосується відгук;

- client – клієнт, який створив відгук;
- score – оцінка, виставлена процедурі у відгуку клієнтом;
- review_text – текст відгуку;
- review_date – дата і час створення відгуку.

Таблиця MasterTreatmentReview має зв'язок з таблицею MasterTreatment, створений полем mastertreatment, типу один до багатьох, оскільки одна процедура може мати багато відгуків про себе, а кожен відгук стосується певної процедури у виконанні майстра.

Таблиця MasterTreatmentReview має зв'язок з таблицею Client, створений полем client, типу один до багатьох, оскільки один клієнт може створити багато відгуків у програмі, а кожен відгук має в якості автора тільки одного клієнта.

Другою сутністю, що стосується відгуків, є сутність, що відповідає відгуку про професію майстра, для збереження якої використовується таблиця MasterProfessionReview, поля якої включають:

- id – ідентифікатор відгуку про професію майстра;
- masterprofession – професія певного майстра, якої стосується відгук;
- client – клієнт, який створив відгук;
- score – оцінка, виставлена діяльності майстра за цією професією у відгуку клієнтом;
- review_text – текст відгуку;
- review_date – дата і час створення відгуку.

Графік роботи майстра протягом тижня в різні дні, якщо для нього такий застосовується, представлений в таблиці Workweek, поля якої включають:

- id – ідентифікатор запису про графік роботи майстра в певний день тижня;
- master – майстер, якого стосується графік;
- day – день тижня, в який реалізується такий графік зазвичай;
- start_day – час початку роботи в цей день;

- end_day – час завершення роботи в цей день;
- break_start – час, коли починається перерва;
- break_end – час, коли закінчується перерва.

Таблиця Workweek має зв'язок з таблицею Master, створений полем master, типу один до багатьох, оскільки один майстер може мати графік роботи на декілька днів на тиждень, а ось кожен запис про роботу в певний день тижня стосується одного конкретного майстра, для якого саме такий графік встановлюється.

Визначення графіку роботи майстра на певну дату реалізується окремою сутністю. Тільки у випадку, якщо запис про роботу на певну дату є, тільки тоді вважається, що майстер в цей день працює (за графіком). Якщо майстер визначив, що він готовий працювати поза графіком, то звичайно, що при пошуку необхідності існування запису про роботу в певну дату не буде. Такі дані представлені в таблиці Workday, поля якої включають:

- id – ідентифікатор запису про роботу в певну дату;
- master – майстер, який працює в цю дату;
- start – час початку роботи в цей день;
- end – час завершення роботи.

Якщо є перерва, то реалізується декілька записів на певну дату. Інформація про роботу в певні дні тижня зберігається по суті для існування шаблонного графіку, який за замовчуванням, якщо це підтверджує майстер, переноситься на роботу в задану дату.

Таблиця Workday має зв'язок з таблицею Master, створений полем master, типу один до багатьох, оскільки один майстер працює в різні дати і відповідно має багато записів про таку роботу, а кожен запис стосується тільки одного конкретного майстра.

Робота програми передбачається в межах країни, для якої вона запускається, проте для майстра, який працює індивідуально, важливою характеристикою стає те, що він може їздити до клієнтів. У такому випадку йому необхідний час для переміщення між замовниками. Відповідно важливо

розуміти, де саме реалізується певна робота. Для цього було виділено сутності міста та району в цьому місті.

Сутність міста, де виконуються послуги в сфері краси, представлена таблицею City, поля якої включають:

- id – ідентифікатор міста;
- name – назва міста.

Сутність району міста представлена таблицею CityArea, поля якої включають:

- id – ідентифікатор району міста;
- city – місто, де знаходиться цей район;
- name – назва району міста.

Таблиця CityArea має зв'язок з таблицею city, створений полем city, типу один до багатьох, оскільки одне місто може містити декілька районів, а один район знаходиться в конкретному місті.

Майстер може надавати процедури у сфері краси в різних місцях, тобто у нього може бути декілька робочих кабінетів у випадках, коли замовлення реалізуються не в приміщенні клієнта. Для збереження цих даних створено таблицю WorkingAddress, поля якої включають:

- id – ідентифікатор робочої адреси майстра;
- mastercityarea – район міста, в якому знаходиться ця адреса;
- address – повністю адреса.

Таблиця WorkingAddress має зв'язок з таблицею MasterCityArea, створений полем mastercityarea, типу один до багатьох, оскільки один район міста може мати багато робочих адрес одного або різних майстрів, а одна робоча адреса стосується тільки одного району, де вона розташована адміністративно.

Майстер може працювати не тільки за різними адресами, але і в різних районах міста. Для того, щоб розуміти, в якому саме районі працює майстер, а вже далі деталізувати те, за якою адресою він працює, введено додатковий прошарок – таблицю MasterCityArea, поля якої включають:

- id – ідентифікатор району міста, в якому працює майстер;
- master – майстер;
- cityarea – район міста, в якому працює майстер.

Таблиця MasterCityArea має зв'язок з таблицею Master, створений полем master, типу один до багатьох, оскільки один район міста, в якому працює майстер, стосується одного конкретного майстра, а один майстер може працювати в різних районах.

Таблиця MasterCityArea має зв'язок з таблицею CityArea, створений полем cityarea, типу один до багатьох, оскільки один район міста може використовуватися для роботи різними майстрами, але один запис про роботу певного майстра в певному районі міста стосується одного району.

Для переписки між майстрами і клієнтами використовується сутність повідомлення, для збереження даних якого створено таблицю Message, поля якої включають:

- id – ідентифікатор повідомлення;
- master – майстер, до якого написано повідомлення, або який створив повідомлення;
- client – клієнт, який написав повідомлення або якому адресовано повідомлення;
- master_to_client – напрямок повідомлення (від клієнта майстру або від майстра клієнту);
- message_text – текст повідомлення;
- message_date – дата і час створення повідомлення.

Таблиця Message має зв'язок з таблицею Client, створений полем client, типу один до багатьох, оскільки один клієнт може мати багато повідомлень, але одне повідомлення стосується одного конкретного клієнта.

Таблиця Message має зв'язок з таблицею Master, створений полем master, типу один до багатьох, оскільки один майстер може мати багато повідомлень, але одне повідомлення стосується одного конкретного майстра.

Для збереження спеціальності майстра створено таблицю Profession, поля якої включають:

- id – ідентифікатор спеціальності;
- title – назва спеціальності.

Оскільки майстер може мати декілька спеціальностей, а будь-яка спеціальність може характеризувати велику кількість майстрів як спеціалістів, то це фактично зв'язок багато до багатьох між таблицями Master і Profession. Його реалізовано через створення проміжної таблиці MasterProfession, поля якої включають:

- id – ідентифікатор спеціальності майстра;
- master – майстер, спеціальність якого визначено;
- profession – спеціальність майстра;
- information – інформація про володіння спеціальністю майстром;
- score – середня оцінка, виставлена вмінням майстра за цією спеціальністю.

Майстер окрім спеціальності характеризується ще освітою. Для збереження освіти майстра створено таблицю Education, поля якої включають:

- id – ідентифікатор навчання майстра;
- masterprofession – професія майстра, за якою відбувалось це навчання;
- information – інформація про навчання, повний опис;
- begin – час початку навчання;
- end – час завершення навчання.

Таблиця Education має зв'язок з таблицею MasterProfession, створений полем masterprofession, типу один до багатьох, оскільки одна спеціальність майстра може підтримуватися різними навчаннями, але кожне навчання реалізується за чітко визначеною спеціальністю.

Особливістю програми також є те, що процедури у майстра можуть характеризуватися засобами, які він використовує для роботи (наприклад,

фарба для фарбування волосся). Для збереження цих даних забезпечено дві сутності. Перша з них представляє категорію засоби, для цього створено таблицю ProductCategory, поля якої включають:

- id – ідентифікатор категорії засобу;
- category_title – назва категорії засобу.

Для збереження самого засобу створено таблицю Product, поля якої включають:

- id – ідентифікатор засобу;
- productcategory – категорія засобу;
- title – назва засобу;
- producer – виробник засобу;
- country – країни виробництва, якщо її необхідно зазначити.

Таблиця Product має зв'язок з таблицею ProductCategory, створений полем productcategory, типу один до багатьох, оскільки одна категорія засобів може мати багато засобів, а один засіб стосується однієї категорії.

Збереження даних про використання цих засобів у процедурах майстра реалізується через таблицю MasterTreatmentProduct, поля якої включають:

- id – ідентифікатор використання засобу в процедурі;
- mastertreatmentspecification – особлива процедура у виконанні майстра, в якій застосовується засіб;
- product – засіб;
- availability – доступність засобу на даний момент.

Таблиця MasterTreatmentProduct реалізує зв'язок багато до багатьох між таблицями MasterTreatmentSpecification і Product: одна особлива процедура майстра може містити різні засоби, а один засіб може використовуватися в різних процедурах.

В результаті була визначена логічна структура БД (рис. 2.3).

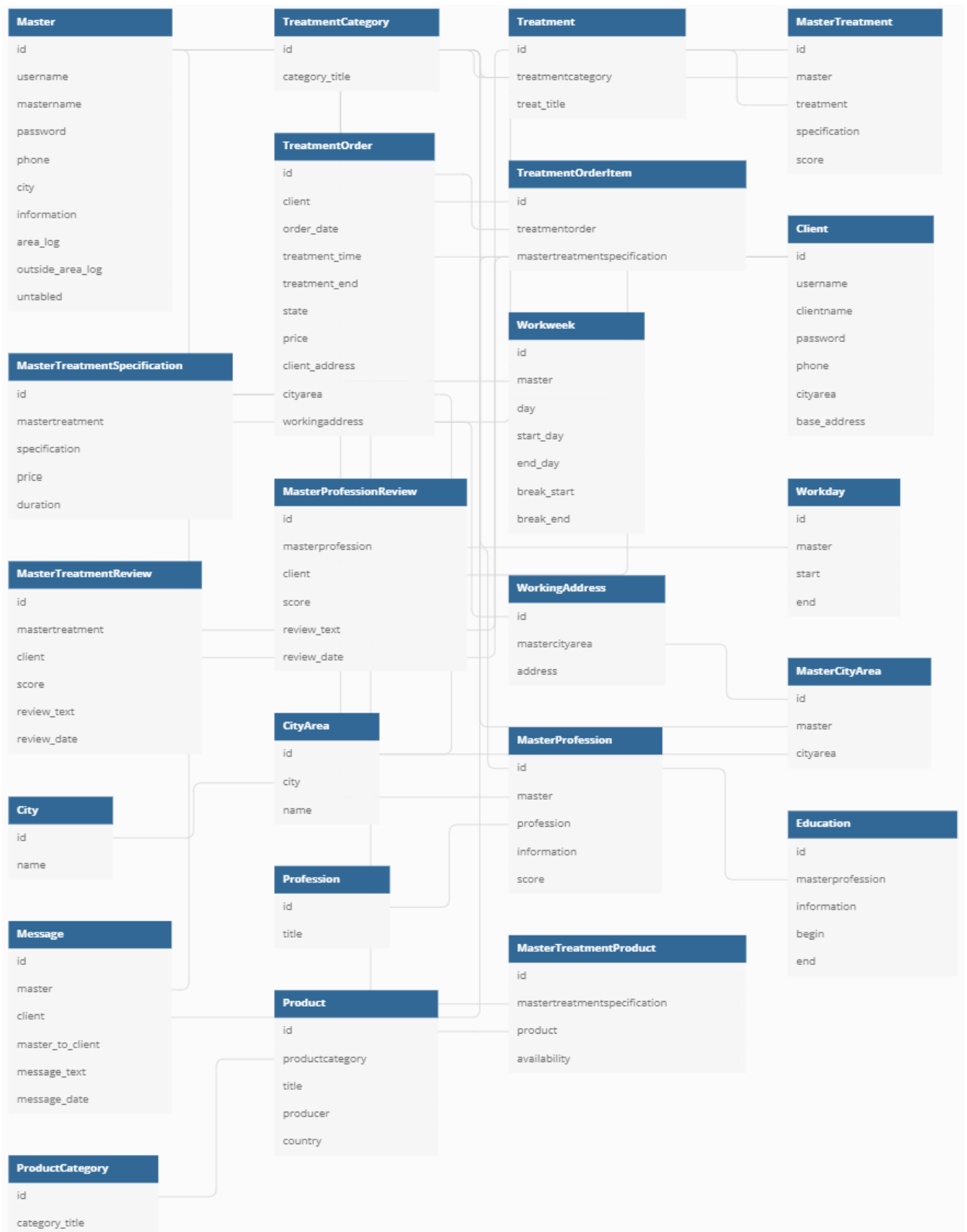


Рисунок 2.3 – Схема структури БД

В якості системи керування БД використовувалась MySQL, що на практиці продемонструвала себе надійною системою, окрім того є повністю безкоштовною.

