МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ
(підпс)
червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: «Інформаційна система надання послуг перукарського салону»

здобувача групи ІН - 93 Волков Ілля Костянтинович

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Волков Ілля
_____
(підпис)

Керівник
старший викладач, кандидат
фізико-математичних наук Олексієнко Г.А. _____
(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпи)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-93 Волков Ілля Костянтинович

1. Тема роботи: «Інформаційна технологія прогнозування курсу валют»
затверджую наказом по СумДУ від *«01» червня 2023 р. № 0475-VI*

2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*

3. Вхідні дані до кваліфікаційної роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження.
2) Огляд технологій, що використовуються для створення Інформаційної системи обліку користувачів салону краси. 3) Розробка Інформаційної системи обліку 4) Аналіз результатів.

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

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
	Аналіз проблеми предметної області, постановка й формування завдань дослідження.		
	Огляд технологій, що використовуються для створення Інформаційної системи обліку користувачів салону краси.		
	Розробка Інформаційної системи обліку		
	Аналіз результатів.		
	Оформлення пояснювальної записки до кваліфікаційної роботи		

Здобувач вищої освіти _____ Керівник _____

(підпис) (підпис)

АНОТАЦІЯ

Записка: 68стр., 9рис., 1додаток, 10 літературних джерел, 1 таблиця.

Обґрунтування актуальності теми роботи - робота над розробкою та впровадженням інформаційної системи обліку користувачів салону краси є вкрай актуальною у сучасному світі, де конкуренція в сфері краси та побутових послуг постійно зростає. Інформаційна система дозволяє автоматизувати процеси, що раніше виконувалися вручну, сприятиме покращенню обслуговування клієнтів, забезпечуватиме швидкий доступ до необхідної інформації, а також дозволить проводити аналізи та статистичні дослідження для визначення ефективності діяльності салону.

Об'єкт дослідження – веб-додаток салону краси.

Мета роботи - створення ефективної та функціональної інформаційної системи, що допоможе управляти клієнтською базою, забезпечувати точний облік візитів та послуг, зберігати інформацію про користувачів та їхні вподобання.

Методи дослідження - інформаційний аналіз, інформаційне моделювання та комп’ютерний експеримент.

Результати - проведено аналіз літературних джерел. Проаналізовано існуючі інформаційні системи та підходи до їх реалізації. Спроектована база даних та реалізовано веб-додаток у вигляді вебсайту, що дозволяє користувачам зручно взаємодіяти з інформаційною системою салону краси. А

саме, співробітники салону можуть керувати записами та надавати необхідні послуги користувачам.

СТВОРЕННЯ САЙТУ-ВІЗИТКИ, САЛОН КРАСИ, ФРЕЙМВОРК DJANGO, PYTHON, HTML, CSS, BOOTSTRAP, ЗАПИС НА ПРИЙОМИ, МЕДИЧНІ ПОСЛУГИ.

ЗМІСТ

ВСТУП

Розвиток глобальної комп'ютерної мережі Інтернет дуже змінив за останні роки життя людини. Розширилася сфера використання мережі у всіх напрямках соціально-економічного життя суспільства: у побуті, навчанні, економічній діяльності. Комерційне використання Інтернету стало атрибутом ділового життя кожного, хто прагне досягти успіху в бізнесі.

Тому Підприємства використовують можливості Інтернет-технологій для активної інтеграції у електронне бізнес-середовище з власним інформаційним ресурсом. Використання можливостей технічного обміну сьогодні дозволяє легше і швидше рекламувати і продавати продукцію та послуги споживачам, вирішувати задачі фінансово-операційного управління, маркетингового планування, підвищувати конкурентоздатність підприємства.

Ефективне автоматизоване інформаційне і технологічне управління – це ключова ланка в підвищенні якості реалізації електроінструментів. Сайт-візитка – веб-система зі стандартизованим інтерфейсом, своєрідний діалог між підприємством та його оточенням – партнерами чи покупцями продукції. Тому доцільно використовувати Internet для обміну інформацією та оптимізації роботи з покупцями.

Що таке сайт візитка?

Сайт візитка – це, як правило, невеликий сайт, часто складається з 3-5 сторінок. Назва говорить сама за себе – це сайт, призначений в першу чергу для презентації компанії. Він може містити невелику кількість інформації про послуги та продуктах компанії.

Головна сторінка зазвичай присвячена конкретному товару, послузі або заходу. Решта сторінок – це зазвичай «Про компанії», «Відгуки», «Портфоліо», «Контакти». Сьогодні люди дуже плановано підходять до

витрати свого часу. Зазвичай вони хочуть отримати послугу максимально близько до того місця, де знаходяться.

Наприклад, людині терміново знадобилося щось роздрукувати. Звичайно, він може поїхати в якесь відоме йому місце, але це втрата часу. Тому він буде шукати копі-центр там, де знаходиться в даний момент. Погодьтеся, бігати по вулицях в пошуках вивіски – це неефективний метод. Набагато ефективніше пошукати в Інтернеті сайти тих копі-центрів, які розташовані неподалік від вашого місцезнаходження.

Якщо вашого сайту не буде в Інтернеті – людина просто не дізнається про ваше існування. А ось якщо знайде – то саме до вас він і звернеться. І це стосується абсолютно будь-якої послуги, яка може знадобитися людині. На сайті є основна інформація про засновника компанії, про послуги, що надаються, та про переваги перед конкурентами. На видному місці контакти та кнопка зворотного дзвінка. Сайт зроблений у вигляді лендінгу, але при бажанні розділи можна винести на окремі сторінки.

Переваги віртуальної візитки для бізнесу:

- Швидкий запуск. Візитки — одні з найпростіших сайтів, тому їх можна зробити за 2-3 дні роботи.
- Низька вартість. Ви можете створити найпростіший сайт-візитку без залучення розробника, а ще для такого типу сайтів підійде навіть самий бюджетний тариф хостингу.
- Безкоштовні та оперативні правки. Якщо у вас зміниться номер телефону або адреса, вам не доведеться витрачати гроші та час на виправлення інформації, як у випадку з паперовими візитками.
- Додатковий канал для реклами. Ви можете просувати свій бізнес за допомогою реклами сайту в пошукових системах.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Аналіз предметної області

Якщо у вас свій салон краси і ви хочете розвиватися і масштабувати свій бізнес – обов’язково створіть власний сайт. Помилково думати, що досить мати спільноти в соціальних мережах. На них можуть вижити тільки невеликі мікро-студії або домашні майстри. Великим салонам потрібні різні канали, що дозволяють нарощувати базу клієнтів. У даній статті ми розглянемо як визначитися з типом сайту і не витратити гроші даремно.

Навіщо салону краси потрібен власний сайт

На жаль не всі бізнесмени цієї сфери розуміють, навіщо їм сайт. Насправді в сфері послуг, власний веб-ресурс – необхідність №1, як люблять зараз говорити must have. Маючи свій сайт власник може не тільки створити імідж своєму салону, але і має можливість поділитися інформацією про свою компанію, розповісти про акції і про те, які професіонали там працюють.

Помилково думати, що сайт потрібен для того, щоб просто розмістити на ньому прайс. Красивий і грамотний ресурс – зробить вас впізнаваними.

Кожен користувач Мережі має право знати про обраний ним салон все: ціни, послуги, відгуки клієнтів та інше. До слова, якісний ресурс виступає в якості реклами. А якщо проаналізувати сайт, власник компанії зможе зібрати важливі відомості про те, що більш затребуване для клієнтів. Таким чином можна поліпшити свої послуги або додати щось нове.

Розглянемо функції, які виконує сайт салону краси:

- Інформаційна. Це найочевидніша функція сайту. Логічно, що основне завдання – поділитися відомостями про компанію та її послуги.
- Розширення клієнтської бази, залучення нових клієнтів. Що потрібно салону краси? Звичайно ж клієнти. Сайт може стати відмінним

способом залучити нових клієнтів. Використовуючи прості інструменти SEO, які доступні всім, можна побачити, що в пошукових системах Яндекс і Гугл, запит “салон краси” зустрічається 1 336 007 разів, а “перукарня” – 1 960 895 разів. Очевидно, що попит саме на послуги перукаря високий і якщо в вашому салоні її немає, варто задуматися. Та й в принципі, запит на “салон краси” теж немаленький, це свідчить про те, що люди шукають інформацію в Інтернеті, і цим варто скористатися.

- Консультація. Якщо у вашого салону немає сайту, то користувач може подумати, що ви пропонуєте класичний набір послуг: манікюр / педикюр, чоловіча стрижка, фарбування волосся. Маючи свій ресурс, ви можете розповісти про всі послуги. Як же користувачу дізнатися, що ви пропонуєте масаж або послуги косметолога? Через сайт зробити це дуже просто.
- Можливість знайти партнерів по бізнесу і співробітників. Розробник може створити спеціальну сторінку, де при необхідності будуть розміщуватися оголошення про набір персоналу. На власному ресурсі можна буде публікувати інформацію про пропозиції майбутнім партнерам.
- Довіра клієнтів. Якщо у компанії, в нашому випадку у салону краси, є свій сайт, це вже +1 до довіри з боку клієнтів. Власник може розмістити там інформацію про персонал і його кваліфікацію, опублікувати дипломи. Також можна зробити окремий розділ з реальними відгуками. Корисна буде і функція “записатися до майстра онлайн”.

Актуальність роботи

Ідентифікатор людини – його ДНК. Ідентифікатор салону краси – його фірмовий стиль. Завдяки унікальним візуальним атрибутам він формує

зовнішній образ закладу, підкреслює його особливості та переваги в очах клієнтів.

Фірмовий стиль робить салон краси впізнаваним, улюбленим та мотивує до замовлення послуг. Салон краси - це набагато більше, ніж просто заклад з надання послуг. Це справжній храм краси та молодості, відвідування якого допомагає покращити психоемоційний стан та підвищити самооцінку.

Проведені в ньому процедури – це ритуал, який відбувається в максимально приємній, естетично привабливий та комфортній обстановці. Її атрибути повинні викликати довіру, підвищувати лояльність клієнтів та ненав'язливо наголошувати на статусі салону краси. У клієнтів вони мають викликати лише позитивні асоціації та емоції, мотивуючи повторний візит.

Домогтися цього можна за допомогою фірмового стилю – сукупність атрибутів, що підкреслюють усі сильні сторони, специфіку та унікальність закладу. Це ефективний інструмент маркетингу, реклами та залучення клієнтів.

Основні атрибути фірмового стилю салону краси

Фірмовий стиль не повинен нагадувати привабливу обгортку з порожнім змістом. Навпаки - він повинен відображати внутрішній світ салону краси, його можливості, ідеї та місію. Донести їх до потенційних та постійних клієнтів допомагають візуальні атрибути, які ненав'язливо впливають на їхню думку та переконують у правильності вибору.

- Логотип - невід'ємний елемент іміджу, що ідентифікує салон краси і відображає його сутність. Втілюючи собою заклад, він зустрічає клієнта на вході, супроводжує у приміщенні та залишається з ним «назавжди» разом із візиткою.