2.4 Висновки за розділом 2

Виконано порівняння мов для реалізації ПЗ, Ruby та Python, в підсумку обрано Python для реалізації. Виписано функціональні вимоги до програми і створено діаграми прецедентів для клієнта та майстра. Забезпечено проєктування БД.

3 РОЗРОБКА ПРОГРАМИ

3.1 Порядок роботи користувача з програмою

Функціональна схема роботи програми (рис. 3.1) дозволила визначити базовий порядок роботи з програмою користувачів.

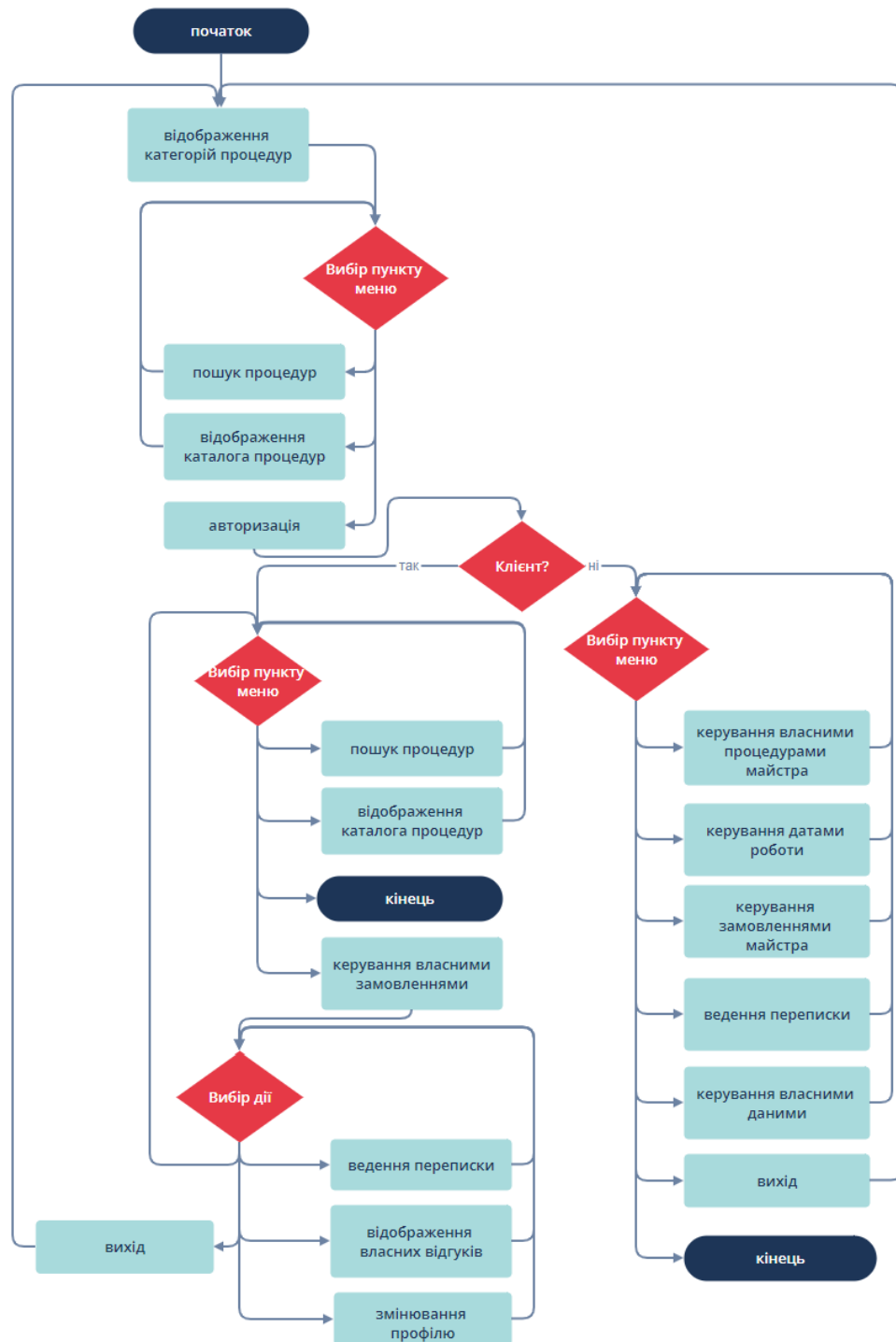


Рисунок 3.1 – Функціональна схема роботи програми

3.2 Опис реалізації програми

Клас `TreatmentCreateView` виконує оброблення даних стосовно додавання процедури у сфері краси в програмі, заповнюючи з форми назву процедури і на основі переданого з форми номеру обраної категорії заповнюючи повні дані категорії процедур, а також заповнюючи для роботи відповідного переліку набір всіх існуючих в програмі категорій процедур у сфері краси.

Клас `TreatmentUpdateView` виконує оброблення даних стосовно редагування процедури у сфері краси в програмі, заповнюючи з форми назву процедури, якщо їх було змінено, і на основі переданого з форми номеру обраної категорії, якщо такий номер було змінено, заповнюючи повні дані категорії процедур, а також заповнюючи для роботи відповідного переліку набір всіх існуючих в програмі категорій процедур у сфері краси.

Клас `CategoryTreatmentsView` забезпечує наповнення переліку прцоедур у сфері краси за заданою категорією цих процедур. Номер категорії передається через параметр запиту, а сформований перелік процедур цієї категорії впорядковується за номером категорії.

Клас `TreatmentsView` забезпечує наповнення переліку всіх наявних процедур у сфері краси в програмі.

Клас `YourTreatmentsView` забезпечує наповнення переліку процедур у сфері краси, які надаються майстром, який виконав авторизацію в програмі і відкрив сторінку з переліком всіх власних процедур. Усі процедури в результаті впорядковуються за назвою процедур.

Клас `MasterTreatmentSpecsView` забезпечує наповнення переліку особливостей виконання процедур, які надаються майстром, який виконав авторизацію в програмі і відкрив сторінку з переліком всіх власних особливостей всіх виконуваних процедур.

Клас `ProductCreateView` виконує оброблення даних стосовно додавання засобу для реалізації процедур у сфері краси в програму,

заповнюючи з форми назву засобу, виробника та країну виробництва, визначаючи за переданим номером повні дані категорії засобу, а також заповнюючи для роботи відповідного переліку набір всіх існуючих в програмі категорій засобів.

Клас `ProductUpdateView` виконує оброблення даних стосовно редагування процедури у сфері краси в програмі, заповнюючи з форми назву засобу, виробника та країну виробництва, якщо їх було змінено, визначаючи за переданим номером повні дані категорії засобу, якщо вона була змінена, а також заповнюючи для роботи відповідного переліку набір всіх існуючих в програмі категорій засобів.

Клас `MasterTreatmentsView` забезпечує наповнення переліку процедур, які надаються майстром, номер якого було передано через параметр запиту, яким було сформовано сторінку.

Клас `CategoryProductsView` забезпечує наповнення переліку засобів, які відносять до категорії засобів, номер якого було передано через параметр запиту, яким було сформовано сторінку. Засоби впорядковуються за назвою. Перелік категорій засобів заповнюється автоматично.

Клас `ProductCategoryCreateView` виконує оброблення даних стосовно додавання категорії засобів для реалізації процедур у сфері краси в програму, заповнюючи з форми назву категорії.

Клас `ProductCategoryUpdateView` виконує оброблення даних стосовно редагування категорії засобів для реалізації процедур у сфері краси в програмі, заповнюючи з форми назву категорії, якщо вона була змінена користувачем.

Клас `MasterTreatmentCreateView` виконує оброблення даних стосовно додавання нової процедури майстра, заповнюючи з форми специфікацію цієї процедури, визначаючи за переданим номером повні дані процедури з каталогу, а також заповнюючи для роботи відповідного переліку набір всіх існуючих в програмі категорій процедур та самих процедур.

Клас `MasterTreatmentUpdateView` виконує оброблення даних стосовно редагування існуючої процедури майстра, заповнюючи з форми специфікацію цієї процедури, якщо вона була змінена, визначаючи за переданим номером повні дані процедури з каталогу, якщо вона має бути змінена, а також заповнюючи для роботи відповідного переліку набір всіх існуючих в програмі категорій процедур та самих процедур.

Клас `MasterTreatmentSpecificationCreateView` виконує оброблення даних стосовно додавання нової особливості деякої процедури у виконанні майстра, заповнюючи з форми специфікацію цієї процедури, вартість цієї особливості, тривалість її виконання, визначаючи за переданим номером повні дані процедури майстра, до якої належить ця особливість, а також заповнюючи для роботи відповідного переліку набір всіх процедур, які надаються майстром, який виконав авторизацію в програмі. Також через параметр `products` передається перелік засобів, за допомогою яких виконується ця реалізація процедури, а далі цей перелік розбирається, виділяються об'єкти цих засобів, а тоді записи про засоби при виконанні процедури майстром вносяться в БД.

Клас `MasterTreatmentSpecificationUpdateView` виконує оброблення даних стосовно редагування існуючої особливості деякої процедури у виконанні майстра, заповнюючи з форми значення наступних параметрів, якщо вони були змінені: специфікацію цієї процедури, вартість цієї особливості, тривалість її виконання. Також виконується підтримка роботи форми з визначенням засобів процедури, переданих через параметр `products`, через який передається перелік засобів, за допомогою яких виконується ця реалізація процедури. Цей перелік розбирається, виділяються об'єкти цих засобів, визначається, які засоби ще не були внесені в БД, тоді вони вносяться в БД для цієї процедури, а якщо якийсь засіб навпаки випав з переліку, то він вилучається з БД стосовно цієї процедури.

Клас `MasterTreatmentDetailView` виконує деталізацію реалізації процедури у виконанні майстра. Окрім даних самої процедури автоматично

заповнюються також дані про особливості виконання цієї процедури, що надаються майстром, відгуки про цю процедуру:

```
mtreatments =
MasterTreatmentSpecification.objects.filter(mastertreatment =
self.object.pk)
    mtreatments = mtreatments.order_by('specification')
    treatmentdataformaster ['specifications'] = mtreatments
    reviews = MasterTreatmentReview.objects.filter(mastertreatment =
self.object.pk)
    reviews = reviews.order_by('-review_date')
    treatmentdataformaster ['reviews'] = reviews
```

Клас `MasterTreatmentSpecificationDetailView` виконує деталізацію особливості реалізації процедури у виконанні майстра. Окрім даних самої особливості процедури автоматично заповнюються також дані про замовлення цієї саме реалізації процедури, що надається майстром, та дані про засоби, які використовуються для цієї процедури.

Клас `TreatmentOrderCreateView` виконує оброблення даних стосовно створення нового замовлення, заповнюючи з форми час початку виконання процедур, який обирається з доступних слотів, що відповідно визначаються за вказаною клієнтом датою на основі класу `TimeslotsListView`. Окрім того через форму заповнюється адреса клієнта, якщо він обрав обслуговування на дому. Через параметри запиту встановлюються повні дані району міста, де має бути виконано обслуговування, на основі відповідного переданого номеру району або робочої адреси майстра на основі відповідного номера, в залежності від того, який саме з цих варіантів був обраний клієнтом. Також відбувається програмне внесення даних про процедури, які були замовленні в цьому замовленні. Значення витягаються через параметр `treatments`. Це перелік, оформлений у вигляді послідовності номерів особливостей реалізації процедур майстром. Ці номери розбираються, виділяються відповідні об'єкти, а тоді відповідні записи про склад замовлення вносяться в БД. Під

час цього розбору також визначається сумарна вартість і тривалість виконання цих процедур, а тоді сумарне значення вноситься в параметри замовлення:

```

for tspec in treatmentobjects:
    TreatmentOrderItem.objects.create(treatmentorder =
self.object.pk, mastertreatmentsspecification = tspec)
    dur = dur + tspec.duration
    price = price + rspec.price
self.price = price
self.treatment_end = self.treatment_time + dur

```

Якщо часу для виконання цієї процедури недостатньо, то такий слот не виводиться під час вибору часу при створенні замовлення. Перевірка виконується на стороні клієнта за переданими даними про наявний вільний час у майстра. Створення замовлення поетапне: спочатку визначається набір процедур і дата, тоді визначається один з доступних часових проміжків.