- Фірмові кольори - оригінальна палітра, що відображає специфіку салону краси, що підкреслює його статус і привабливість. Викликаючи позитивні асоціації, вона є у дизайні інтер'єру, уніформи, візиток та інших атрибутів фірмового стилю.
- Фірмова уніформа – витриманий в єдиному стилі, кольорі, фасони одяг для майстрів та інших співробітників закладу. Вона вказує на їхню приналежність до цього салону краси, підкреслює корпоративний дух і зовнішню дисципліну.
- Фірмовий текстиль - простирадла, рушники, серветки та інший витратний матеріал можна виготовити в єдиному стилі, із застосуванням фірмових кольорів або використовувати для нанесення логотипу, тим самим підкресливши унікальність салону краси.
- Слоган (девіз) - лаконічна фраза, що відображає сутність салону краси або його місію. Вона має бути зрозумілою та надихаючою для клієнтів, співзвучною їх інтересам та світосприйняттю.

Створення сайту на Django – це готовий інструмент для ведення бізнесу. Свій власний онлайн бізнес-проект має бути у кожної компанії, яка дбає не тільки про свій імідж, а й про додаткове залучення потенційних клієнтів. Сьогодні вирішити питання щодо розробки проекту можна за допомогою величезної кількості інструментів. І до одного з найпопулярніших належить Django. Даний фреймворк дозволяє створювати веб-додатки різного ступеня складності, він відноситься до функціонального рішення, за допомогою якого вдається створити онлайн сервіс, що відповідає всім очікуванням замовника.

Головна перевага Джанго – клієнт зобов'язаний дбати лише про логіку майбутнього проекту, решту виконає фреймворк. Якщо є бажання виділитись

серед мільйонів, ідеальним варіантом буде розробка саме на Django. Це дозволить отримати унікальний продукт, який виділяється з величезної кількості аналогічних проектів.

Під час замовлення послуги з використанням Джанго ресурс буде створено з нуля, з урахуванням індивідуальних вимог клієнта. Не потрібний функціонал не буде сповільнювати роботу майданчика, надаючи відвідувачам непотрібну інформацію. Django – це використання банальних шаблонів. Це створений спеціально для потреб клієнта фреймворк

Почнемо розробку сайту на Джанго

Сайт – це не лише гарна картинка, яку бачить відвідувач. Як і в будь-якому бізнесі є своя «кухня». В даному випадку це платформа або «двигун», який дає можливість працювати з ресурсом, наповнювати його необхідною інформацією та змінювати потрібні модулі. Сьогодні багато фахівців використовують системи CMS, які працюють за принципом шаблонів. Замовник вказує необхідний функціонал, а команда програмістів по черзі вбудовує ці «цеглинки» у сайт. Нестача такого варіанта в тому, що є велика ймовірність того, що хтось інтегрує код не зовсім правильно, але так як замовник у даному випадку ще й виконавець - проблем не очікується

Мета роботи:

Створити сайт за допомогою фреймворку Django для покращення досвіду взаємодії клієнта з компанією

1.2 Аналіз ринку і конкурентів

Будь-який бізнес починається з планування та аналізу. Цьому варто приділити час, щоб не вилетіти в трубу. Навіть найпростіший маркетинговий аудит дасть вам такі вигоди:

- Полегшить запуск бізнесу.

- Дозволить заощадити на рекламі.
- Покаже в якому напрямку рухатися.

Щоб побудувати успішний бізнес, вивчіть ринок і проаналізуйте конкурентів. Розглянемо докладніше.

1. Ринок салонів краси. Тут важливо зуміти зібрати інформацію про б'юті-сервіси в конкретному місті в обраному районі. Дайте відповідь на такі питання: Які послуги вже існують? Який діапазон цін? Яких послуг бракує? В ідеалі вивчити не тільки сайти конкурентів, а й прогулятися по району, де ви плануєте відкрити свій салон. Не соромлячись заходьте і цікавтеся послугами конкурентів. Уже після першого дослідження власнику бізнесу буде простіше скласти грамотний бізнес-план, спланувати свій бюджет і зрозуміти, які майстри будуть потрібні його салону. Якщо ви готові ризикнути, то спробуйте відкрити те, чого в принципі в обраному вами районі немає. Наприклад, навколо багато експрес – перукарень, але немає барбершопа. Чому б не почати розвиватися в цій справі? Задумайтесь про більш перспективні ніші в б'юті-індустрії.
2. Аналіз конкурентів. Після того як ви провели аналіз ринку салонів, в принципі список конкурентів уже готовий. Але варто розуміти, що конкурувати можна по-різному. А щоб точніше розуміти, які ніші вже щільно зайняті, а де конкуренція нижче, зверніть увагу на: цінову політику конкурентів, у чому їх унікальна пропозиція, як салони-конкуренти представлені в рекламних каналах.

Одразу відзначимо, що б’юті-індустрія не стоїть на місці. Тут постійно впроваджується щось нове, унікальне. Важливо проводити маркетинговий аудит регулярно, а результати застосовувати в роботі.

Аналіз аналогів

Завданням дипломної роботи є створення сайту салона-краси який орієнтований на записування клієнтів. Беручи це до уваги, було проведено моніторинг різних сайтів такого типу.

Наприклад, сайт «whitelily.kiev.ua» (http://whitelily.kiev.ua/?gclid=CjwKCAjwvsqZBhAlEiwAqAHElWP19d03GGxxiAIweNg7NbMBT077NgBB7LhEVbaly_aATMuBAxNQ3xoCFQwQAvD_BwE) (рис.1.1), сайт «kafo.kiev.ua» (https://kafo.kiev.ua/ua/) (рис. 1.2), сайт «maija.ua» (https://maija.ua/) (рис. 1.3).

Рисунок 1.1 – Інтерфейс сайту «http://whitelily.kiev.ua/»

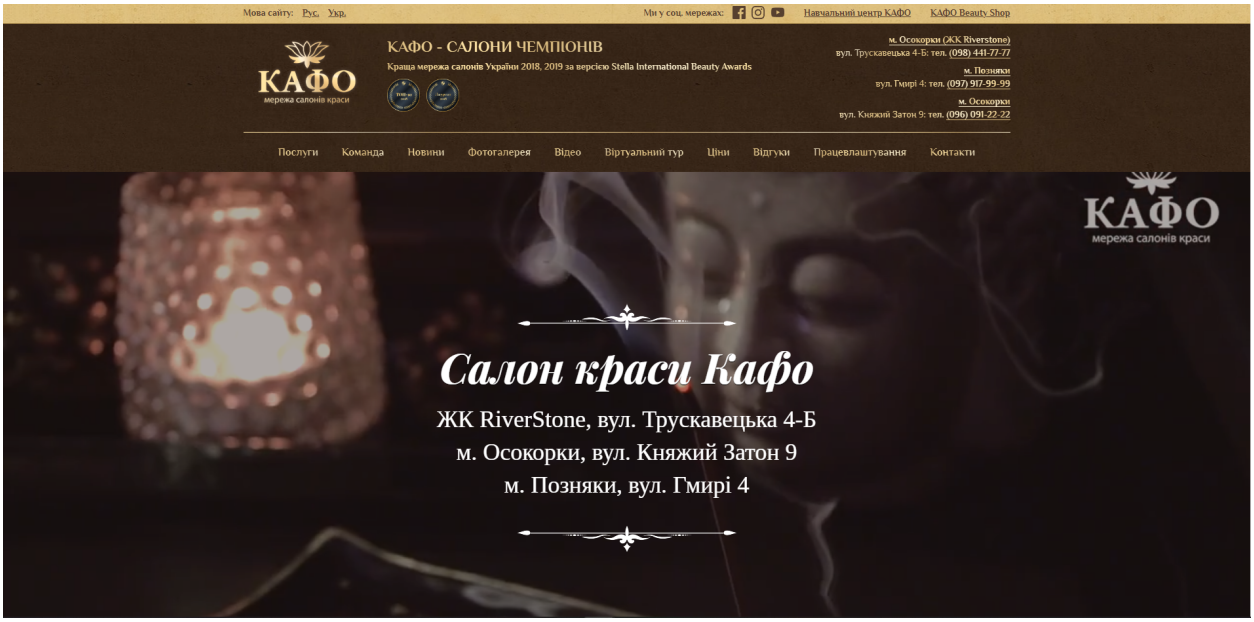


Рисунок 1.2 – Інтерфейс сайту «https://kafo.kiev.ua/ua/»

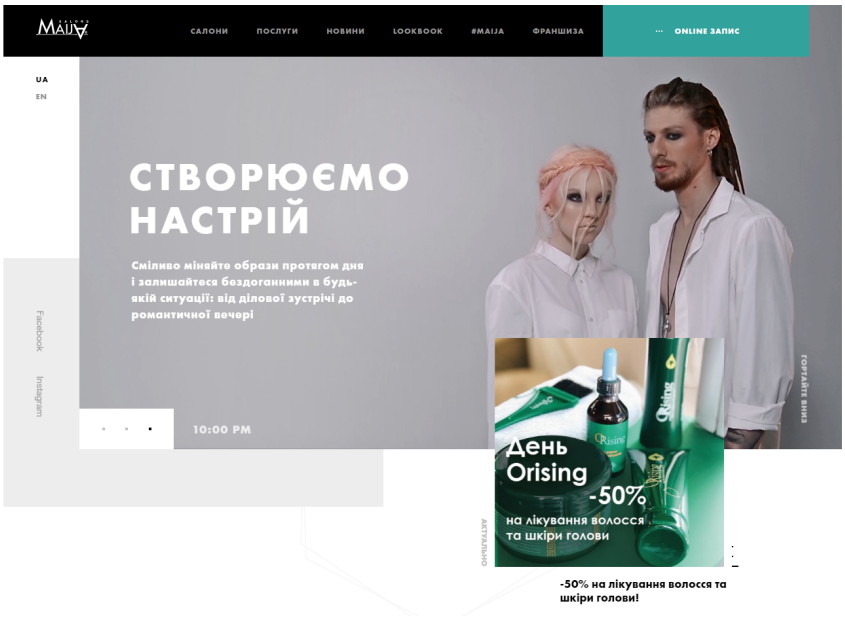


Рисунок 1.3 – Інтерфейс сайту «http://whitelily.kiev.ua/»

Таблиця 1.1 – Порівняльний аналіз аналогів салонів-краси (оцінки по 5-ти бальній шкалі)

Критерії	Сайт «http://whitelily.kiev.ua/»	Сайт «kafo.kiev.ua»	Сайт «maija.ua»
Зручність використання сайту	3	5	2
Дизайн сайту	1	2	5
Інформація про найближчі салони-краси	5	5	5
Коментарі інших людей	0	0	0
Фільтр областей, міст, районів	2	5	4
Наявність особистого кабінету	2	3	5
Пошук процедуру	3	4	5
Детальний опис процедур	4	5	3
Наявність онлайн-консультанта	0	0	0
Наявність історії компанії	5	3	4

Згідно з результатами проведеного аналізу аналогів можна виокремити деякі переваги та недоліки, посилаючись на які буде створено наш сайт

Переваги веб-сайтів аналогів:

- Детальна інформація про компанію та послуги
- Зовнішній вигляд та зручний функціонал сайту

- Можливість перейти за посилання на потрібну інформацію
- Наявність розділу з найпопулярнішими питаннями та відповідями на них
- Можливість пошуку інформації
- Детальний опис процедур
- Авторизація через особистий кабінет

Недоліки веб-сайтів аналогів:

- Відсутність відгуків
- Відсутність онлайн-консультантів

1.3 Постановка задачі

Необхідно реалізувати наступні задачі

- виконати огляд інструментів для розробки та обрати оптимальні;
- розробити інтуїтивно зрозумілий інтерфейс веб-додатка;
- виконати проектування архітектури веб-додатка компанії;
- розробити алгоритм реалізації продукту;
- виконати тестування веб-додатку та зробити відповідні висновки по роботі.

У разі успішної реалізації буде можливим виконати запис на вільний час у перукарні, слідкувати за розширенням франшизи, побудовою нових філіалів, дізнатися ціни послуг та залишити відгук.

2 ВИБІР МЕТОДІВ ВИРІШЕННЯ ЗАДАЧІ

2.1 Вибір мови програмування

У розробці веб-додатків існує багато мов програмування, кожна з яких має свої переваги та недоліки. Ось декілька мов програмування, які часто використовуються для розробки веб-додатків:

Python:

1. Python є однією з найпопулярніших мов програмування для розробки веб-додатків. Вона має простий і зрозумілий синтаксис, що робить її досить легкою у вивченні. Python також має велику кількість бібліотек і фреймворків, які сприяють швидкому розвитку веб-додатків. Django і Flask є двома популярними фреймворками Python для веб-розробки, які забезпечують потужні інструменти для створення високоякісних веб-додатків.

Я обрав Python для розробки веб-додатків з-за його простоти, гнучкості і широкого спектру бібліотек і фреймворків. Python має велику активну спільноту, що означає наявність багатьох ресурсів, документації та підтримки, які допоможуть у вирішенні будь-яких проблем під час розробки. Крім того, Python також ефективно працює з іншими мовами програмування та технологіями, що дає можливість інтегрувати його з існуючими системами.

JavaScript:

2. JavaScript є ще однією популярною мовою програмування для розробки веб-додатків. Вона використовується для взаємодії з користувачем на веб-сторінках і додає динамічність до веб-додатків. JavaScript є основною мовою для фронтенд-розробки, а також використовується на бекенді за допомогою фреймворків, таких як Node.js. JavaScript також

має багато бібліотек і фреймворків, таких як React, Angular і Vue.js, які полегшують розробку веб-додатків.

Вибір мови програмування для розробки веб-додатків залежить від різних факторів, таких як особистий досвід, тип проекту та вимоги клієнта. У моєму випадку, я обрав Python, оскільки я знаю його добре і впевнений у його можливостях для розробки веб-додатків. Крім того, наявність потужних фреймворків, таких як Django, дозволяє швидко створювати складні веб-додатки з мінімальними зусиллями.

Аналіз принципів побудови інтернет-сайту візитки

Перший крок до створення продукту є узгодження теми сайту з замовником, після цього виконавець має зробити план реалізації. Для початку створення плану обираються інструменти для розробки, у нашому випадку це мова Python та фреймворк Django. Після чого складається макет сайту за допомогою якого створюється front-end частина. Наступним кроком буде планування бізнес-логіки та її реалізація разом з усім back-endom. Також дуже важлива складова якісно зробленого продукту це написання тестів.

2.2 Вибір фреймворку

Flask:

Flask - це легкий мікро фреймворк для розробки веб-додатків на Python. Він має мінімальний набір інструментів і не накладає жорстких правил щодо організації проекту. Flask надає основні можливості для створення веб-додатків, але при цьому залишає велику вільність у виборі компонентів та архітектури додатку. Він є простим у вивченні і використанні, зокрема для початківців.

Django:

Django - це повноцінний фреймворк для розробки веб-додатків на Python. Він пропонує повний набір інструментів і компонентів, що спрощує процес розробки. Django має вбудовану адміністративну панель, шаблонний двигун, ORM (об'єктно-реляційний мапер), систему маршрутизації і багато інших корисних функцій. Він пропонує чітку структуру проекту і дотримання найкращих практик розробки веб-додатків. Django підходить для розробки великих і складних проектів, а його висока продуктивність і масштабованість роблять його популярним серед професіоналів.

Причини, чому я обрав Django для розробки веб-додатків, засновані на моєму досвіді та його перевагах:

1. Комплексність і повнота: Django надає все необхідне для розробки веб-додатків, включаючи адміністративну панель, авторизацію, маршрутизацію, ORM і багато іншого. Це дозволяє ефективно працювати над проектом, зосереджуючись на його функціональності, замість того, щоб будувати всі компоненти з нуля.
2. Структурованість: Django пропонує чітку структуру проекту, що сприяє організації коду і полегшує спільну роботу в команді. Це допомагає зберігати код організованим і легким для розуміння, а також спрощує масштабування та підтримку проекту у майбутньому.
3. Багата екосистема: Django має широку спільноту розробників, що призводить до наявності великої кількості сторонніх бібліотек, модулів і розширень. Це дозволяє швидко реалізувати різноманітні функціональність і використовувати готові рішення для типових задач.
4. Продуктивність і масштабованість: Django побудований з орієнтацією на продуктивність і масштабованість. Він може ефективно обробляти великі обсяги даних і високе навантаження. Крім того, Django має

вбудовану підтримку кешування, оптимізацію запитів і інші інструменти для підвищення продуктивності додатків.

5. Моє досвід і знайомство: Я маю попередній досвід роботи з Django і добре знайомий з його концепціями та практиками. Це дозволяє мені більш ефективно працювати над проектами на Django, зменшує час на розвиток та забезпечує якісний результат.

Загалом, Django є потужним і надійним фреймворком для розробки веб-додатків, який допомагає прискорити процес розробки і забезпечити високу якість проекту. Його повнота, структурованість та масштабованість роблять його ідеальним вибором для серйозних та складних проектів.

2.3 Вибір СУБД

СУБД (Системи управління базами даних) — це програмні засоби, які дозволяють зберігати, організовувати та керувати базами даних. Ось кілька стислих описів популярних СУБД:

1. SQLite: Це легковага вбудована СУБД, яка зберігає бази даних у звичайних файлових структурах. Вона не вимагає окремого сервера і проста у використанні. SQLite підтримує багато мов програмування і використовується широко в мобільних додатках та простих веб-проектах.
2. MySQL: Це відкрита реляційна СУБД, яка надає широкі можливості для роботи з базами даних. MySQL є потужним та масштабованим рішенням, яке використовується у великих веб-проектах та системах з великим обсягом даних.
3. PostgreSQL: Це реляційна СУБД з високою надійністю, підтримкою транзакцій та розширеними можливостями. PostgreSQL підтримує

складні запити, географічні дані, повнотекстовий пошук та багато інших функцій. Він є популярним вибором для веб-додатків з великим обсягом даних та потребою у розширених можливостях.

У Django, який є популярним фреймворком веб-розробки, можна використовувати різні СУБД для зберігання даних. Однак, SQLite є частим вибором для Django проектів, особливо на етапі розробки та в невеликих проектах. Ось декілька обґрунтувань, чому використовувати SQLite у Django проектах:

1. Легкість використання: SQLite не потребує окремого сервера баз даних, що робить його легким у встановленні та налаштуванні. Це особливо зручно на етапі розробки, коли не потрібно використовувати складну інфраструктуру баз даних.
2. Портативність: База даних SQLite представлена у вигляді одного файлу, що робить її легко переносною. Вона може бути включена у веб-додаток та зберігатись разом з ним, що спрощує розгортання проекту на різних серверах.
3. Швидкодія: SQLite працює швидко для багатьох типів завдань, особливо для невеликих проектів з обмеженим обсягом даних. Вона має низький рівень накладних витрат та високу продуктивність при невеликому навантаженні.
4. Сумісність з Django: Django має вбудовану підтримку SQLite та простий інтерфейс для роботи з ним. Він надає ORM (об'єктно-реляційне відображення), що дозволяє зручно працювати з базою даних, незалежно від використовуваної СУБД.

Враховуючи ці фактори, SQLite може бути відмінним вибором для Django проектів, зокрема на початкових етапах розробки та в невеликих

проектах, де простота, легкість використання та переносність мають вагому роль.

Веб програмування — галузь веб розробки і різновид дизайну, в завдання якої входить проектування користувальницьких веб інтерфейсів для сайтів та веб додатків. Веб Дизайнери проектують логічну структуру веб сторінок, продумують найбільш зручні рішення подачі інформації, а також займаються художнім оформленням веб проекту. В результаті перетину двох галузей людської діяльності грамотний веб дизайнер повинен бути знайомий з останніми веб технологіями і володіти відповідними художніми якостями.

Теоретично гіпертекст — це всього лише зручний спосіб представлення інформації. Але на практиці гіпертекст — це можливість зробити посилання на інші документи за допомогою слів, фраз, малюнків. Ім'я кожного з цих місць можна зв'язати з іншим документом, у якому міститься більш докладна інформація. Коли користувач вибирає посилання в першому документі, браузер відкриває другий документ із більш докладними даними.

2.4 Особливості

1. Інформація ніяк не впорядковується — документи просто зв'язуються один з одним за допомогою посилань. Хоча головною метою багатьох методів є саме впорядкування інформації тим або іншим способом (наприклад, у виді ієрархії), у гіпертексті основна увага приділяється створенню інформаційних зв'язків. Таким чином, гіпертекст — це спроба створення моделі, що описує спосіб представлення інформації в мозку людини.

2. Інформаційні зв'язки можуть існувати між самими різними документами. Створюючи впорядкований список або схему, ви помічаєте на кожне місце в списку або ієрархії (тобто в структурі) тільки один елемент. А в

гіпертексті кожен інформаційний фрагмент (або елемент) може знаходитися в багатьох, причому зовсім різних, місцях структури.

Термін гіпермедіа використовується для опису того, що ви знаходите в Web. Гіпермедіа — це природне узагальнення поняття гіпертексту, що відноситься до документів, у яких розміщується не тільки текст, але і мультимедіа, тобто зображення, відеозаписи і звук. Ці елементи також можна зв'язувати з іншими документами гіпермедіа. Наприклад, на Web-сторінці можна зв'язати зображення з документом таким чином, що якщо користувач клацне на зображенні, браузер відкриває відповідний документ.

Враховуючи зазначені вище задачі, було обрано такі мови програмування – HTML (використовуючи технологію «BEM»), CSS (використовуючи препроцесор «SCSS»), JavaScript (Використовуючи бібліотеку «jQuery»), PHP, MySQL.

HTML (Hypertext Markup Language — мова гіпертекстової розмітки) служить для опису Web-сторінки, що зберігається у виді звичайного текстового файлу з розширенням *.htm або *.html. Головна мета HTML — описати формат вмісту Web-сторінки, він описується з допомогою дескрипторів (tag) HTML.

Дескриптори визначають способи форматування тексту, служать розпізнавальними знаками зображень або таблиць, дозволяють зв'язувати слова або фрази з іншими документами в Internet.

Якщо дати коротке визначення Web-сторінки, то це комбінація тексту і дескрипторів HTML, що описують способи форматування цього тексту.

Мова HTML є лінійною мовою й у стандартному варіанті не підтримує циклів і розгалужень. Браузер передивляється документ від початку і до кінця, одразу форматуючи сторінку.

Чистий HTML— це мова, що описує, як повинен виглядати документ. Його також можна назвати мовою розмітки.