При збереженні об'єкта, в моделі, також перевіряється, чи не перекривається цей об'єкт з іншими об'єктами замовлень за часом виконання, і якщо це так, то об'єкт в БД не зберігається.

Клас MasterTreatmentOrdersView забезпечує наповнення переліку замовлень процедур у сфері краси, при цьому через параметр передається необхідний стан запитів, і якщо цей стан задано, то виводяться тільки замовлення, адресовані даному майстру, що знаходяться в заданому стані, і впорядковані за часом замовлення, починаючи з останнього створеного. Якщо стан не задано, то виводяться всі замовлення майстра. Це підтримка роботи над керуванням замовленнями.

Клас TreatmentOrderUpdateView виконує оброблення даних стосовно редагування існуючого замовлення, заповнюючи при необхідності з форми час початку виконання процедур, який обирається з доступних слотів, що відповідно визначаються за вказаною клієнтом датою на основі класу TimeslotsListView. Окрім того через форму оновлюється адреса клієнта, якщо

він обрав обслуговування на дому. Через параметри запиту оновлюються повні дані району міста, де має бути виконано обслуговування, на основі відповідного переданого номеру району або робочої адреси майстра на основі відповідного номера, в залежності від того, який саме з цих варіантів був обраний клієнтом. Також відбувається програмне внесення даних про процедури, які були замовлені в цьому замовленні. Значення витягаються через параметр `treatments`. Це перелік, оформлений у вигляді послідовності номерів особливостей реалізації процедур майстром. Ці номери розбираються, виділяються відповідні об'єкти, а тоді перевіряється, чи такі записи вже були внесені в БД. Якщо ні, то вони додаються. Якщо були внесені в замовлення раніше, а тепер зникли, то навпаки вилучаються.

За описаним способом створено і інші класи для підтримки роботи інших функцій, які було описано на діаграмах прецедентів.

Клас `MasterTreatmentReviewsListView` забезпечує наповнення переліку відгуків про процедуру в сфері краси, що надається майстром. Номер процедури передається через параметр запиту до вебсторінки `mtreatment`, а відгуки впорядковуються за часом створення, спочатку видаючи останні створені.

Клас `YourMasterTreatmentReviewsListView` забезпечує наповнення переліку відгуків про процедури в сфері краси у виконанні різних майстрів, що були створені авторизованим у програмі клієнтом.

Клас `SearchTreatmentView` забезпечує наповнення переліку з результатами пошуку процедур у сфері краси, що надаються різними майстрами. Пошук реалізується через фільтри, значення яких задаються через параметри, що включають параметр номеру категорії процедур `category`, параметр назви процедури `treatment`, параметр мінімальної оцінки процедури `score`, параметр району міста, де має працювати майстер, `area`. У відповідності з заданим параметрами відбираються всі процедури у виконанні різних майстрів, які відповідають цим фільтрам. Створений в результаті перелік повертається по завершенні роботи.

Клас `YourTreatmentOrdersView` забезпечує наповнення переліку власних замовлень, створених клієнтом, який увійшов у програму.

Клас `TreatmentOrderCancelView` реалізує скасування замовлення шляхом зміни стану замовлення на скасоване.

3.3 Висновки за розділом 3

Розглянуто те, яким чином фактично програма підтримує логіку взаємодії з користувачем за допомогою функціональної схеми програми. Розглянуто те, за допомогою яких класів та яким чином реалізовано програму.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програма призначена для пошуку клієнтами майстрів, які здатні надати процедури у сфері краси у власних приміщеннях або індивідуально на умовах клієнта, оформлювати замовлення, а майстрам організовувати власний бізнес без участі великих гравців, керуючи розкладом власної роботи за допомогою керування замовленнями.

4.2 Умови виконання програми

Технічні засоби в мінімальній конфігурації, достатній для роботи програми, охоплюють 1 Гб пам'яті жорсткого диску, 4 Гб оперативної пам'яті, 4 ядра процесору (частота 2,8 ГГц).

4.3 Тестування і опис принципів експлуатації програми

Роботу програми перевірено шляхом її тестування за сценаріями використання програми користувачами, які було описано в розділі 2. Після ітераційного усунення виявлених проблем за результатами останнього тестування визначено успішне проходження тестування за всіма сценаріями.

Для доступу з клієнтської сторони до програми потрібно звернутися до неї за адресою вебсервера, після чого буде відкрито інтерфейс клієнта. Інтерфейс клієнта без авторизації містить всі стандартні розділи окрім розділу «Власне», тобто не дозволяє переглянути і вносити ніякі власні дані. Але за допомогою пункту меню «Увійти» можна перейти до авторизації в програмі, звідки доступні кнопка «Майстер», вона дозволяє виконати авторизацію майстра в програмі, після чого відкриється відповідний інтерфейс його роботи.

Інтерфейс клієнта містить 3 розділи:

– розділ «Процедури» дозволяє переглядати всі категорії процедур у сфері краси, а за обраною категорією переходити до відображення всіх процедур цієї категорії, після чого можна відкривати відповідну процедуру, а тоді переходити до пошуку за вказаною процедурою;

– розділ «Пошук» дозволяє виконати пошук майстрів або пошук процедур у сфері краси: однойменні кнопки дозволяють перемикатися між режимами, під час пошуку процедур потрібно вказати категорію процедури, назву процедури, ввести мінімальний середній бал, який допускається, обрати район міста, де має виконуватися пошук, а тоді відобразиться перелік процедур з результатами пошуку, відсортованими за середнім балом (рис. 4.1), у свою чергу під час пошуку майстра вказується район і середній бал мінімальний, який обчислюється при формуванні результатів як усереднений бал всіх професій майстра;

– розділ «Власне» дозволяє керувати даними авторизованого клієнта.

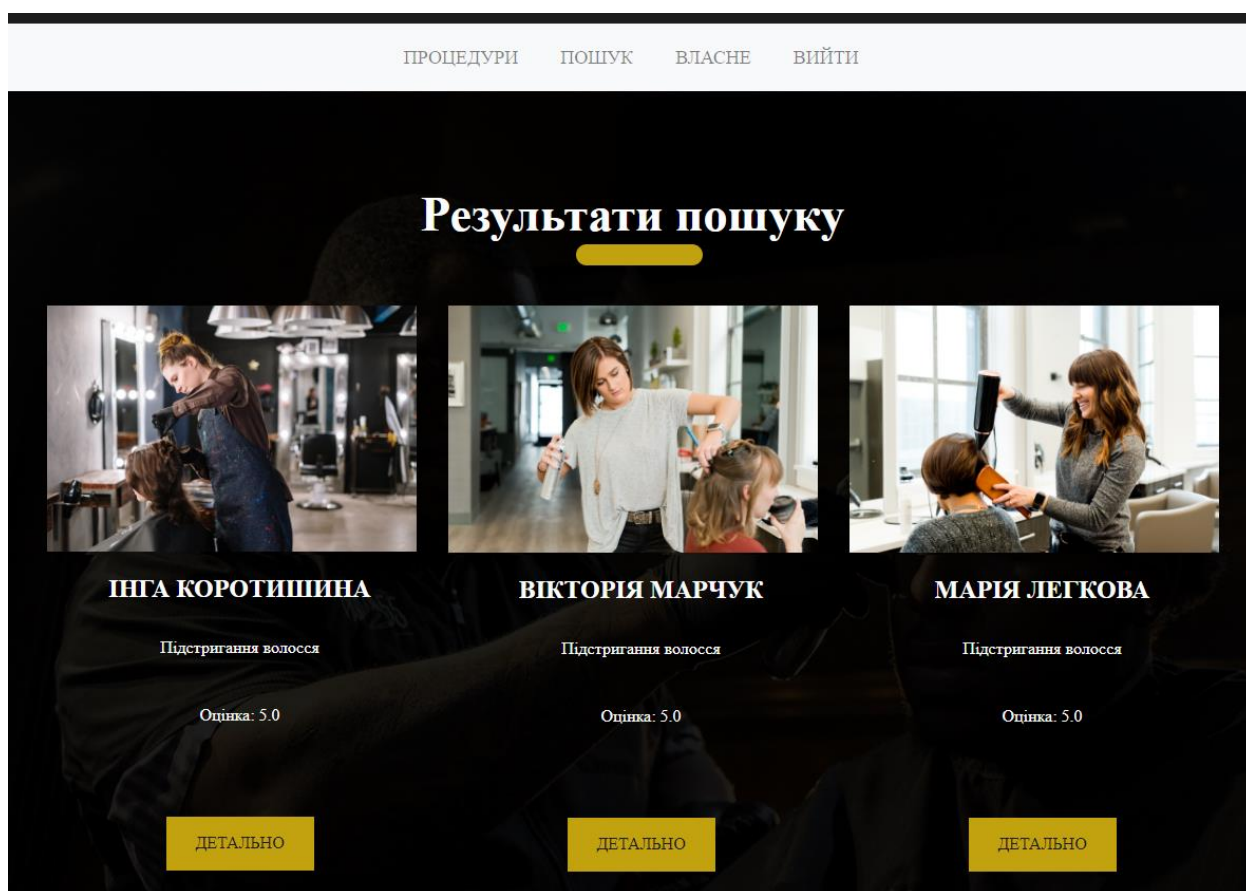


Рисунок 4.1 – Пошук процедур

Розділ «Власне» дозволяє отримати доступ до переписки з майстрами, звідки можна обрати потрібно майстра і переглянути перелік повідомлень, а також відправити нове повідомлення, отримати доступ до відгуків клієнта, який увійшов в програму, до власного профілю. Також цей розділ дозволяє перейти до замовлень авторизованого клієнта. Усі ці переходи доступні за допомогою відповідних кнопок на сторінці, що відкривається при виборі цього розділу.

У випадку вибору власних замовлень відображається перелік замовлень клієнта, за якими можна або переглянути їх склад, або скасувати їх. Оформити ж замовлення можна зі сторінки особливостей виконання процедури з краси, тобто загалом вибудована ієрархія категорія процедури – процедура, що дозволяє переміщатися між процедурами загалом, а після цього процедура майстра – особливість. Саме зі сторінки особливості відбувається додання її до замовлення. На кожній такій сторінці доступна кнопка «Додати до замовлення» та «Оформити замовлення», якщо якась процедура вже була додана до замовлення.

Доступ з результатів пошуку процедур відбувається на сторінки процедур майстра. Доступ з результатів пошуку майстрів відбувається на сторінки майстрів, де перелічені всі процедури, які надаються майстром, надається можливість перейти до перегляду доступних місць на задану дату, виводиться перелік професій майстра з середніми оцінками за кожною, перелік освіт майстра, перелік місць роботи майстра.

Зі сторінки конкретної процедури майстра надається інформація про перелік особливостей процедур (звідси також доступні одразу кнопки «Замовити»), про вартість кожної особливої реалізації цієї процедури, перелік відгуків клієнтів (рис. 4.2). Перелік відгуків відображається з можливістю переміщення між відгуками стрілками. За кожним показується автор, текст відгуку та оцінка.

Відгуки клієнтів

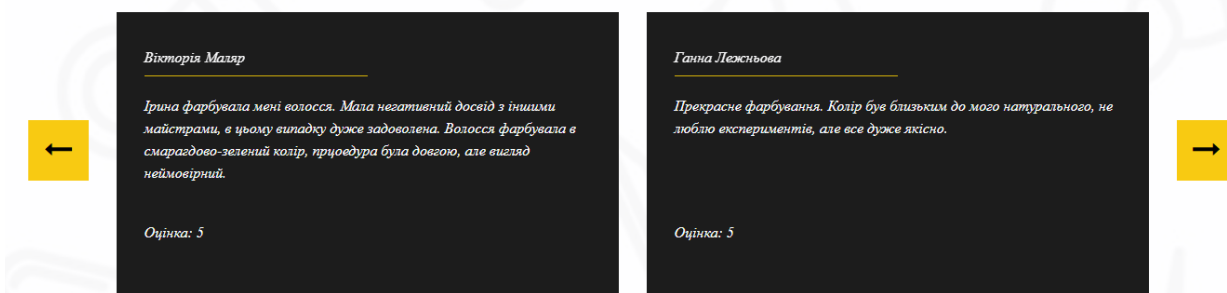


Рисунок 4.2 – Перегляд відгуків про процедуру майстра на сторінці процедури

У свою чергу інтерфейс майстра містить такі розділи:

- розділ «Замовлення» дозволяє отримати доступ до замовлень, які були оформлені щодо процедур авторизованого майстра: надаються дві кнопки «Графік» для відображення всіх замовлень, відсортованих за датою виконання, та «Керування» для відображення і керування (кнопками прийняти і скасувати) всіма замовленнями, адресованими майстру, сортуючи їх за датою створення замовлення;
- розділ «Процедури» дозволяє керувати всіма процедурами майстра (рис. 4.3), при цьому відображаються всі процедури, які може виконувати майстер з кнопками «Редагувати» для переходу до змінювання даних процедури, з можливістю переходу на сторінку процедури майстра при натисканні на фотографію процедури та з кнопкою «Додати» внизу сторінки, що дозволяє додати нову процедуру зі сфери краси, яку тепер виконує майстер;
- розділ «Дати» дозволяє визначити перелік дат, за якими буде працювати майстер, відповідно на них будуть прийматися замовлення;
- розділ «Переписка» дозволяє переглядати повідомлення від різних клієнтів, відправляти їм повідомлення;

– розділ «Профіль» дозволяє відобразити дані про навчання, професії, адреси роботи та стандартний графік роботи за днями тижня майстра, звідси можна переходити на відповідні сторінки, додавати нові позиції та редагувати існуючі.