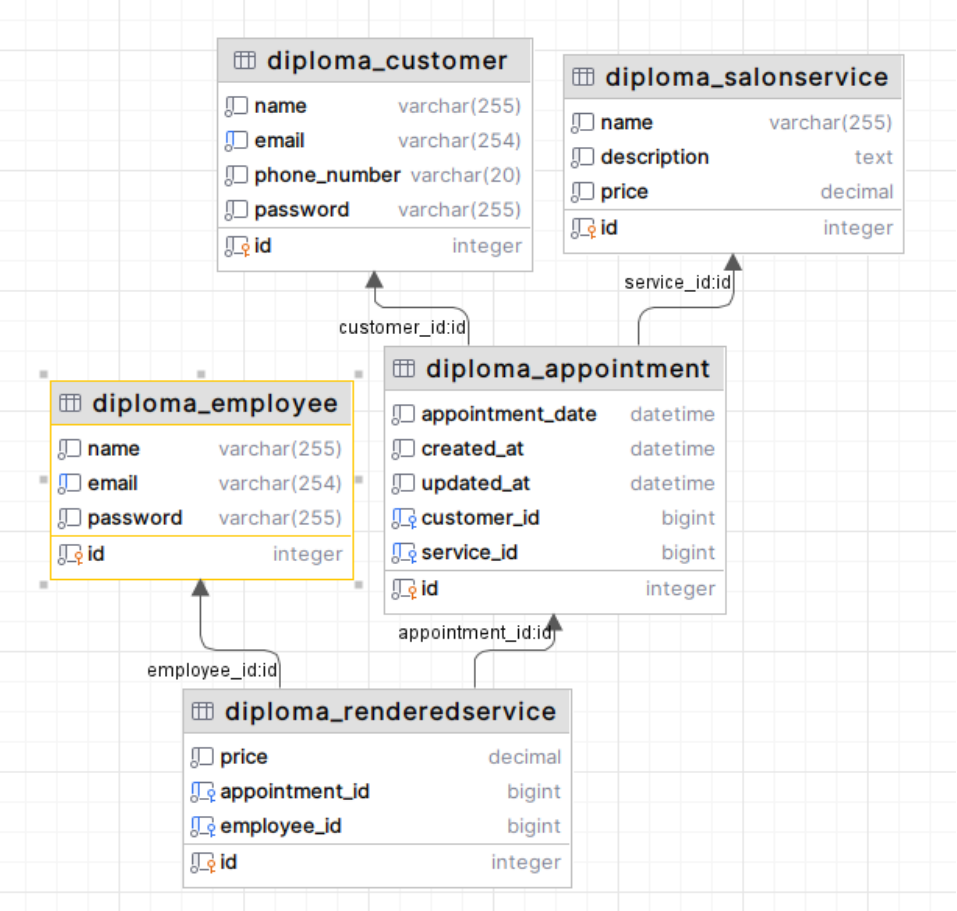
Основним елементом HTML є дескриптори або теги. Документ форматується при додаванні дескриптора, що точно вказує, як повинен виглядати текст. Дескриптори HTML розташовані у кутових дужках . Умовно дескриптори можна розбити на три частини:

• Дескриптори, які інформують браузер про те, що документ є HTML-документом, і дескриптори коментарів.

• Дескриптори заголовків HTML-документа. • Дескриптори тіла HTML-документа.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Проектування бази даних



3.1 - ERD

Основні сутності в моїй системі включають User, Employee, Customer, SalonService, Appointment та RenderedService. Нижче наведено короткий опис кожної сутності:

1. User: Це базова модель, яка використовується для представлення користувачів системи. Має загальну інформацію про користувачів, таку як електронна пошта, ім'я та номер телефону. Має два підкласи - Employee та Customer.

2. Employee: Представляє співробітників салону краси. Наслідується від User. Включає поле role, яке вказує, що користувач є співробітником.
3. Customer: Представляє клієнтів салону краси. Наслідується від User.
4. SalonService: Модель, що описує послуги, які надаються салоном краси. Містить інформацію про назву послуги, її опис та ціну.
5. Appointment: Модель, що представляє записи на послуги. Містить зв'язки з користувачем (customer) та послугою (service). Також містить інформацію про дату та час запису.
6. RenderedService: Модель, що представляє послуги, які були надані у рамках запису (Appointment). Містить зв'язки з Appointment та Employee.

ER-діаграма наочно показує ці зв'язки та структуру даних. Вона допомагає вам краще розуміти, як дані взаємодіють між собою.

3.2 Огляд інтерфейсу сайту

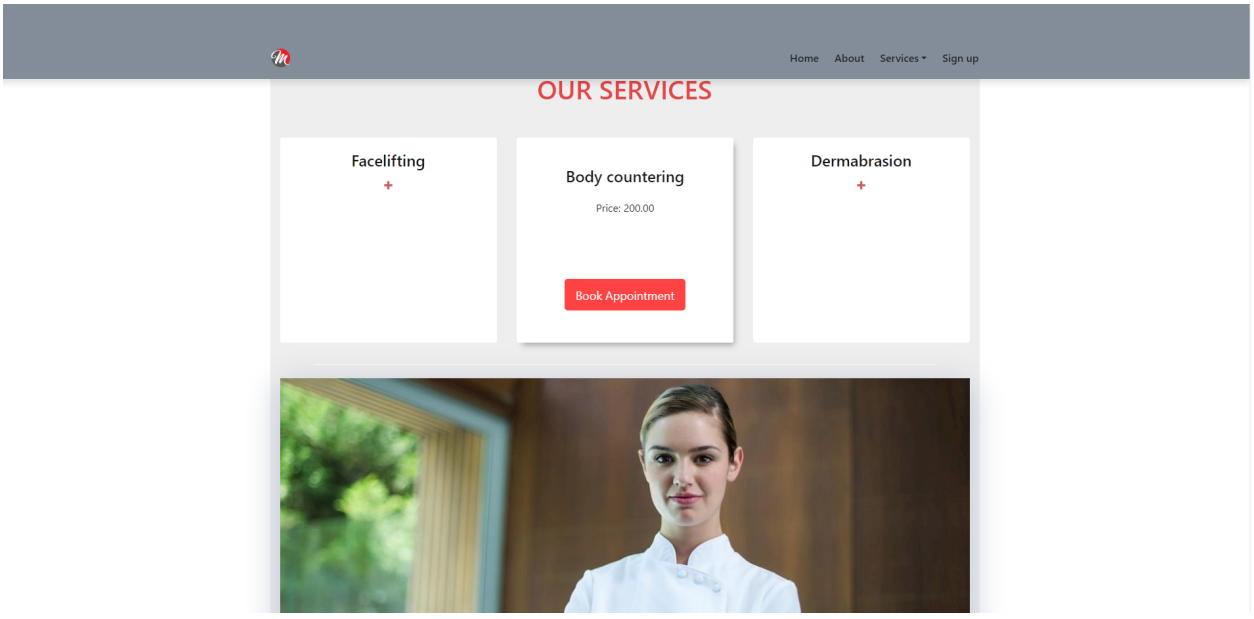


Рисунок 3.2 - Головна сторінка, кожна послуга містить короткий опис на передній частині картки та кнопку для запису на іншій стороні

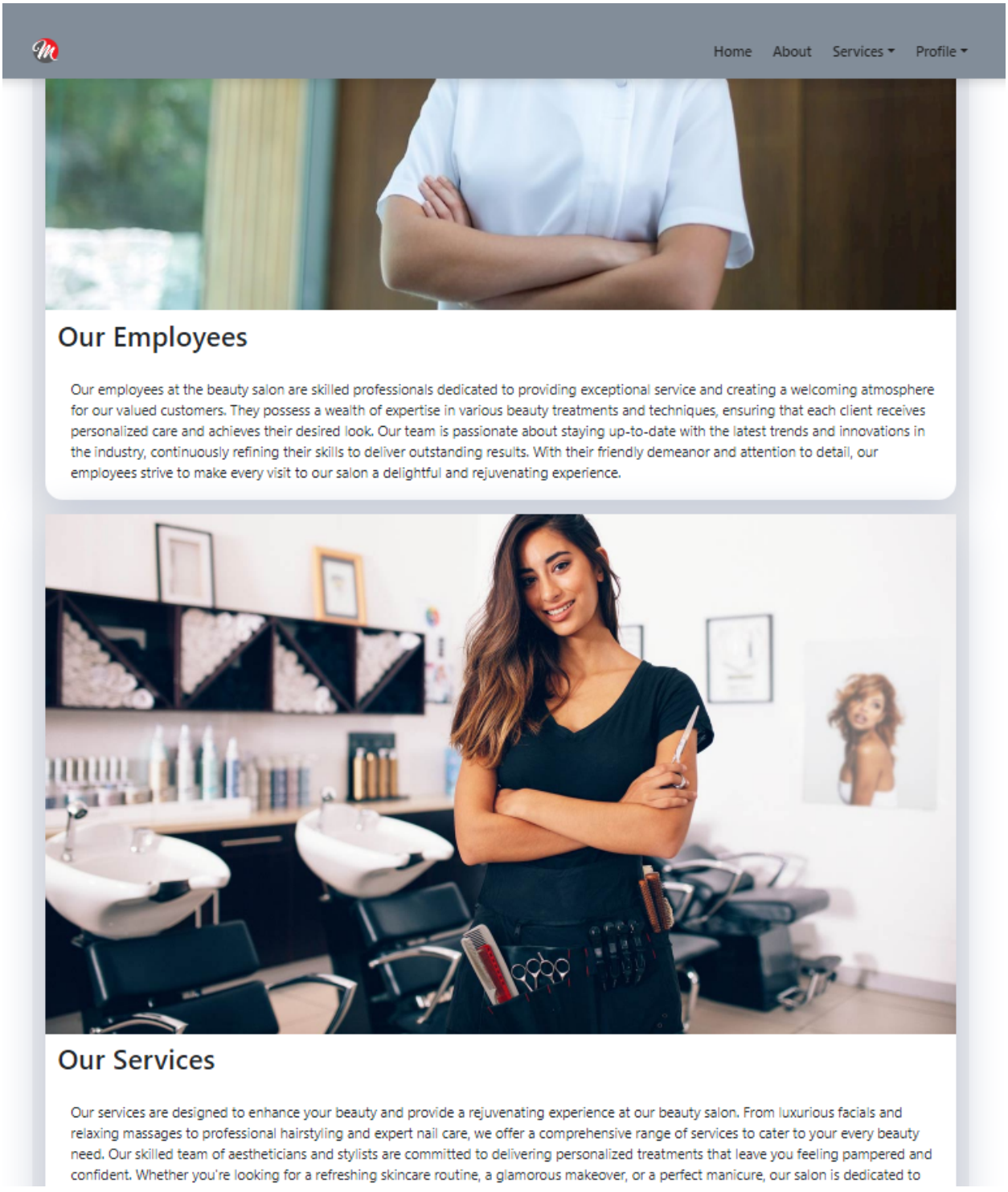


Рисунок 3.3 - Продовження головної сторінки, інформація про послуги та робітників салону