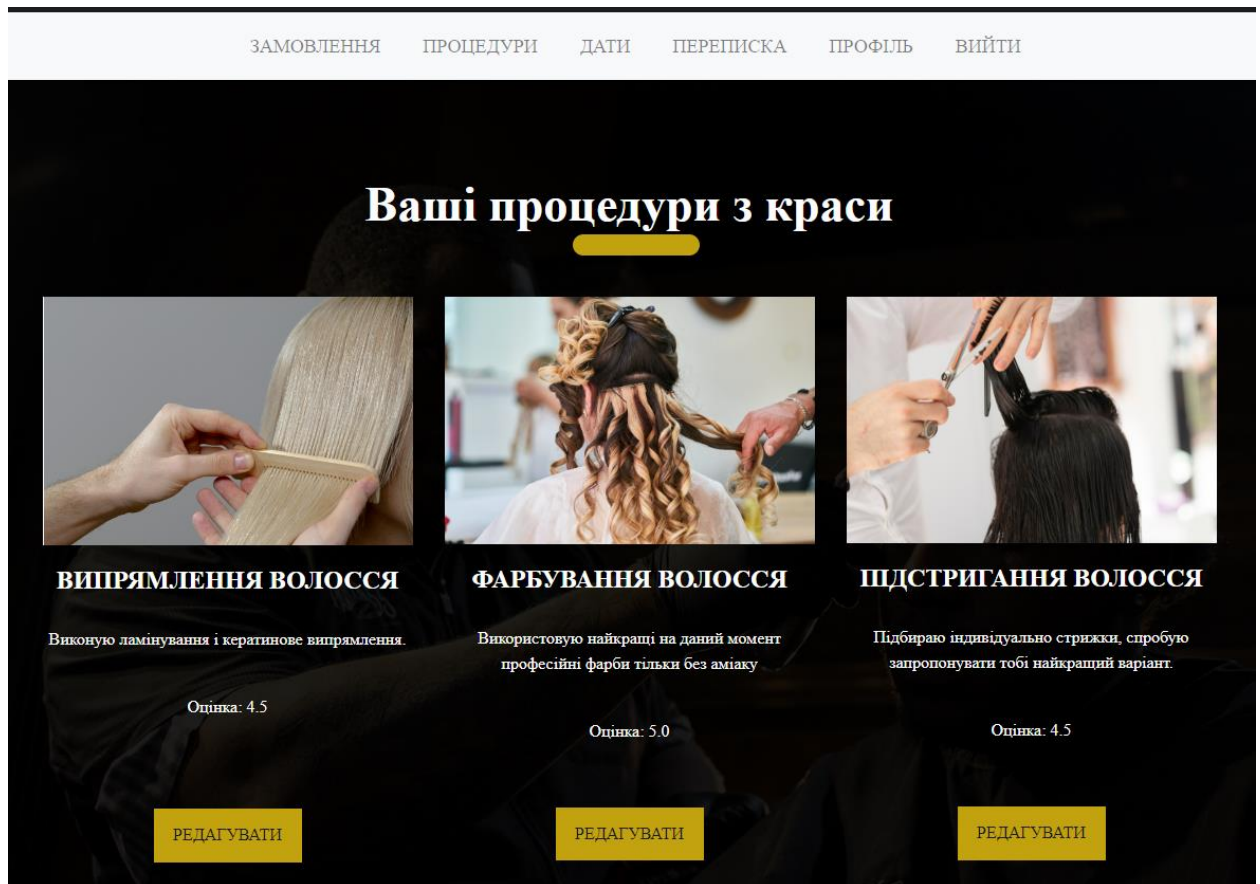


Рисунок 4.3 – Перегляд власних процедур майстром

Інтерфейс адміністратора є стандартним табличним представленням, що дозволяє переглядати, редагувати і додавати відповідні дані.

4.4 Висновки за розділом 4

Описано процес роботи користувачів з програмою, приведено основну логіку доступу до відповідних функцій і даних у програмі.

ВИСНОВКИ

Дипломну роботу виконано, реалізуючи досягнення мети, що полягала в розробці ПЗ бронювання індивідуальних процедур у сфері краси для підтримки роботи індивідуальних майстрів з клієнтами.

Для досягнення мети було виконано огляд предметної області, розглянуто аналоги, виділено основний прототип розробки, яким є застосунок Barb, який дозволяє виконувати пошук не тільки салонів краси, але і майстрів. Основною ідеєю програми, що розробляється, є робота саме з індивідуальними майстрами, а особливостями є можливість створення відгуків про спеціальності майстрів, що дозволяє розподіляти різні результати роботи майстра загалом, та про процедури в сфері краси, що надаються майстром, окрім того особливістю є можливість визначення тих засобів, які використовуються майстром в процедурах, можливість планування роботи майстра і замовлення процедур прямо через застосунок.

За результатами аналізу були визначені сценарії використання програми клієнтом і майстром, відповідно створені діаграми прецедентів, виконано порівняння мов програмування Python і Ruby, обрано Python для реалізації програми, створено структуру БД, порядок функціонування програми, реалізовано і описано класи програми та логіку роботи з програмою.

Програма призначена для пошуку клієнтами майстрів, які здатні надати процедури у сфері краси у власних приміщеннях або індивідуально на умовах клієнта, оформлювати замовлення, а майстрам організовувати власний бізнес без участі великих гравців, керуючи розкладом власної роботи за допомогою керування замовленнями.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Sickler, J. Beauty Industry [Electronic resource] : Cosmetic Market Share, Trends, and Statistics / J. Sickler. – Access mode : <https://terakeet.com/blog/beauty-industry/>.
2. 8 Beauty Ecommerce Trends That Will Define the Industry In 2022 [Electronic resource]. – Access mode : <https://www.yieldify.com/free-guides/beauty-ecommerce-trends/>.
3. Ketabchi, N. Looks That Thrill [Electronic resource] : Inside the Booming Beauty Industry / N. Ketabchi. – Access mode : <https://www.toptal.com/finance/growth-strategy/beauty-industry>.
4. Roberts, R. 2022 Beauty Industry Trends & Cosmetics Marketing [Electronic resource] : Statistics and Strategies for Your Ecommerce Growth / R. Roberts. – Access mode : <https://commonthreadco.com/blogs/coachs-corner/beauty-industry-cosmetics-marketing-ecommerce>.
5. Marchessou, S. Taking a good look at the beauty industry [Electronic resource] / Sophie Marchessou, Emma Spagnuolo. – Access mode : <https://www.mckinsey.com/industries/retail/our-insights/taking-a-good-look-at-the-beauty-industry>.
6. Cosmetics Industry in the U.S. – statistics & facts [Electronic resource]. – Access mode : <https://www.statista.com/topics/1008/cosmetics-industry/>.
7. Barb.ua [Електрон. ресурс] : салони краси, барбершопи, тату салони, косметологи, масажисти, перукарі, майстри манікюру, епіляції, татуажу, макіяжу в Києві. – Режим доступу : <https://barb.ua/uk>.
8. Салон краси бізнес-класу КАФО в Києві (біля станцій метро Осокорки, Позняки) [Електрон. ресурс]. – Режим доступу : <https://kafo.kiev.ua/ua/>.
9. Matthes, E. Python Crash Course [Text] : A Hands-On, Project-Based Introduction to Programming / E. Matthes. – San Francisco : No Starch Press,

2019. – 544 p.

10. Vincent, W. S. Django for Professionals [Text] : Production websites with Python & Django (Welcome to Django) / W. S. Vincent. – Bristol : WelcomeToCode, 2020. – 314 p.

ДОДАТОК А
Технічне завдання

Вступ

Салони краси активно розвиваються, мають достатньо великий попит на власну роботу і є достатньо організованими у своїй діяльності зокрема в нашій країні. Проте не всі потребують послуг саме салонів, деяким клієнтам зручним може бути контактування з індивідуальними майстрами. Саме на тих клієнтів направлена розробка програми в роботі.

A.1 Підстави для розробки

Розробка в межах дипломної кваліфікаційної роботи реалізується на підставі наказу № 102 від 27 квітня 2022 р. за Національним університетом «Запорізька політехніка» за темою «Програмне забезпечення бронювання індивідуальних процедур у сфері краси».

A.2 Призначення розробки

Програма призначена для пошуку клієнтами майстрів, які здатні надати процедури у сфері краси у власних приміщеннях або індивідуально на умовах клієнта, оформлювати замовлення, а майстрам організовувати власний бізнес без участі великих гравців, керуючи розкладом власної роботи за допомогою керування замовленнями.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Функції, які виконуватиме програма за результатами розробки:

- оформлення замовлень процедур у сфері краси від майстра;
- пошук майстрів;
- пошук процедур майстрів;

- перегляд сторінки майстра;
- перегляд процедури майстра;
- внесення майстром власних процедур;
- визначення засобів, які майстер використовує для виконання процедури;
- перегляд переліку процедур, які виконує майстер;
- визначення навчань майстра;
- визначення спеціальностей майстра;
- перегляд розкладу роботи майстра на задану дату;
- створення відгуку про спеціальність майстра;
- перегляд відгуків про майстра;
- створення відгуку про процедуру у виконанні майстра;
- перегляд переліку власних замовлень;
- перегляд переліку замовлень на роботу майстра;
- встановлення власної вартості замовлення майстром;
- скасування замовлення;
- підтвердження замовлення;
- визначення типового графіку роботи майстра;
- визначення роботи за інтервалом дат майстром;
- визначення власних робочих адрес майстром;
- ведення переписки між майстром і клієнтом;
- визначення відсутності засобів для процедури;
- реєстрація і авторизація майстрів і клієнтів.

А.3.2 Умови експлуатації

Рекомендовано для забезпечення роботи програми виділити системного адміністратора, до зобов'язань якого віднести підтримку роботи вебсервера, сервера БД та внесення за потреби базових даних.

А.3.3 Вимоги до складу та параметрів технічних засобів

Технічні засоби в мінімальній конфігурації, достатній для роботи програми, охоплюють 1 Гб пам'яті жорсткого диску, 4 Гб оперативної пам'яті, 4 ядра процесору (частота 2,8 ГГц).

А.4 Порядок контролю та приймання

Остаточна перевірка отриманих результатів виконання роботи здійснюється під час захисту дипломної роботи.

ДОДАТОК Б
Текст програми

```

from django.views.generic.edit import CreateView
from django.views.generic import ListView, UpdateView
from django.views.generic.detail import DetailView
from django.contrib.auth.mixins import LoginRequiredMixin
from django.urls import reverse_lazy
from datetime import date
from .models import Treatment, TreatmentCategory,
MasterTreatmentSpecification, MasterTreatment, TreatmentOrder, Master,
TreatmentOrderItem, Client, MasterTreatmentReview, CityArea,
WorkingAddress, Product, MasterTreatmentProduct, ProductCategory

```

```

class TreatmentCreateView (LoginRequiredMixin, CreateView):
    model = Treatment
    fields = ['treat_title']
    template_name = 'newtreatment.html'

    def get_success_url(self):
        return reverse_lazy('treatmentview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        tcateg = get_object_or_404(TreatmentCategory,
id=self.kwargs['category'])
        kwargs['category'] = tcateg
        categories = TreatmentCategory.objects.all()
        kwargs['categories'] = categories
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        tcateg = get_object_or_404(TreatmentCategory,
id=self.kwargs['category'])
        self.treatmentcategory = tcateg
        return super().form_valid(form)

class TreatmentUpdateView (LoginRequiredMixin, UpdateView):
    model = Treatment
    fields = ['treat_title']
    template_name = 'edittreatment.html'

```

```

    def get_success_url(self):
        return reverse_lazy('treatmentview', kwargs={'pk':
self.object.pk})

```

```

    def get_context_data(self, **kwargs):
        tcateg = get_object_or_404(TreatmentCategory,
id=self.kwargs['category'])
        kwargs['category'] = tcateg
        categories = TreatmentCategory.objects.all()
        kwargs['categories'] = categories
        return super().get_context_data(**kwargs)

```

```

    def form_valid(self, form):
        tcateg = get_object_or_404(TreatmentCategory,
id=self.kwargs['category'])
        self.treatmentcategory = tcateg
        return super().form_valid(form)

```

```

class CategoryTreatmentsView (ListView):
    model = Treatment
    template_name = 'categorytreatments.html'

    def get_queryset(self):
        tcateg = self.request.GET.get('category')
        treatments =
Treatment.objects.filter(treatmentcategory = tcateg)
        treatments = treatments.order_by('treat_title')
        return treatments

```

```

class TreatmentsView (ListView):
    model = Treatment
    template_name = 'treatments.html'

    def get_queryset(self):
        treatments = Treatment.objects.all()
        treatments = treatments.order_by('treat_title')
        return treatments

```

```

class YourTreatmentsView (ListView):
    model = Treatment
    template_name = 'yourtreatments.html'

    def get_queryset(self):
        masterid = self.request.user
        masterid = masterid.id
        masterdata = Master.objects.filter(id = masterid)
        mtreatments = MasterTreatment.objects.filter(master =
masterdata)

        mtreatments =
mtreatments.order_by('treatment__treat_title')
        return mtreatments

class MasterTreatmentSpecsView (ListView):
    model = MasterTreatmentSpecification
    template_name = 'mastertreatmentspecs.html'

    def get_queryset(self):
        masterid = self.request.user
        masterid = masterid.id
        masterdata = Master.objects.filter(id = masterid)
        mtreatments =
MasterTreatmentSpecification.objects.filter(mastertreatment__master =
masterdata)

        mtreatments = mtreatments.order_by('id')
        return mtreatments

class ProductCreateView (LoginRequiredMixin, CreateView):
    model = Product
    fields = ['title', 'producer', 'country']
    template_name = 'newproduct.html'

    def get_success_url(self):
        return reverse_lazy('productsview', kwargs={'pk':
self.object.productcategory.pk})

    def get_context_data(self, **kwargs):

```

```

        categ = get_object_or_404(ProductCategory,
id=self.kwargs['category'])
        kwargs['category'] = categ
        categories = ProductCategory.objects.all()
        kwargs['categories'] = categories
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        categ = get_object_or_404(ProductCategory,
id=self.kwargs['category'])
        self.productcategory = categ
        return super().form_valid(form)

class ProductUpdateView (LoginRequiredMixin, UpdateView):
    model = Product
    fields = ['title', 'producer', 'country']
    template_name = 'editproduct.html'

    def get_success_url(self):
        return reverse_lazy('productsview', kwargs={'pk':
self.object.productcategory.pk})

    def get_context_data(self, **kwargs):
        categ = get_object_or_404(ProductCategory,
id=self.kwargs['category'])
        kwargs['category'] = categ
        categories = ProductCategory.objects.all()
        kwargs['categories'] = categories
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        categ = get_object_or_404(ProductCategory,
id=self.kwargs['category'])
        self.productcategory = categ
        return super().form_valid(form)

class MasterTreatmentsView (ListView):
    model = MasterTreatment
    template_name = 'mastertreatments.html'

```

```

def get_queryset(self):
    masterid = self.request.GET.get('master')
    masterdata = Master.objects.filter(id = masterid)
    mtreatments = MasterTreatment.objects.filter(master =
masterdata)

    mtreatments = mtreatments.order_by('id')
    return mtreatments


class CategoryProductsView (ListView):
    model = Product
    template_name = 'categoryproducts.html'

    def get_queryset(self):
        categ = self.request.GET.get('category')
        categ = ProductCategory.objects.get(id = categ)
        products = Product.objects.filter(productcategory =
categ)

        products = products.order_by('title')
        return products

    def get_context_data(self, **kwargs):
        products = super(CategoryProductsView,
self).get_context_data(**kwargs)
        categ = self.request.GET.get('category')
        categ = ProductCategory.objects.get(id = categ)
        products ['category'] = categ
        categ = ProductCategory.objects.all()
        products ['categories'] = categ
        return products


class ProductCategoryCreateView (LoginRequiredMixin,
CreateView):
    model = ProductCategory
    fields = ['category_title']
    template_name = 'newproductcategory.html'

    def get_success_url(self):
        return reverse_lazy('productcategoriesview',
kwargs={'pk': self.object.pk})

```

```

class ProductCategoryUpdateView (LoginRequiredMixin,
UpdateView):
    model = ProductCategory
    fields = ['category_title']
    template_name = 'editproductcategory.html'

    def get_success_url(self):
        return reverse_lazy('productcategoriesview',
kwargs={'pk': self.object.pk})

class ProductCategoriesView (ListView):
    model = ProductCategory
    template_name = 'productcategories.html'

class ProductsView (ListView):
    model = Product
    template_name = 'products.html'

class MasterTreatmentCreateView (LoginRequiredMixin,
CreateView):
    model = MasterTreatment
    fields = ['specification']
    template_name = 'newmastertreatment.html'

    def get_success_url(self):
        return reverse_lazy('mastertreatmentview',
kwargs={'pk': self.object.pk})

    def get_context_data(self, **kwargs):
        treatment = get_object_or_404(Treatment,
id=self.kwargs['treatment'])
        kwargs['treatment'] = treatment
        categories = TreatmentCategory.objects.all()
        kwargs['categories'] = categories
        treatments = Treatment.objects.all()
        kwargs['treatments'] = treatments
        return super().get_context_data(**kwargs)

```

```

    def form_valid(self, form):
        treatment = get_object_or_404(Treatment,
id=self.kwargs['treatment'])
        self.treatment = treatment
        masterid = self.request.user
        masterid = masterid.id
        masterdata = Master.objects.filter(id = masterid)
        self.master = masterdata
        return super().form_valid(form)

```

```

class MasterTreatmentUpdateView (LoginRequiredMixin,
UpdateView):

```

```

    model = MasterTreatment
    fields = ['specification']
    template_name = 'editmastertreatment.html'

```

```

    def get_success_url(self):
        return reverse_lazy('mastertreatmentview',
kwargs={'pk': self.object.pk})

```

```

    def get_context_data(self, **kwargs):
        treatment = get_object_or_404(Treatment,
id=self.kwargs['treatment'])
        kwargs['treatment'] = treatment
        categories = TreatmentCategory.objects.all()
        kwargs['categories'] = categories
        treatments = Treatment.objects.all()
        kwargs['treatments'] = treatments
        return super().get_context_data(**kwargs)

```

```

    def form_valid(self, form):
        treatment = get_object_or_404(Treatment,
id=self.kwargs['treatment'])
        self.treatment = treatment
        return super().form_valid(form)

```

```

class MasterTreatmentSpecificationCreateView
(LoginRequiredMixin, CreateView):
    model = MasterTreatmentSpecification

```

```

fields = ['specification', 'price', 'duration']
template_name = 'newmastertreatmentspec.html'

def get_success_url(self):
    return reverse_lazy('mastertreatmentview',
kwargs={'pk': self.object.mastertreatment.pk})

def get_context_data(self, **kwargs):
    mtreatment = get_object_or_404(MasterTreatment,
id=self.kwargs['treatment'])
    kwargs['treatment'] = mtreatment
    masterid = self.request.user
    masterid = masterid.id
    masterdata = Master.objects.filter(id = masterid)
    mastertreatments =
MasterTreatment.objects.filter(master = masterdata)
    kwargs['mastertreatments'] = mastertreatments
    return super().get_context_data(**kwargs)

def form_valid(self, form):
    mtreatment = get_object_or_404(MasterTreatment,
id=self.kwargs['treatment'])
    self.mastertreatment = mtreatment
    products = str(self.request.POST.get('products'))
    products = products.split(',')
    productobjects = []
    for prod in products:
        product = Product.objects.get(id = prod)
        productobjects.append(product)
    for prod in productobjects:
        MasterTreatmentProduct.objects.create(mastertreatmentspecificatio
n = self.object.pk, product = prod)
    return super().form_valid(form)

class MasterTreatmentSpecificationUpdateView
(LoginRequiredMixin, UpdateView):
    model = MasterTreatment
    fields = ['specification', 'price', 'duration']

```

```

template_name = 'editmastertreatment.html'

def get_success_url(self):
    return reverse_lazy('mastertreatmentview',
kwargs={'pk': self.object.pk})

def get_context_data(self, **kwargs):
    mtreatment = get_object_or_404(MasterTreatment,
id=self.kwargs['treatment'])
    kwargs['treatment'] = mtreatment
    masterid = self.request.user
    masterid = masterid.id
    masterdata = Master.objects.filter(id = masterid)
    mastertreatments =
MasterTreatment.objects.filter(master = masterdata)
    kwargs['mastertreatments'] = mastertreatments
    products = str(self.request.POST.get('products'))
    products = products.split(',')
    productobjects = []
    for prod in productobjects:
        product = Product.objects.get(id = prod)
        productobjects.append(product)
    products =
MasterTreatmentProduct.objects.filter(mastertreatmentspecification =
self.object.pk)
    for prod in productobjects:
        res = products.filter(product = prod)
        if not res:

        MasterTreatmentProduct.objects.create(mastertreatmentspecificatio
n = self.object.pk, product = prod, availability = 1)
    for prod in products:
        res = productobjects.filter(id = prod.product.id)
        if not res:

        MasterTreatmentProduct.objects.filter(mastertreatmentspecificatio
n = self.object.pk, product = prod).delete()
    return super().get_context_data(**kwargs)

```

```

    def form_valid(self, form):
        mtreatment = get_object_or_404(MasterTreatment,
id=self.kwargs['treatment'])
        self.mastertreatment = mtreatment
        return super().form_valid(form)

class MasterTreatmentDetailView (DetailView):
    model = MasterTreatment
    template_name = 'mastertreatment.html'

    def get_context_data(self, **kwargs):
        treatmentdataformaster =
super().get_context_data(**kwargs)
        mtreatments =
MasterTreatmentSpecification.objects.filter(mastertreatment =
self.object.pk)
        mtreatments = mtreatments.order_by('specification')
        treatmentdataformaster ['specifications'] =
mtreatments
        reviews =
MasterTreatmentReview.objects.filter(mastertreatment = self.object.pk)
        reviews = reviews.order_by('-review_date')
        treatmentdataformaster ['reviews'] = reviews
        return treatmentdataformaster

class MasterTreatmentSpecificationDetailView (DetailView):
    model = MasterTreatmentSpecification
    template_name = 'mastertreatmentspec.html'

    def get_context_data(self, **kwargs):
        treatmentdataformaster =
super().get_context_data(**kwargs)
        spec =
TreatmentOrderItem.objects.filter(mastertreatmentspecification =
self.object.pk)
        orders = TreatmentOrder.objects.filter(items__contains
= spec)
        orders = orders.order_by('-treatment_time')
        treatmentdataformaster ['orders'] = orders

```

```

        products =
MasterTreatmentProduct.objects.filter(mastertreatmentsspecification =
spec)

        products = products.order_by('product__title')
        treatmentdataformaster ['products'] = products
        return treatmentdataformaster

class MasterTreatmentProductCreateView (LoginRequiredMixin,
CreateView):
    model = MasterTreatmentProduct
    fields = ['availability']
    template_name = 'newmastertreatmentproduct.html'

    def get_success_url(self):
        return reverse_lazy('mastertreatmentsspecview',
kwargs={'pk': self.object.mastertreatmentsspecification.pk})

    def get_context_data(self, **kwargs):
        spec = get_object_or_404(MasterTreatmentSpecification,
id=self.kwargs['spec'])
        kwargs['spec'] = spec
        product = get_object_or_404(Product,
id=self.kwargs['product'])
        kwargs['product'] = product
        masterid = self.request.user
        masterid = masterid.id
        masterdata = Master.objects.filter(id = masterid)
        mastertreatments =
MasterTreatmentSpecification.objects.filter(mastertreatment__master =
masterdata)

        kwargs['mastertreatments'] = mastertreatments
        products = Product.objects.all()
        kwargs['products'] = products
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        spec = get_object_or_404(MasterTreatmentSpecification,
id=self.kwargs['spec'])
        self.mastertreatmentsspecification = spec