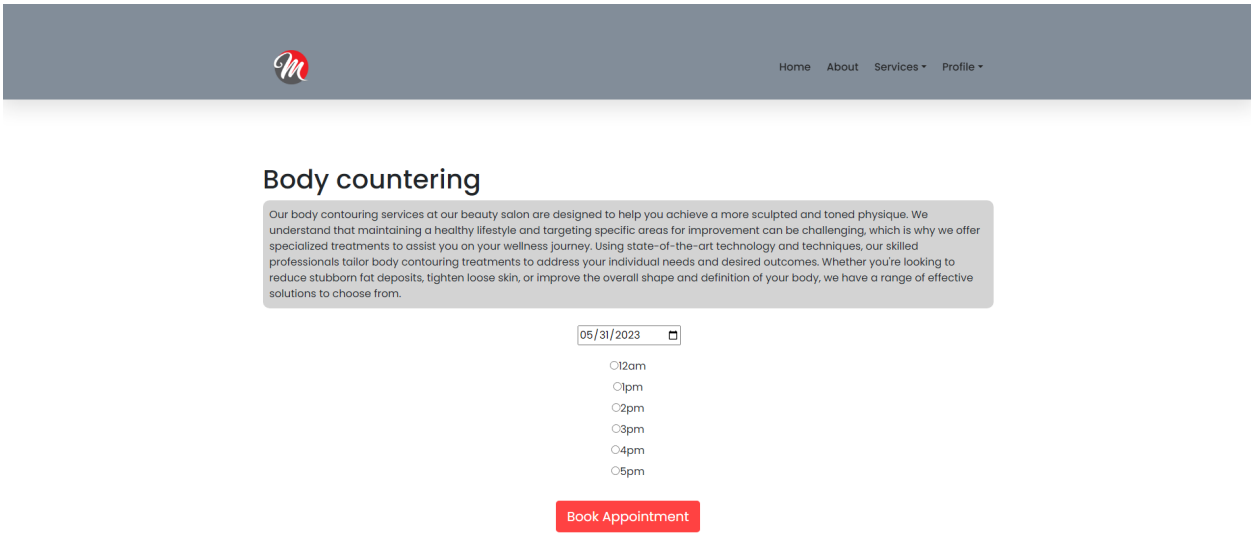


Рисунок 3.4 - Сторінка запису на послугу, користувач може вибрати дату(будь яку але не більшу за сьогодні) І йому буде показано декілька опцій щодо часу запису

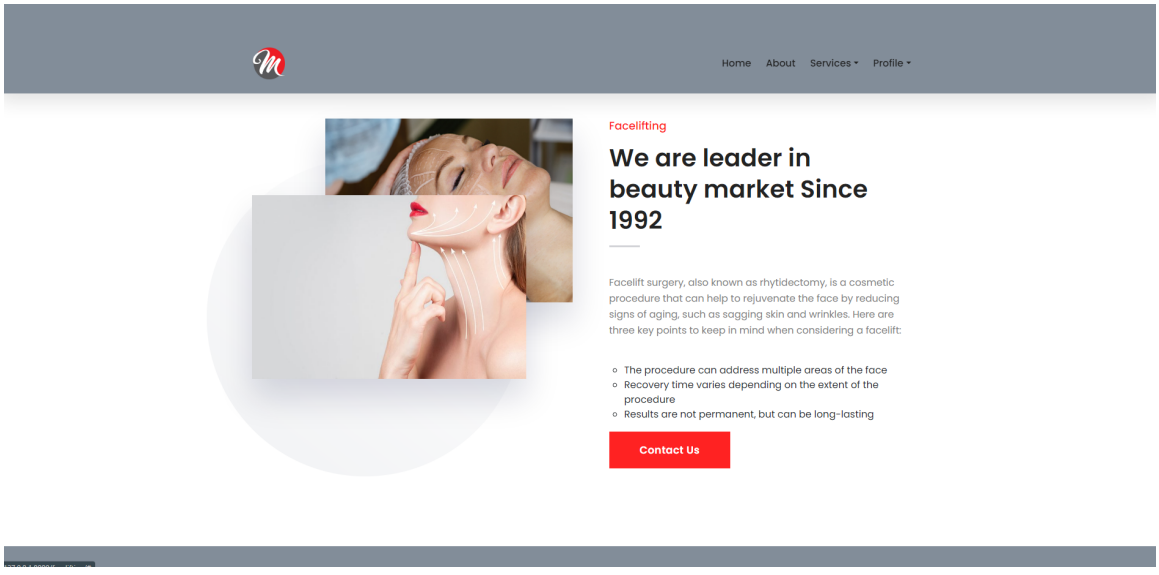


Рисунок 3.5 - Сторінка про компанію

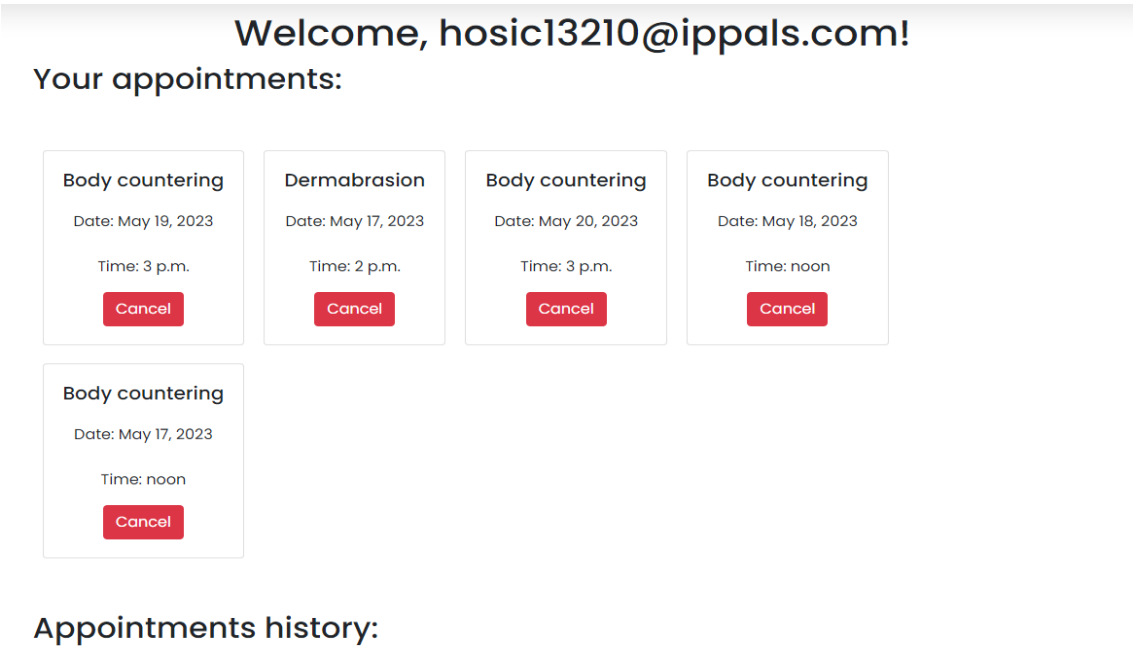


Рисунок 3.6 - Кабінет користувача,подивитись минулі записи або скасувати майбутні

3.3 Реалізація особистого кабінету

Для виконання функціонала особистого кабінету я створив функцію profile:

```
    @login_required
    def profile(request):
        user: User = request.user
        current_date = datetime.now().date()
        if user.is_employee():
            upcoming_appointments =
Appointment.objects.filter(appointment_date__gte=current_date)
            booked_times_in_appointments =
RenderedService.objects.filter(employee_id=user.id)
            past_appointments =
RenderedService.objects.filter(employee_id=user.id)
```

```
            available_appointments = upcoming_appointments.exclude(
id__in=booked_times_in_appointments.values_list('appointment_id__id',
flat=True)
            ).exclude(
appointment_date__in=past_appointments.values_list('appointment_id__a
ppointment_date', flat=True),
appointment_time__in=past_appointments.values_list('appointment_id__a
ppointment_time', flat=True)
            )

            context = {
                'appointments': available_appointments,
                'past_appointments': past_appointments,

            }
            if request.method == 'POST':
                RenderedService.objects.create(
                    employee_id=Employee.objects.get(pk=user.id),
appointment_id=Appointment.objects.get(pk=request.POST.get('appointme
nt_id'))
                )
                messages.success(request, 'Accepted appointment
successfully.')

            return render(request, 'members/employee_profile.html',
context)
        else:
            user = request.user
            upcoming_appointments =
Appointment.objects.filter(customer=user,
appointment_date__gte=current_date)
```

```
            past_appointments =
Appointment.objects.filter(customer=user,
appointment_date__lt=current_date)
            if request.method == 'POST':
                appointment_id = request.POST.get('appointment_id')
                if appointment_id:
                    appointment = get_object_or_404(Appointment,
pk=appointment_id)
                    appointment.delete()
                    messages.success(request, 'Appointment cancelled
successfully.')

            context = {
                'user': user,
                'appointments': upcoming_appointments,
                'past_appointments': past_appointments,
            }

            return render(request, 'members/profile.html', context)
```

Ця функція залежно від ролі користувача (**співробітник або клієнт**) відображає профільний шаблон для відповідного типу користувача. Давайте розглянемо її роботу та вимоги до об'єктів моделі.

1. Авторизація користувача: Декоратор @login_required перевіряє, чи користувач авторизований перед виконанням функції. Якщо користувач не авторизований, він буде перенаправлений на сторінку входу.
2. Визначення типу користувача: За допомогою request.user отримується об'єкт користувача, який виконав запит. Перевірка user.is_employee() визначає, чи є користувач співробітником.
3. Обробка для співробітника: Якщо користувач є співробітником, функція виконує наступні дії:

- Отримує майбутні записи на прийоми (upcoming_appointments), в яких дата прийому більше або дорівнює поточній даті.
- Отримує список заброньованих часів (booked_times_in_appointments) співробітника.
- Отримує минулі записи на прийоми (past_appointments), в яких дата прийому менше поточної дати.
- Фільтрує майбутні записи на прийоми, щоб відобразити доступні часи прийому (available_appointments). Це робиться шляхом виключення заброньованих часів та часів з минулих записів.
- Якщо запит відправлено методом POST (наприклад, форма прийому запису), створюється об'єкт RenderedService з вибраним співробітником і прийомом.
- Повертає відповідний шаблон 'members/employee_profile.html' разом з контекстом, який містить доступні записи на прийоми (appointments) та минулі записи (past_appointments).

4. Обробка для клієнта: Якщо користувач не є співробітником, функція виконує дії для клієнта:
    - Отримує майбутні записи на прийоми (upcoming_appointments), в яких користувач є клієнтом та дата прийому більше або дорівнює поточній даті.
    - Отримує минулі записи на прийоми (past_appointments), в яких користувач є клієнтом та дата прийому менше поточної дати.
    - Якщо запит відправлено методом POST (наприклад, форма відміни запису), видаляє відповідний об'єкт Appointment.
    - Повертає відповідний шаблон 'members/profile.html' разом з контекстом, який містить дані користувача (user), майбутні записи (appointments) та минулі записи (past_appointments).

Вимоги до об'єктів моделі:

- User: Об'єкт користувача з розширеними полями, такими як електронна пошта, ім'я, номер телефону, роль (співробітник або клієнт).
- Employee: Наслідується від User і представляє співробітника салону краси.
- Customer: Наслідується від User і представляє клієнта салону краси.
- SalonService: Модель, що описує послуги салону краси.
- Appointment: Модель, що представляє записи на прийоми. Містить зв'язки з користувачем (customer) та послугою (service).
- RenderedService: Модель, що представляє послуги, надані у рамках запису на прийом. Містить зв'язки з Appointment та Employee.

Ця функція відображає сторінку профілю для користувачів, залежно від їхньої ролі (співробітник або клієнт) та виконує відповідні дії, такі як відображення майбутніх та минулих записів на прийоми, створення нових записів на прийом, відміна записів тощо.