```

```

        product = get_object_or_404(Product,
id=self.kwargs['product'])
        self.product = product
        return super().form_valid(form)

class MasterTreatmentProductUpdateView (LoginRequiredMixin,
UpdateView):
    model = MasterTreatmentProduct
    fields = ['availability']
    template_name = 'editmastertreatmentproduct.html'

    def get_success_url(self):
        return reverse_lazy('mastertreatmentspecview',
kwargs={'pk': self.object.mastertreatmentspecification.pk})

    def get_context_data(self, **kwargs):
        spec = get_object_or_404(MasterTreatmentSpecification,
id=self.kwargs['spec'])
        kwargs['spec'] = spec
        product = get_object_or_404(Product,
id=self.kwargs['product'])
        kwargs['product'] = product
        masterid = self.request.user
        masterid = masterid.id
        masterdata = Master.objects.filter(id = masterid)
        mastertreatments =
MasterTreatmentSpecification.objects.filter(mastertreatment__master =
masterdata)

        kwargs['mastertreatments'] = mastertreatments
        products = Product.objects.all()
        kwargs['products'] = products
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        spec = get_object_or_404(MasterTreatmentSpecification,
id=self.kwargs['spec'])
        self.mastertreatmentspecification = spec
        product = get_object_or_404(Product,
id=self.kwargs['product'])

```

```

        self.product = product
        return super().form_valid(form)

class TreatmentOrderCreateView (LoginRequiredMixin, CreateView):
    model = TreatmentOrder
    fields = ['treatment_time', 'client_address']
    template_name = 'newtreatmentorder.html'

    def get_success_url(self):
        return reverse_lazy('treatmentorderview',
kwargs={'pk': self.object.pk})

    def get_context_data(self, **kwargs):
        cityarea = get_object_or_404(CityArea,
id=self.kwargs['cityarea'])
        kwargs['cityarea'] = cityarea
        address = get_object_or_404(WorkingAddress,
id=self.kwargs['address'])
        kwargs['address'] = address
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        clientid = self.request.user
        clientid = clientid.id
        clientdata = Client.objects.filter(id = clientid)
        self.client = clientdata
        cityarea = get_object_or_404(CityArea,
id=self.kwargs['cityarea'])
        self.cityarea = cityarea
        address = get_object_or_404(WorkingAddress,
id=self.kwargs['address'])
        self.workingaddress = address
        self.state = 1
        treatments = str(self.request.POST.get('treatments'))
        treatments = treatments.split(',')
        treatmentobjects = []
        self.order_date = date.today()
        dur = None
        for tspec in treatments:

```

```

        spec =
MasterTreatmentSpecification.objects.get(id = tspec)
        treatmentobjects.append(spec)
        dur = spec.duration
        fdur = dur
        price = 0
        for tspec in treatmentobjects:
            TreatmentOrderItem.objects.create(treatmentorder
= self.object.pk, mastertreatmentSpecification = tspec)
            dur = dur + tspec.duration
            price = price + rspec.price
        dur = dur - fdur
        self.price = price
        self.treatment_end = self.treatment_time + dur
        return super().form_valid(form)

class MasterTreatmentOrdersView (ListView):
    model = TreatmentOrder
    template_name = 'mastertreatmentorders.html'

    def get_queryset(self):
        masterid = self.request.user
        masterid = masterid.id
        masterdata = Master.objects.filter(id = masterid)
        items =
TreatmentOrderItem.objects.filter(mastertreatmentSpecification__master
treatment__master = masterdata)
        orders = TreatmentOrder.objects.filter(items__contains
= items)

        statep = self.kwargs['state']
        orders = orders.filter(state = statep)
        orders = orders.order_by('-order_date')
        return orders

class TreatmentOrderUpdateView (LoginRequiredMixin, UpdateView):
    model = TreatmentOrder
    fields = ['treatment_time', 'client_address']
    template_name = 'edittreatmentorder.html'

```

```

def get_success_url(self):
    return reverse_lazy('treatmentorderview',
kwargs={'pk': self.object.pk})

def get_context_data(self, **kwargs):
    cityarea = get_object_or_404(CityArea,
id=self.kwargs['cityarea'])
    kwargs['cityarea'] = cityarea
    address = get_object_or_404(WorkingAddress,
id=self.kwargs['address'])
    kwargs['address'] = address
    return super().get_context_data(**kwargs)

def form_valid(self, form):
    cityarea = get_object_or_404(CityArea,
id=self.kwargs['cityarea'])
    self.cityarea = cityarea
    address = get_object_or_404(WorkingAddress,
id=self.kwargs['address'])
    self.workingaddress = address
    treatments = str(self.request.POST.get('treatments'))
    treatments = treatments.split(',')
    treatmentobjects = []
    for tspec in treatments:
        spec =
MasterTreatmentSpecification.objects.get(id = tspec)
        treatmentobjects.append(spec)
    treatments =
TreatmentOrderItem.objects.filter(treatmentorder = self.object.pk)
    for tspec in treatmentobjects:
        res =
treatments.filter(mastertreatmentsspecification = tspec)
        if not res:

TreatmentOrderItem.objects.create(treatmentorder =
self.object.pk, mastertreatmentsspecification = tspec)
        for treat in treatments:
            res = treatmentobjects.filter(id =
treat.mastertreatmentsspecification.id)

```

```
if not res:
```

```
    TreatmentOrderItem.objects.filter(treatmentorder =
self.object.pk, mastertreatmentsspecification =
treat.mastertreatmentsspecification.id).delete()
    return super().form_valid(form)
```

```
class MasterTreatmentOrderUpdateView (LoginRequiredMixin,
UpdateView):
```

```
    model = TreatmentOrder
```

```
    fields = ['treatment_time', 'client_address', 'price',
'treatment_time', 'treatment_end']
```

```
    template_name = 'mastereditttreatmentorder.html'
```

```
    def get_success_url(self):
```

```
        return reverse_lazy('treatmentorderview',
kwargs={'pk': self.object.pk})
```

```
    def get_context_data(self, **kwargs):
```

```
        cityarea = get_object_or_404(CityArea,
id=self.kwargs['cityarea'])
```

```
        kwargs['cityarea'] = cityarea
```

```
        address = get_object_or_404(WorkingAddress,
id=self.kwargs['address'])
```

```
        kwargs['address'] = address
```

```
        return super().get_context_data(**kwargs)
```

```
    def form_valid(self, form):
```

```
        cityarea = get_object_or_404(CityArea,
id=self.kwargs['cityarea'])
```

```
        self.cityarea = cityarea
```

```
        address = get_object_or_404(WorkingAddress,
id=self.kwargs['address'])
```

```
        self.workingaddress = address
```

```
        treatments = str(self.request.POST.get('treatments'))
```

```
        treatments = treatments.split(',')
```

```
        treatmentobjects = []
```

```
        for tspec in treatments:
```

```

        spec =
MasterTreatmentSpecification.objects.get(id = tspec)
        treatmentobjects.append(spec)
        treatments =
TreatmentOrderItem.objects.filter(treatmentorder = self.object.pk)
        for tspec in treatmentobjects:
            res =
treatments.filter(mastertreatmentsspecification = tspec)
            if not res:

```

```

        TreatmentOrderItem.objects.create(treatmentorder =
self.object.pk, mastertreatmentsspecification = tspec)
        for treat in treatments:
            res = treatmentobjects.filter(id =
treat.mastertreatmentsspecification.id)
            if not res:

```

```

        TreatmentOrderItem.objects.filter(treatmentorder =
self.object.pk, mastertreatmentsspecification =
treat.mastertreatmentsspecification.id).delete()
        return super().form_valid(form)

```

```

class TreatmentOrderCancelView (LoginRequiredMixin, UpdateView):
    model = TreatmentOrder
    template_name = 'canceltreatmentorder.html'

    def get_success_url(self):
        return reverse_lazy('treatmentorderview',
kwargs={'pk': self.object.pk})

    def form_valid(self, form):
        self.state = 0
        return super().form_valid(form)

```

```

class TreatmentOrderUpView (LoginRequiredMixin, UpdateView):
    model = TreatmentOrder
    template_name = 'uptreatmentorder.html'

```

```

    def get_success_url(self):
        return reverse_lazy('treatmentorderview',
                               kwargs={'pk': self.object.pk})

    def form_valid(self, form):
        self.state = 2
        return super().form_valid(form)

class YourTreatmentOrdersView (ListView):
    model = TreatmentOrder
    template_name = 'yourtreatmentorders.html'

    def get_queryset(self):
        clientid = self.request.user
        clientid = clientid.id
        clientdata = Client.objects.filter(id = clientid)
        orders = TreatmentOrder.objects.filter(client =
clientdata)

        way = self.kwargs['one']
        if way:
            orders = orders.order_by('-treatment_time')
        else:
            orders = orders.order_by('-order_date')
        return mtreatments

class MasterGraphicView (ListView):
    model = TreatmentOrder
    template_name = 'mastergraphic.html'

    def get_queryset(self):
        masterid = self.request.user
        masterid = masterid.id
        masterdata = Master.objects.filter(id = masterid)
        items =
TreatmentOrderItem.objects.filter(mastertreatmentspecification__master
treatment__master = masterdata)
        orders = TreatmentOrder.objects.filter(items__contains
= items)

```

```

        orders = orders.order_by('-treatment_time')
        return orders

class MasterTreatmentReviewCreateView (LoginRequiredMixin,
CreateView):
    model = MasterTreatmentReview
    fields = ['availability']
    template_name = 'newmastertreatreview.html'

    def get_success_url(self):
        return reverse_lazy('mastertreatmentreview',
kwargs={'pk': self.object.pk})

    def get_context_data(self, **kwargs):
        mtreatment = get_object_or_404(MasterTreatment,
id=self.kwargs['treatment'])
        kwargs['treatment'] = mtreatment
        clientid = self.request.user
        clientid = clientid.id
        clientid = Client.objects.get(id = clientid)
        orders = TreatmentOrder.objects.filter(client =
clientid)

        treatments = []
        for order in orders:
            for item in order.items:
                if item not in treatments:

treatments.append(item.mastertreatmentsspecification.mastertreatme
nt)

        kwargs['treatments'] = treatments
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        clientid = self.request.user
        clientid = clientid.id
        clientid = Client.objects.get(id = clientid)
        self.client = clientid

```

```

        treatment = get_object_or_404(MasterTreatment,
id=self.kwargs['treatment'])
        self.mastertreatment = treatment
        return super().form_valid(form)

```

```

class SearchTreatmentView (ListView):
    model = MasterTreatment
    template_name = 'treatsearch.html'

    def get_queryset(self):
        categ = self.kwargs['category']
        categ = get_object_or_404(TreatmentCategory, id =
categ)

        treat = self.kwargs['treatment']
        treat =
Treatment.objects.filter(treat_title__icontains = treat)
        minscore = self.kwargs['score']
        cityarea = self.kwargs['area']
        cityarea = get_object_or_404(CityArea, id = cityarea)
        mastertreatments =
MasterTreatment.objects.filter(treatment__reatmentcategory = categ)
        if treat:
            mastertreatments =
mastertreatments.filter(treatment__treat_title__icontains = treat)
            if minscore:
                mastertreatments =
mastertreatments.filter(score__gte = minscore)
            if cityarea:
                mastertreatments =
mastertreatments.filter(master__cityareas__contains = cityarea)
            mastertreatments = mastertreatments.order_by('score')
        return mastertreatments

    def get_context_data(self, **kwargs):
        searchdata = super(SearchTreatmentView,
self).get_context_data(**kwargs)
        treatmentcategory = self.kwargs['category']
        cityarea = self.kwargs['area']

```

```

        treatmentcategory =
get_object_or_404(TreatmentCategory, id = treatmentcategory)
        cityarea = get_object_or_404(CityArea, id = cityarea)
        searchdata ['category'] = treatmentcategory
        searchdata ['area'] = cityarea
        cities = City.objects.all()
        kwargs['cities'] = cities
        cityareas = CityArea.objects.all()
        kwargs['cityareas'] = cityareas
        return searchdata