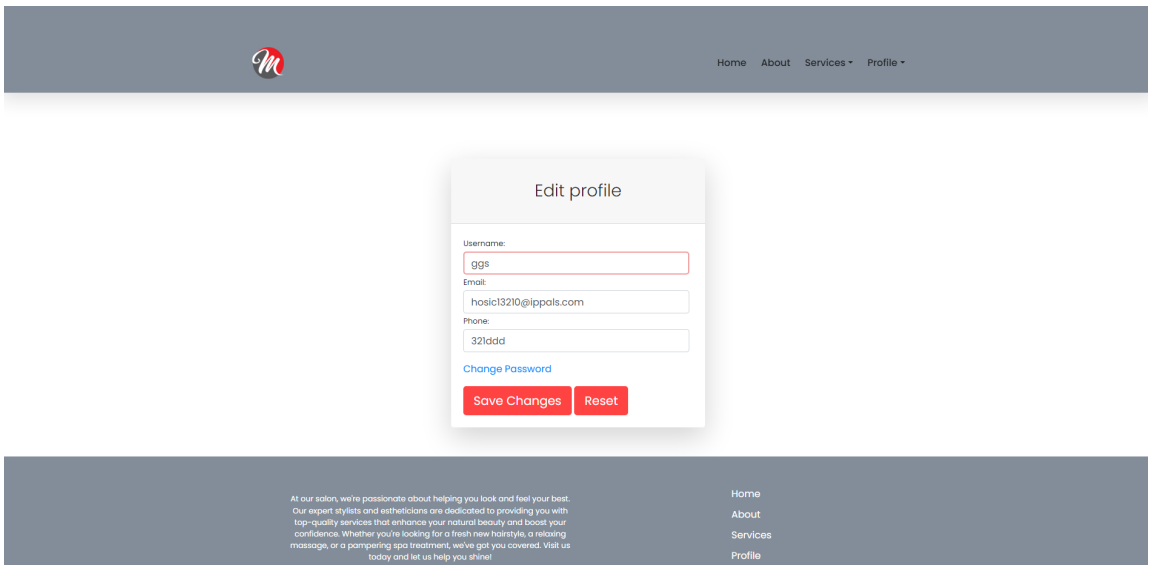


Рисунок 3.7 - Зміна даних користувача

Коли користувач намагається змінити дані свого профілю, поля його моделі вже заповнені даними, це було можливо виконати завдяки кастомній формі

Та функції:

```
    @login_required
    def edit_profile(request):
        if request.method == 'POST':
            user_form = UpdateUserForm(request.POST,
instance=request.user)
            if user_form.is_valid():
                user_form.save()
                messages.success(request, 'Your profile is updated
successfully')
                return redirect(to='profile')
        else:
            user_form = UpdateUserForm(instance=request.user)

        return render(request, 'members/edit_profile.html',
{'user_form': user_form})
```

Ця функція відображає сторінку редагування профілю користувача. Давайте розглянемо її роботу.

1. Авторизація користувача: Декоратор @login_required перевіряє, чи користувач авторизований перед виконанням функції. Якщо користувач не авторизований, він буде перенаправлений на сторінку входу.
2. Обробка запиту: Функція перевіряє, чи запит був відправлений методом POST. Якщо так, це означає, що користувач збирається оновити свій профіль.

3. Формування форми: Створюється екземпляр форми UpdateUserForm, яка приймає дані з POST-запиту та використовує поточного користувача як інстанцію (instance) для заповнення полів форми.
4. Перевірка валідності форми: Перевіряється, чи форма є валідною за допомогою методу is_valid(). Якщо форма є валідною, зберігаються зміни в профілі користувача, повідомлення про успішне оновлення профілю відображаються за допомогою messages.success(), і користувач перенаправляється на сторінку профілю.
5. Відображення форми: Якщо запит не є методом POST, це означає, що користувач просто відкриває сторінку редагування профілю. У цьому випадку відображається шаблон 'members/edit_profile.html', який містить форму user_form для редагування профілю користувача.

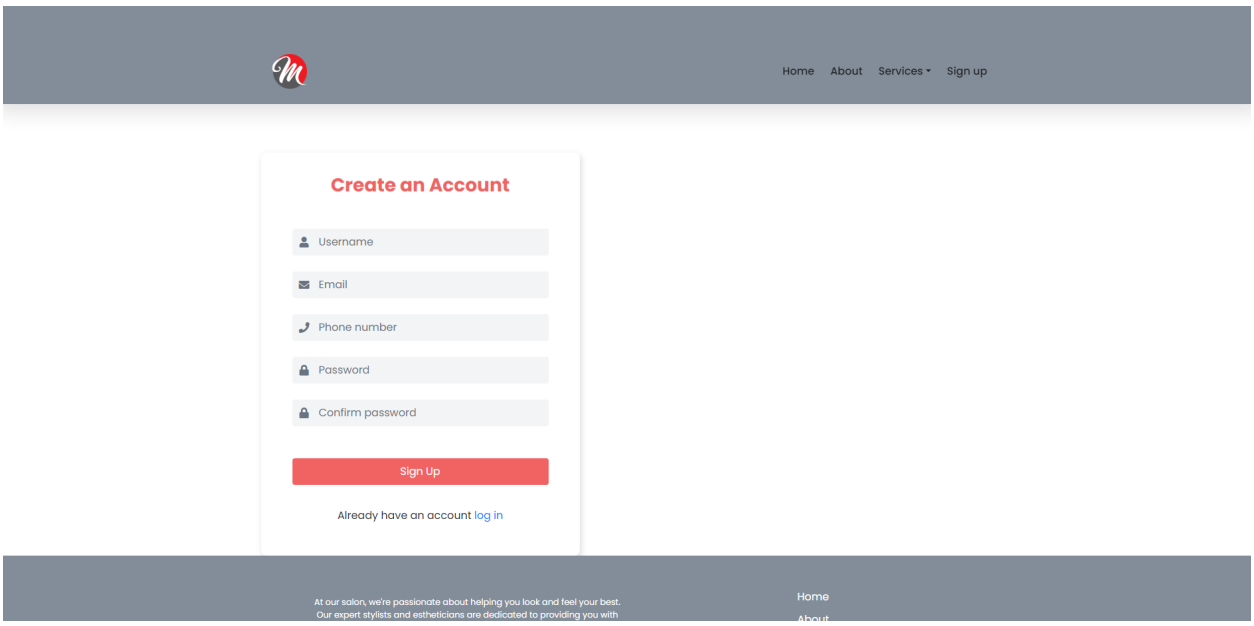


Рисунок 3.8 - Реєстрація користувача

Ця функція відображає сторінку реєстрації для нових користувачів. Давайте розглянемо її роботу.

1. Обробка запиту: Функція перевіряє, чи запит був відправлений методом POST. Якщо так, це означає, що користувач намагається зареєструватися.
2. Формування форми: Створюється екземпляр форми CustomerCreationForm, яка приймає дані з POST-запиту.
3. Перевірка валідності форми: Перевіряється, чи форма є валідною за допомогою методу is_valid(). Якщо форма є валідною, новий користувач зберігається у системі за допомогою методу save(), який створює новий об'єкт користувача на основі наданих даних форми.
4. Аутентифікація та вхід в систему: Після успішної реєстрації користувача, викликається функція authenticate() для перевірки введених даних, таких як електронна пошта та пароль, і отримання об'єкта користувача. Якщо аутентифікація успішна, користувач автоматично увійде в систему за допомогою функції login(), після чого відбудеться перенаправлення на домашню сторінку (home).
5. Відображення форми: Якщо запит не є методом POST, це означає, що користувач просто відкриває сторінку реєстрації. У цьому випадку відображається шаблон 'members/signup.html', який містить форму form для введення даних реєстрації користувача.

Чому саме обрано базовий спосіб реєстрації для користувачів:

- Базовий спосіб реєстрації для користувачів був обраний, оскільки функція використовує стандартні форми Django, такі як CustomerCreationForm, для реєстрації нових користувачів. Це забезпечує швидкий старт і простоту реєстраційного процесу без необхідності додаткової настройки та розробки власних форм.
- Використання базового способу реєстрації також дозволяє використовувати вбудовану систему аутентифікації та авторизації

Django, яка включає в себе функції, такі як перевірка пароля, аутентифікація користувача та зберігання сесій.

- Базовий спосіб реєстрації також може бути легко розширений та налаштований згідно з потребами проекту, додавши додаткові поля до форми реєстрації або використовуючи власний підклас форми.

У цілому, базовий спосіб реєстрації забезпечує простоту використання та інтеграцію з існуючими компонентами Django для обробки аутентифікації та реєстрації користувачів.

3.4 AJAX

Також окрему увагу я хотів би приділити системі що надає змогу динамічно отримувати інформації щодо записів(AJAX).

Цей код включає дві функції та відповідний HTML-шаблон, які дозволяють динамічно показувати доступні часи для запису за допомогою Ajax. Давайте розглянемо його детальніше:

1. Функція get_available_times(request): Ця функція є представленням Django, яке обробляє Ajax-запит і повертає доступні часи для запису.
    - Параметри запиту, такі як вибрана дата appointment_date та ідентифікатор послуги service_id, передаються через POST-запит.
    - Функція виконує запит до бази даних, щоб отримати наявні записи existing_appointments на обрану дату та послугу.
    - Далі вона отримує поточного користувача і шукає його записи times_of_current_user на обрану дату.
    - За допомогою цих інформаційних параметрів вона формує список доступних часів available_times, перевіряючи, які часи ще не зайняті.

- На вихіді функція повертає відповідь JSON зі списком доступних часів.

2. HTML-шаблон my_site/service_detail.html: Цей шаблон містить форму для бронювання послуги та скрипт JavaScript, який взаємодіє з сервером за допомогою Ajax для отримання доступних часів запису на основі вибраної дати.
    - Форма має поле appointment_date, яке дозволяє користувачеві вибрати дату для запису.
    - Є також контейнер available-times, який буде оновлюватися з допомогою Ajax для відображення доступних часів на обрану дату.
    - Скрипт JavaScript виконує наступні дії:
        - Він отримує значення обраної дати та ідентифікатор послуги при зміні поля appointment_date.
        - Потім він створює Ajax-запит до функції get_available_times зі зазначеними параметрами.
        - Після успішного отримання відповіді від сервера, він оновлює контейнер available-times, створюючи HTML-код для кожного доступного часу.
        - Крім того, скрипт перевіряє, чи був обраний час перед надсиланням форми, і виводить повідомлення, якщо час не був обраний.
3. Функція service_booking(request, pk): Ця функція обробляє сторінку бронювання послуги.
    - Вона отримує об'єкт SalonService за допомогою ідентифікатора pk.

- Якщо метод запиту є POST, вона перевіряє, чи форма date_form (яка містить дату запису) є дійсною.
- Якщо форма дійсна, створюється новий об'єкт Appointment з введеними даними та повідомленням про успішне бронювання.
- Якщо форма недійсна, повертається повідомлення про невдале бронювання разом із формою.
- Якщо метод запиту не є POST, вона створює форму AppointmentForm та рендерить сторінку з відповідними даними.
- Він також містить JavaScript-код, який взаємодіє з сервером за допомогою Ajax, як описано вище, для отримання доступних часів.

Основна мета цього коду полягає в тому, щоб дозволити користувачам динамічно вибирати дату та час запису на основі наявних записів і зручно виконувати бронювання послуги.

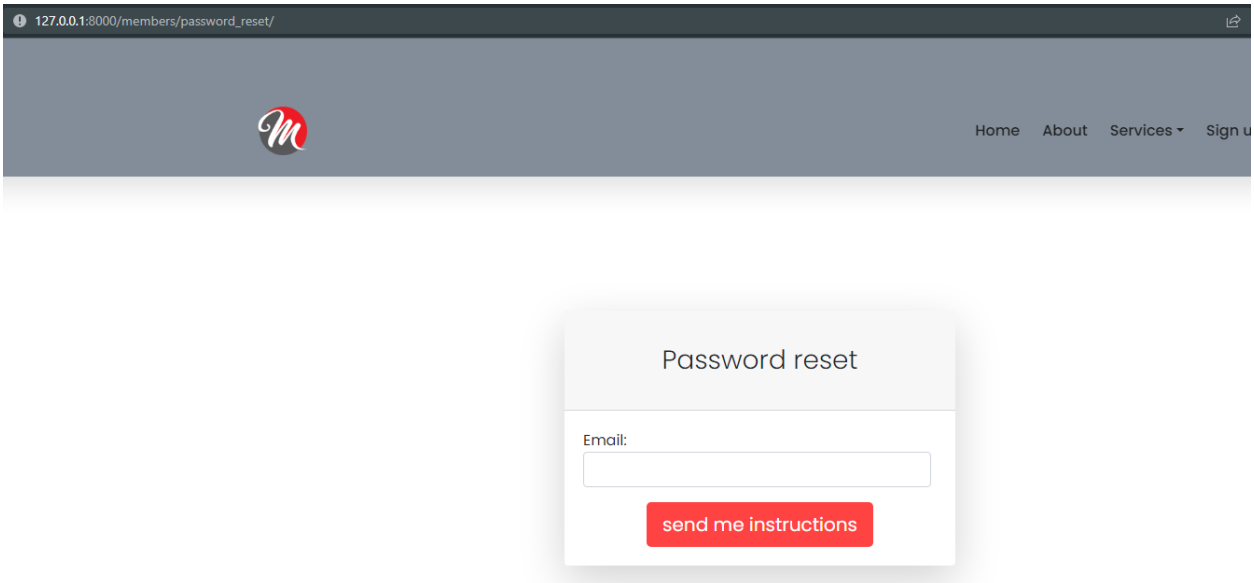


Рисунок 3.9 - Відновлення паролю

Важливим аспектом мого проекту було реалізація функціоналу відновлення паролю за допомогою django.contrib.auth.urls. Це вбудований модуль Django, який надає готові URL-шаблони для автентифікації та керування обліковими записами користувачів. За допомогою django.contrib.auth.urls я зміг легко включити сторінки для відновлення паролю у своєму проекті. Це включає сторінки для введення електронної пошти, підтвердження по електронній пошті та зміни паролю. Всі ці сторінки мають відповідні URL-шаблони, які автоматично обробляються фреймворком Django. Це значно спростило процес реалізації функціоналу відновлення паролю. Я не потребував додаткового коду для обробки форм, перевірки введених даних або відправки електронних листів з посиланнями на підтвердження. Все це вже було вбудовано в django.contrib.auth.urls.

Загалом, використання django.contrib.auth.urls було ефективним та зручним способом реалізації функціоналу відновлення паролю у моєму проекті. Це дозволило зосередитися на інших аспектах розробки та забезпечило безпечну та зручну процедуру відновлення паролю для користувачів мого сайту.

ВИСНОВКИ

Процес створення сайту візитки для салону краси був надзвичайно корисним для мене. Під час розробки проекту я вдало покращив свої навички роботи з HTML та CSS. Вивчення цих мов дозволило мені створити стильний та привабливий дизайн, який належним чином відображає концепцію салону краси.Крім того, я освоїв базові навички використання AJAX для динамічної оновлення контенту на сторінці. Використання AJAX дозволило мені реалізувати функціональність, таку як динамічне завантаження доступних часів для запису, без перезавантаження сторінки. Це значно покращило користувальницький досвід та зробило проект більш інтерактивним.Одним з найбільших досягнень в цьому проекті було освоєння Django, яка допомогла мені реалізувати багатофункціональну бекенд частину сайту. За допомогою Django, я створив моделі бази даних, виконав запити до бази даних, обробив форми, аутентифікував користувачів та реалізував безпеку.Процес проєктування бази даних був також важливою частиною проекту. Я використовував свої знання для створення зв'язків між сутностями, зберігання та організації даних для ефективної роботи з ними.

Загалом, цей проект дозволив мені розвинути мої навички роботи з HTML, CSS, AJAX та Django, Python. Я отримав практичний досвід розробки веб-додатків та покращив свої навички проєктування бази даних. Я впевнений, що ці навички будуть корисні у моїй майбутній кар'єрі веб-розробника і допоможуть мені реалізувати ще більш складні та інноваційні проекти.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Django офіційна документація. [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.djangoproject.com/.
2. Python офіційна документація. [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.python.org/.
3. "Clean Code: A Handbook of Agile Software Craftsmanship" (Robert C. Martin).
4. "Refactoring: Improving the Design of Existing Code" (Martin Fowler).
5. "Learning Django Web Development" (Sanjeev Jaiswal).
6. "Django for Beginners: Build websites with Python and Django" (William S. Vincent).
7. "Django Design Patterns and Best Practices" (Arun Ravindran).
8. "Flask Web Development with Python Tutorial" (Corey Schafer) - доступно на YouTube. [Електронний ресурс]. – Режим доступу до ресурсу: https://www.youtube.com/watch?v=MwZwr5Tvyxo&list=PL-osiE80TeTs4UjLw5MM6OjgkjFeUxCYH.
9. "HTML and CSS: Design and Build Websites" (Jon Duckett).
10. "CSS: The Definitive Guide" (Eric A. Meyer, Estelle Weyl).

ДОДАТОК

Members.views

```
from django.contrib import messages
    from django.contrib.auth import authenticate, login, logout
    from django.contrib.auth.decorators import login_required
    from django.contrib.auth.views import PasswordChangeView
    from django.contrib.messages.views import SuccessMessageMixin
    from django.shortcuts import render, redirect, get_object_or_404
    from django.urls import reverse_lazy
    from my_site.forms import CustomerCreationForm, CustomerLoginForm, UpdateUserForm
    from my_site.models import Appointment, User, RenderedService, Employee
    from datetime import datetime


    def login_view(request):
        if request.method == 'POST':
            email = request.POST['username']
            password = request.POST['password']

            # Check if the user is a customer
                customer = authenticate(request, email=email, password=password)
            if customer is not None:
```

```
                login(request, customer)
                        messages.success(request, 'Logged in
successfully.')
                return redirect('home')

            # Check if the user is an employee
                employee = authenticate(request, email=email,
password=password)
            if employee is not None:
                login(request, employee)
                        messages.success(request, 'Logged in
successfully.')
                return redirect('home')

            error = 'Invalid email or password'
                return render(request, 'members/login.html',
{'error': error})
        else:
            form = CustomerLoginForm()
                return render(request, 'members/login.html',
{'form': form})


    def signup_view(request):
        if request.method == 'POST':
            form = CustomerCreationForm(request.POST)
            if form.is_valid():
                form.save()
```

```
                            new_user  =  authenticate(request,
email=form.cleaned_data['email'],

password=form.cleaned_data['password1'], )
                if new_user:
                    login(request, new_user)
                    return redirect('home')
        else:
            form = CustomerCreationForm()
             return   render(request,   'members/signup.html',
{'form': form})



    def logout_view(request):
        logout(request)
        return redirect('home')



    @login_required
    def profile(request):
        user: User = request.user
        current_date = datetime.now().date()
        if user.is_employee():
                                        upcoming_appointments    =
Appointment.objects.filter(appointment_date__gte=current_date
)
                                booked_times_in_appointments    =
RenderedService.objects.filter(employee_id=user.id)
```

```
                                        past_appointments     =
RenderedService.objects.filter(employee_id=user.id)

                                    available_appointments    =
upcoming_appointments.exclude(

id__in=booked_times_in_appointments.values_list('appointment_
id__id', flat=True)
            ).exclude(

appointment_date__in=past_appointments.values_list('appointme
nt_id__appointment_date', flat=True),

appointment_time__in=past_appointments.values_list('appointme
nt_id__appointment_time', flat=True)
            )

            context = {
                'appointments': available_appointments,
                'past_appointments': past_appointments,

            }
            if request.method == 'POST':
                RenderedService.objects.create(

employee_id=Employee.objects.get(pk=user.id),

appointment_id=Appointment.objects.get(pk=request.POST.get('a
ppointment_id'))
```

```
                )
                        messages.success(request,  'Accepted
appointment successfully.')

                                    return     render(request,
'members/employee_profile.html', context)
        else:
            user = request.user
                                  upcoming_appointments     =
Appointment.objects.filter(customer=user,
appointment_date__gte=current_date)
                                      past_appointments     =
Appointment.objects.filter(customer=user,
appointment_date__lt=current_date)
            if request.method == 'POST':
                                           appointment_id    =
request.POST.get('appointment_id')
                if appointment_id:
                                               appointment   =
get_object_or_404(Appointment, pk=appointment_id)
                    appointment.delete()
                        messages.success(request,  'Appointment
cancelled successfully.')

            context = {
                'user': user,
                'appointments': upcoming_appointments,
                'past_appointments': past_appointments,
            }
```

```
            return render(request, 'members/profile.html',
context)


    @login_required
    def edit_profile(request):
        if request.method == 'POST':
                    user_form = UpdateUserForm(request.POST,
instance=request.user)
            if user_form.is_valid():
                user_form.save()
                   messages.success(request, 'Your profile is
updated successfully')
                return redirect(to='profile')
        else:
            user_form = UpdateUserForm(instance=request.user)

         return render(request, 'members/edit_profile.html',
{'user_form': user_form})


    class ChangePasswordView(SuccessMessageMixin,
PasswordChangeView):
        template_name = 'members/change_password.html'
            success_message = "Successfully Changed Your
Password"
        success_url = reverse_lazy('profile')
```

My_site.views

```
    from django.contrib import messages
    from        django.contrib.auth.decorators        import
login_required
    from django.http import JsonResponse
    from django.shortcuts import redirect, get_object_or_404
    from django.contrib.auth import logout
    from django.shortcuts import render, redirect
    from  my_site.forms  import  AppointmentForm,  DateForm,
TIME_CHOICES
    from my_site.models import SalonService, Appointment
    import datetime


    def home(request):
       services = SalonService.objects.all()
             return   render(request,   'my_site/home.html',
{'services': services})


    def facelifting(request):
       return render(request, 'my_site/facelifting.html', )


    def dermabrasion(request):
       return render(request, 'my_site/dermabrasion.html', )


    def body_contouring(request):
```

```
                                return          render(request,
'my_site/body_contouring.html', )


    @login_required
    def service_booking(request, pk):
        service = get_object_or_404(SalonService, pk=pk)
        if request.method == 'POST':
            date_form = DateForm(request.POST)
            if date_form.is_valid():

Appointment.objects.create(customer_id=request.user.id,

service_id=service.id,

appointment_date=date_form.cleaned_data['appointment_date'],

appointment_time=request.POST.get('appointment_time')
                                                )
                messages.success(request, 'Appointment booked
successfully')
                                        return    render(request,
'my_site/service_detail.html', {
                    'service': service,
                    'date_form': date_form,
                })
            else:
                messages.success(request, 'Appointment booked
successfully')
```

```
                                    return   render(request,
'my_site/service_detail.html', {
                    'date_form': date_form,
                    'service': service,
                })

        appointment_form = AppointmentForm()
        return render(request, 'my_site/service_detail.html',
{
            'date_form': DateForm(),
            'service': service,
            'appointment_form': appointment_form,
        })


    def get_available_times(request):
        selected_date = request.POST.get('appointment_date')
        service_id = request.POST.get('service_id')
                                existing_appointments         =
Appointment.objects.filter(appointment_date=selected_date,

service_id=service_id)
        current_customer = request.user
                                times_of_current_user         =
Appointment.objects.filter(customer_id=current_customer.id,
appointment_date=selected_date)

        available_times = [(choice[0], choice[1]) for choice
in TIME_CHOICES
```

```
                                                                    if
datetime.datetime.strptime(choice[0],

'%H:%M:%S').time() not in existing_appointments.values_list(
                            'appointment_time',  flat=True)  and
datetime.datetime.strptime(choice[0],  '%H:%M:%S').time()  not
in

times_of_current_user.values_list('appointment_time',
flat=True)]
        return JsonResponse(available_times, safe=False)


    def about(request):
        return render(request, 'my_site/about.html')


    def logout_view(request):
        logout(request)
        return redirect('signup')
```