```

```

class YourMasterTreatmentReviewsListView (ListView):
    model = MasterTreatmentReview
    template_name = 'yourtreatmentreviews.html'

    def get_queryset(self):
        clientid = self.request.user
        clientid = clientid.id
        clientdata = Client.objects.filter(id = clientid)
        reviews = MasterTreatmentReview.objects.filter(client
= clientdata)
        reviews = reviews.order_by('-review_date')
        return reviews

```

```

class MasterTreatmentReviewsListView (ListView):
    model = MasterTreatmentReview
    template_name = 'treatmentreviews.html'

    def get_queryset(self):
        mtreatment = MasterTreatment.objects.get(id =
self.request.GET.get('mtreatment'))
        reviews =
MasterTreatmentReview.objects.filter(mastertreatment = mtreatment)
        reviews = reviews.order_by('-review_date')
        return reviews

    def get_context_data(self, **kwargs):
        reviewsdata = super(MasterTreatmentReviewsListView,
self).get_context_data(**kwargs)

```

```
        mtreatment = MasterTreatment.objects.get(id =  
self.request.GET.get('mtreatment'))  
        reviewsdata ['mtreatment'] = mtreatment  
        return reviewsdata
```

ДОДАТОК В
Слайди презентації



Рисунок В.1 – Слайд № 1

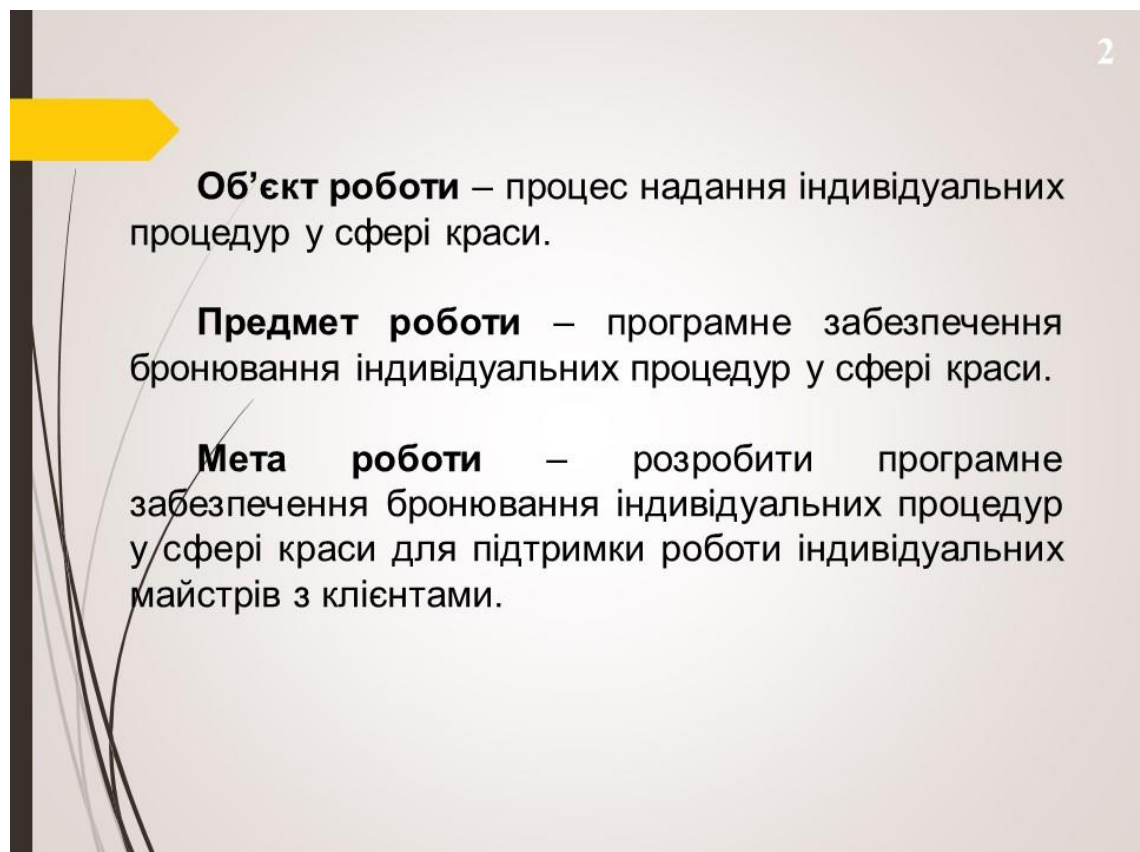


Рисунок В.2 – Слайд № 2

Завдання роботи

3

- аналіз предметної області;
- проєктування бази даних;
- проєктування програмного забезпечення бронювання індивідуальних процедур у сфері краси;
- програмна реалізація програмного забезпечення бронювання індивідуальних процедур у сфері краси.

Рисунок В.3 – Слайд № 3

Результати аналізу предметної області

4

Критерій	Barb	Дипломна програма
Пошук процедур у сфері краси	+	+
Пошук майстрів	+	+
Пошук салонів	+	–
Перегляд процедури за категоріями	+	+
Відгуки	Про майстрів	Про спеціальності майстрів та про процедури
Замовлення процедур	–	+
Планування роботи майстра	–	+
Визначення засобів виконання процедур	–	+
Відправлення повідомлення майстру	+	+

Рисунок В.4 – Слайд № 4

Вибір мови програмування			
Критерій	Python	Ruby	5
Об'єктно-орієнтована мова програмування	Так	Так	
Простота вивчення мови	Вища	Менша	
Високорівнева мова	Так	Так	
Підтримка множинного наслідування	Так	Ні	
Вебфреймворк	Django	Ruby on Rails	
Готові бібліотеки для повторного використання	Більше	Менше	