Models

```
    from         django.contrib.auth.base_user         import
AbstractBaseUser, BaseUserManager
    from django.contrib.auth.models import PermissionsMixin,
AbstractUser
    from django.db import models
```

```
    class EmployeeManager(BaseUserManager):
        def create_user(self, email, name, phone_number,
password=None, **extra_fields):
            if not email:
                raise ValueError('Users must have an email
address')

            user = self.model(
                email=self.normalize_email(email),
                name=name,
                phone_number=phone_number,
                **extra_fields,
            )

            user.set_password(password)
            user.save(using=self._db)
            return user

        def create_superuser(self, email, name, phone_number,
password, **extra_fields):
            user = self.create_user(
                email=email,
                name=name,
                phone_number=phone_number,
                password=password,
                **extra_fields,
            )
            user.is_staff = True
            user.is_superuser = True
```

```
            user.save(using=self._db)
            return user


    class CustomerManager(BaseUserManager):
          def  create_user(self,  email,  name,  phone_number,
password=None):
            if not email:
                  raise ValueError('Users must have an email
address')

            user = self.model(
                email=self.normalize_email(email),
                name=name,
                phone_number=phone_number,
            )

            user.set_password(password)
            user.save(using=self._db)
            return user

        def create_superuser(self, email, name, phone_number,
password):
            user = self.create_user(
                email=email,
                name=name,
                phone_number=phone_number,
                password=password,
            )
```

```
            user.is_staff = True
            user.is_superuser = True
            user.save(using=self._db)
            return user


    class User(AbstractUser):
        class Role(models.TextChoices):
            CUSTOMER = 'CU', 'Customer'
            EMPLOYEE = 'EM', 'Employee'

        username = models.CharField(max_length=255)
        phone_number = models.CharField(max_length=255)
        base_role = Role.CUSTOMER
                role    =    models.CharField(max_length=50,
choices=Role.choices, default=base_role)
        email = models.EmailField(unique=True)
        USERNAME_FIELD = 'email'
                REQUIRED_FIELDS    =    ['name',    'username',
'phone_number']

        def is_employee(self):
            return self.role == self.Role.EMPLOYEE

        def save(self, *args, **kwargs):
            if not self.pk:
                self.role = self.base_role
                return super().save(*args, **kwargs)
```

```
class Employee(User):
    base_role = User.Role.EMPLOYEE
    objects = EmployeeManager()

    class Meta:
        verbose_name = 'Employee'

    def __str__(self):
        return self.email


class Customer(User):
    base_role = User.Role.CUSTOMER
    objects = CustomerManager()

    def __str__(self):
        return self.email

    class Meta:
        verbose_name = 'Customer'

    def has_perm(self, perm, obj=None):
        return True

    def has_module_perms(self, app_label):
        return True
```

```
    class SalonService(models.Model):
        name = models.CharField(max_length=255)
        description = models.TextField()
              price    =    models.DecimalField(max_digits=10,
decimal_places=2)


    class Appointment(models.Model):
                    customer     =     models.ForeignKey(User,
on_delete=models.CASCADE)
                service    =    models.ForeignKey(SalonService,
on_delete=models.CASCADE)
                                    appointment_date          =
models.DateField(auto_created=True)
                                    appointment_time          =
models.TimeField(auto_created=True)
        created_at = models.DateTimeField(auto_now_add=True)
        updated_at = models.DateTimeField(auto_now=True)

        class Meta:
            unique_together = ('service', 'appointment_date',
'customer', 'appointment_time')


    class RenderedService(models.Model):
           appointment_id   =   models.ForeignKey(Appointment,
on_delete=models.CASCADE)
                employee_id    =    models.ForeignKey(Employee,
on_delete=models.CASCADE)
```

Admin

```
    from django.contrib import admin
    from django.contrib.auth.admin import UserAdmin

    from my_site.models import Customer, Appointment,
SalonService, Employee, RenderedService, User



    @admin.register(Customer, Appointment, SalonService,
RenderedService, User)
    class AuthorAdmin(admin.ModelAdmin):
       pass


    class EmployeeAdmin(UserAdmin):
       model = Employee
        list_display = ('email', 'phone_number', 'is_staff',
'is_superuser')
       fieldsets = (
           (None, {'fields': ('email', 'password')}),
                ('Personal info', {'fields': ('username',
'phone_number')}),
                 ('Permissions', {'fields': ('is_active',
'is_staff', 'is_superuser', 'groups', 'user_permissions')}),
              ('Important dates', {'fields': ('last_login',
'date_joined')}),
       )
       add_fieldsets = (
```

```
            (None, {
                'classes': ('wide',),
                                'fields':  ('email',  'username',
'phone_number', 'password1', 'password2'),
            }),
        )



    admin.site.register(Employee, EmployeeAdmin)
```

Diplom.urls

```
    """
    URL configuration for diplom project.

    The `urlpatterns` list routes URLs to views. For more
information please see:

https://docs.djangoproject.com/en/4.2/topics/http/urls/
    Examples:
    Function views
        1. Add an import:  from my_app import views
         2. Add a URL to urlpatterns:  path('', views.home,
name='home')
    Class-based views
        1. Add an import:  from other_app.views import Home
             2.  Add  a  URL  to  urlpatterns:     path('',
Home.as_view(), name='home')
    Including another URLconf
         1. Import the include() function: from django.urls
import include, path
```

```
          2.  Add  a  URL  to  urlpatterns:    path('blog/',
include('blog.urls'))
    """

    from django.contrib import admin
    from django.urls import path, include

    urlpatterns = [
       path('admin/', admin.site.urls),
       path('', include('my_site.urls')),
       path('members/', include('members.urls')),
                                              path('members/',
include('django.contrib.auth.urls')),
    ]
```

My_site.urls

```
    from django.urls import path
    from . import views
    from django.conf import settings
    from django.conf.urls.static import static

    urlpatterns = [
                    path('', views.home, name='home'),
                                 path('about/',  views.about,
name='about'),
                          path('logout/',  views.logout_view,
name='logout'),
                                   path('services/<int:pk>/',
views.service_booking, name='service_booking'),
```

```
                                path('get_available_times/',
views.get_available_times, name='get_available_times'),
                        path('facelifting/', views.facelifting,
name='facelifting'),
                                            path('dermabrasion/',
views.dermabrasion, name='dermabrasion'),
                                        path('body_contouring/',
views.body_contouring, name='body_contouring'),
                                ]  +  static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)                              +
static(settings.STATIC_URL)
```

Members.urls

```
    from django.conf import settings
    from django.conf.urls.static import static
    from django.urls import path

    from members import views

    urlpatterns = [
                            path('signup/', views.signup_view,
name='signup'),
                              path('login/', views.login_view,
name='login'),
                            path('logout/', views.logout_view,
name='logout'),
                               path('profile/', views.profile,
name='profile'),
```

```
                                        path('edit_profile/',
views.edit_profile, name='edit_profile'),
                                    path('password_change/',
views.ChangePasswordView.as_view(), name='password_change'),
                                ]  +  static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)                                +
static(settings.STATIC_URL)
```

service_detail.html(AJAX)

```
{% extends 'my_site/base.html' %}
{% load static custom_filters %}
{% block content %}
<div  class="service-details"  style="margin-top:  100px;
height: 700px;">
<h1 style="text-align: left">{{ service.name }}</h1>
<p style="text-align: left;
background: lightgray;
border-radius: 10px;
padding: 10px;">{{ service.description }}</p>
<form   method="post"   id="my_form"   style="text-align:
center;
}">
{% csrf_token %}
<div id="date-container">
<label  for="appointment_date_label"  style="margin-top:
30px;margin-bottom: 0px"></label>
{{ date_form.appointment_date }}
<div       id="available-times"       style="margin-top:
20px"></div>
```

```
    </div>
    <br>
    <button type="submit" class="btn btn-lg"
    style="background-color:                    #ff4242;color:
#ffffff;margin-bottom: 50px" value="Book Appointment">Book
    Appointment
    </button>
    <input    type="hidden"    id="service_id"    value="{{
service.id }}">
    </form>
    </div>


    <script>
    function getCookie(name) {
    var cookieValue = null;
    if (document.cookie && document.cookie !== '') {
    var cookies = document.cookie.split(';');
    for (var i = 0; i < cookies.length; i++) {
    var cookie = cookies[i].trim();
    // Does this cookie string begin with the name we want?
    if  (cookie.substring(0,  name.length  +  1)  ===  (name  +
'=')) {
    cookieValue                                          =
decodeURIComponent(cookie.substring(name.length + 1));
    break;
    }
    }
    }
```

```
    return cookieValue;
    }

    var availableTimesDiv =
document.getElementById("available-times");
    // Check if the div has inner fields
    if (availableTimesDiv.innerHTML.trim() === "") {
    console.log("Please select another date.");
    } else {
    console.log("Please select an appointment time.");
    }

    // Add an event listener to the form submission
    document.getElementById("my_form").addEventListener("sub
mit", function (event) {
    // Prevent the form from submitting if a time has not
been selected
    if
(!document.querySelector('input[name="appointment_time"]:chec
ked')) {
    event.preventDefault(); // Stop the form submission
    alert("Please select an appointment time.");
    return false;
    }
    // Allow the form to submit if a time has been selected
    return true;
    });
```

```
    var dateInput = document.getElementById('id_appointment_date');
    var serviceId = document.getElementById('service_id');
    dateInput.addEventListener('change', function () {
    var xhr = new XMLHttpRequest();
    xhr.open('POST', '{% url "get_available_times"%}');
    xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
    xhr.setRequestHeader('X-CSRFToken', getCookie('csrftoken')); // Include the CSRF token in the request headers
    xhr.onload = function () {
    if (xhr.status === 200) {
    console.log(xhr.responseText);
    var availableTimes = JSON.parse(xhr.responseText);
    var html = '';
    for (var i = 0; i < availableTimes.length; i++) {
    html += '<label><input type="radio" name="appointment_time" value="' + availableTimes[i][0] + '">' + availableTimes[i][1] + '</label><br>';
    }
    availableTimesDiv.innerHTML = html;
    } else {
    console.error('Request failed. Returned status of ' + xhr.status);
    }
    };
```

```
    xhr.send('appointment_date='                              +
encodeURIComponent(this.value)     +     '&service_id='     +
encodeURIComponent(serviceId.value));
    });


    </script>


    {% endblock %}
```