Рисунок В.5 – Слайд № 5

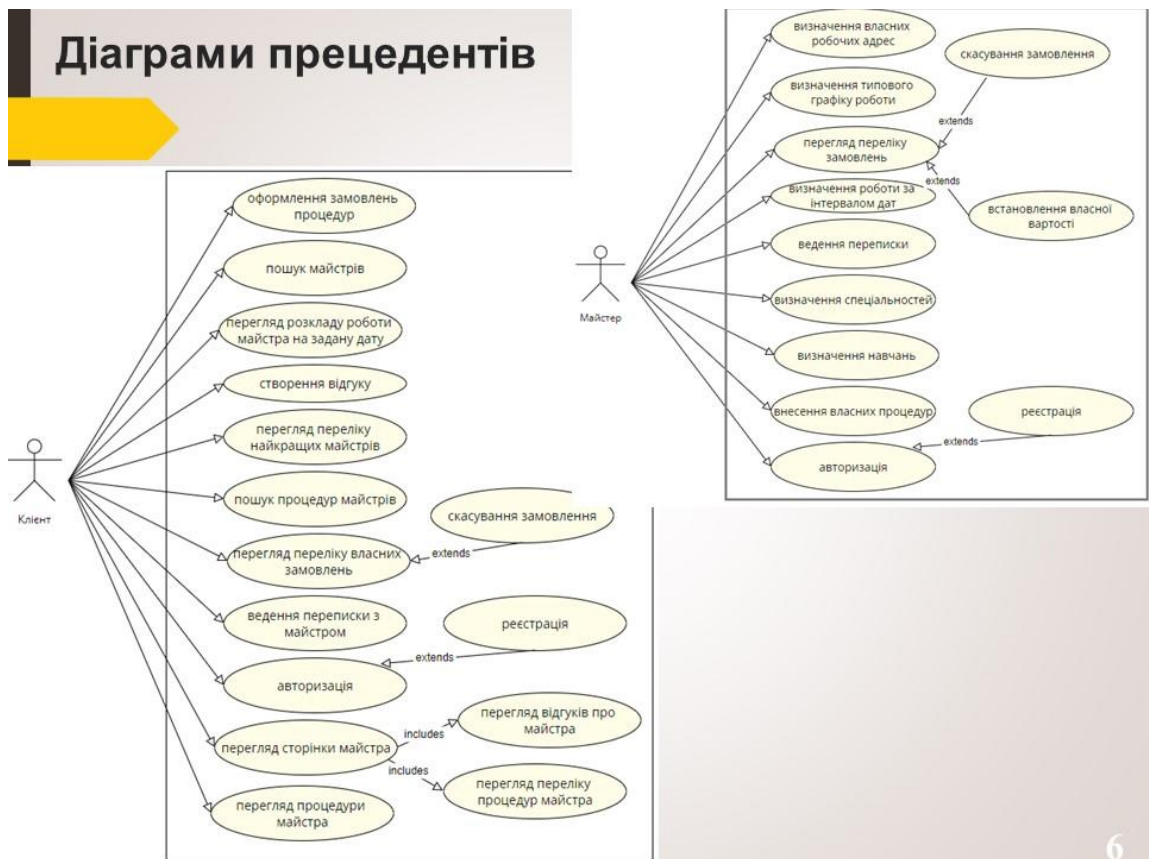


Рисунок В.6 – Слайд № 6

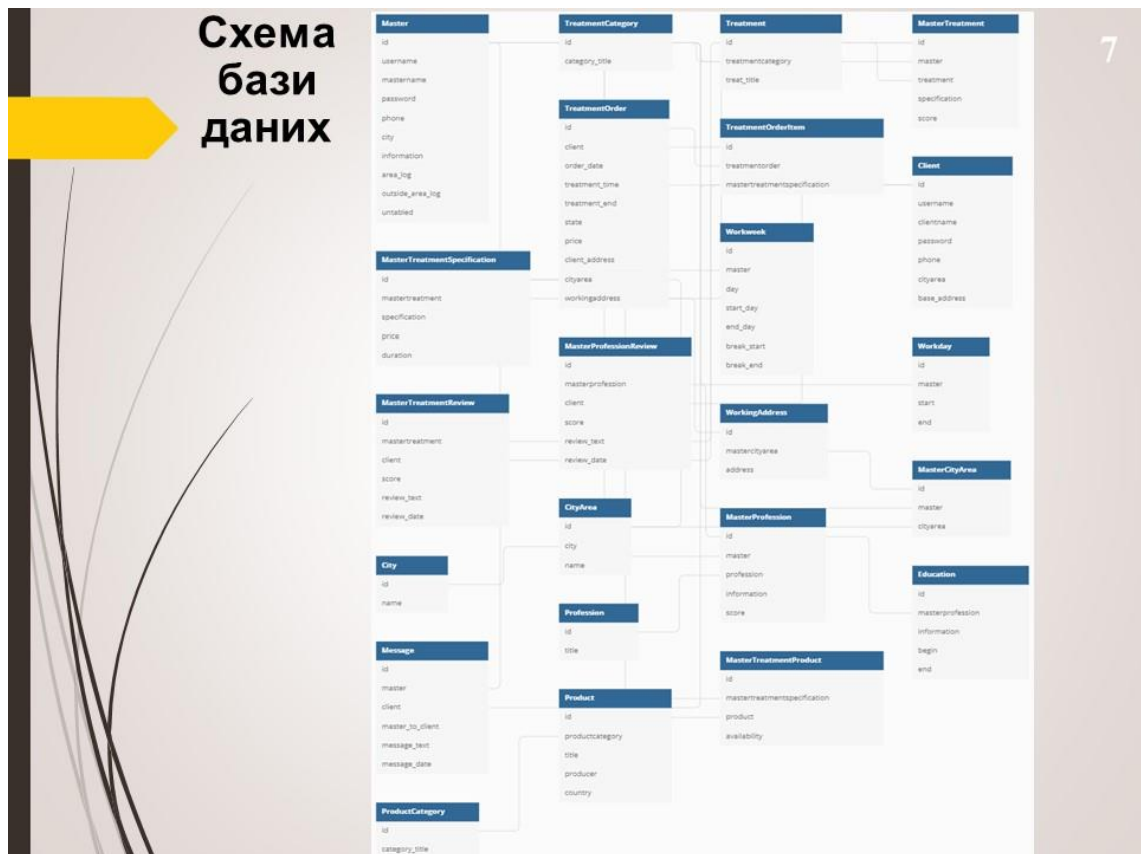


Рисунок В.7 – Слайд № 7



Рисунок В.8 – Слайд № 8

Пошук процедур у сфері краси у програмі

9

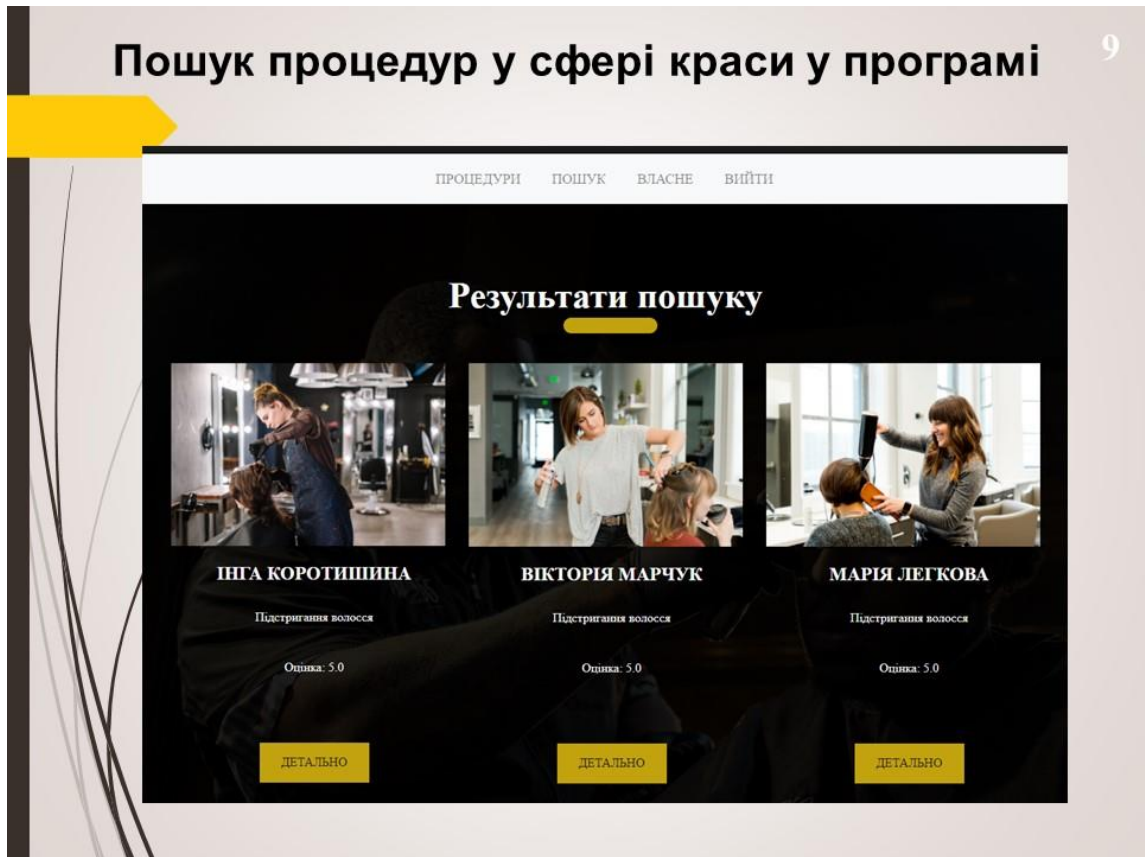


Рисунок В.9 – Слайд № 9

Перегляд відгуків про процедуру у виконанні майстра в програмі

10

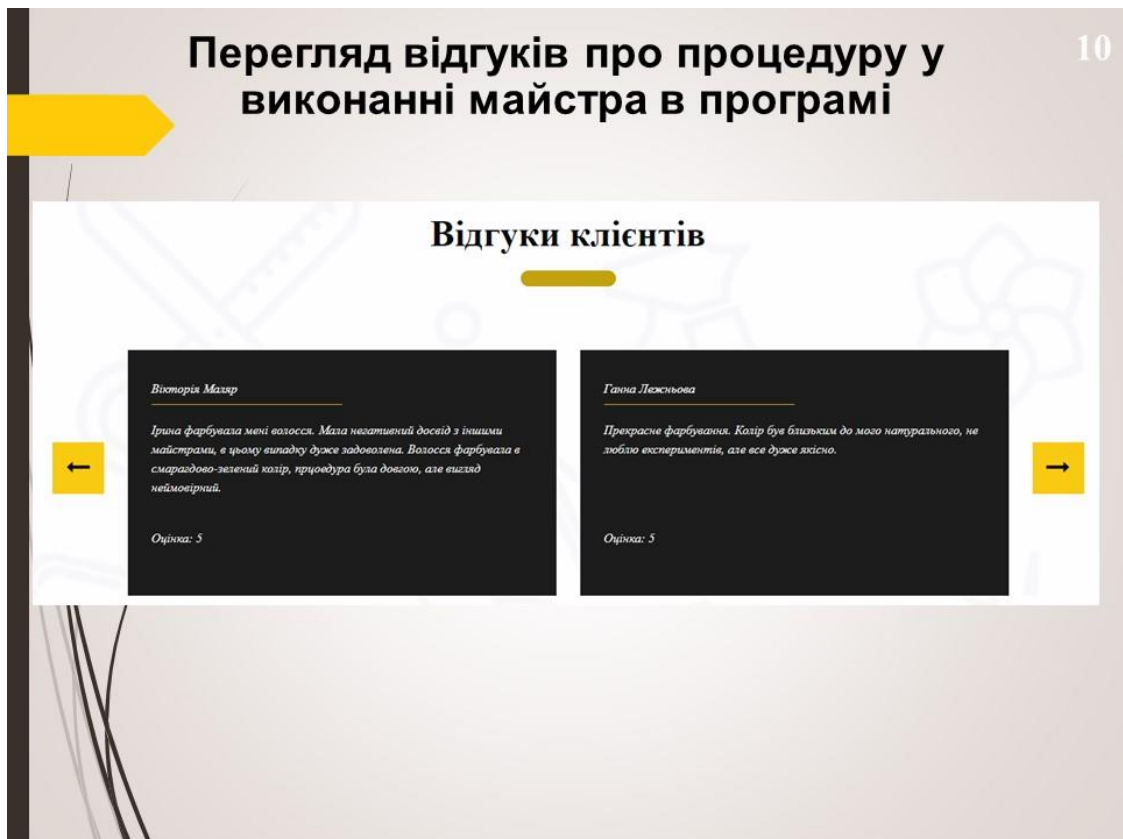


Рисунок В.10 – Слайд № 10

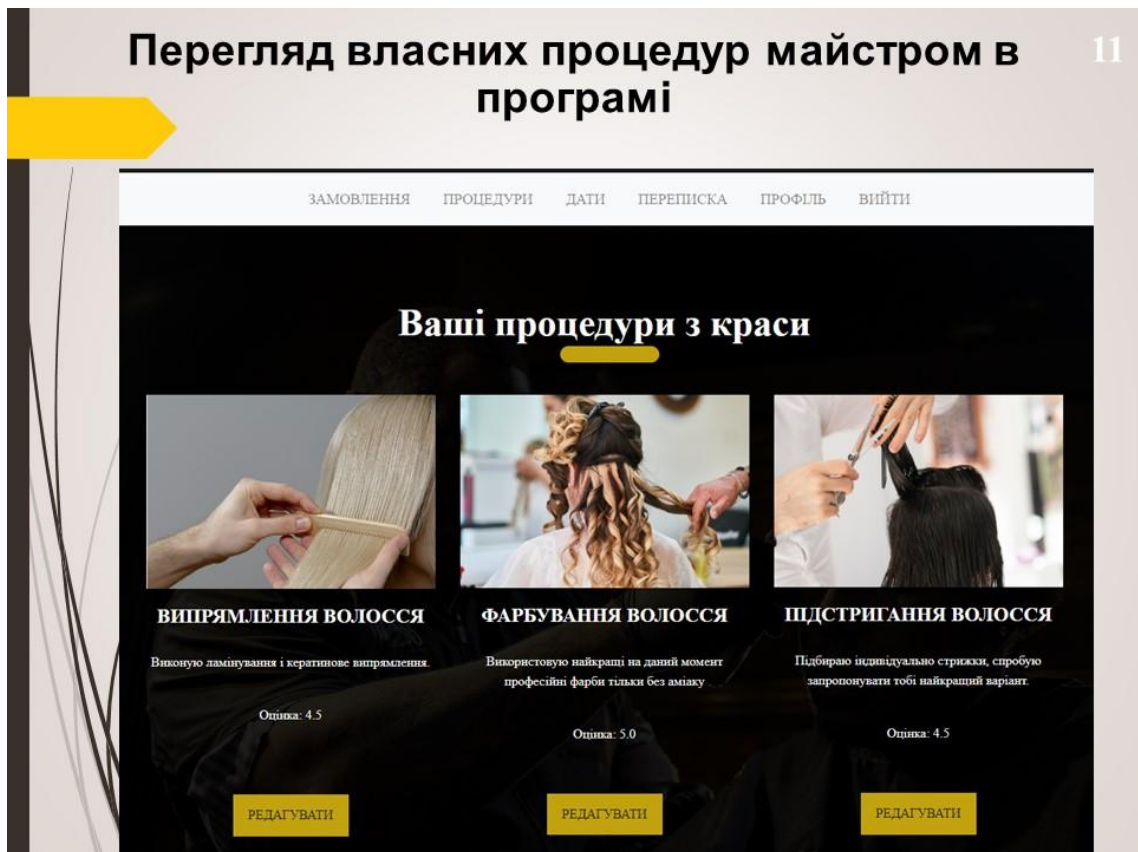


Рисунок В.11 – Слайд № 11

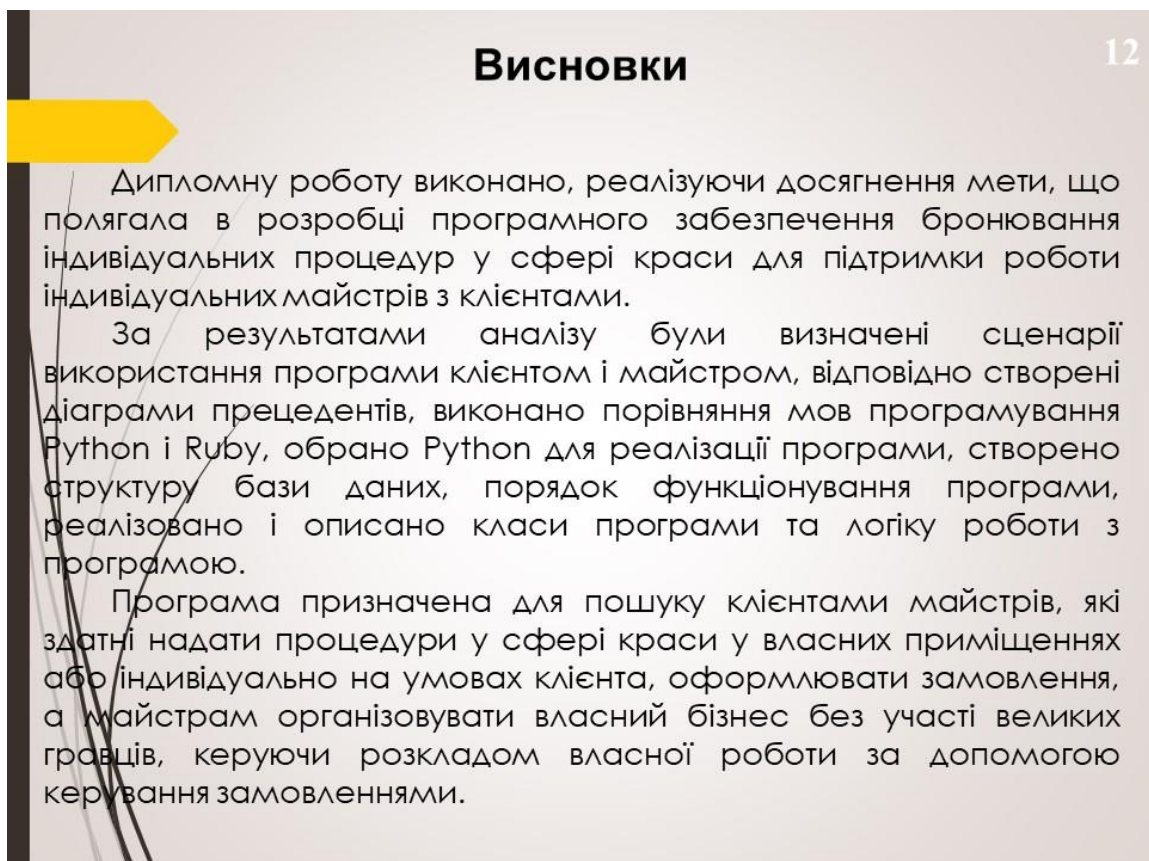


Рисунок В.12 – Слайд № 12