Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Веб-застосунок для автоматизації роботи салону краси та організації роботи з клієнтами
Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ. 200120.01.03.ПЗ

Виконав студент III курсу група ІПЗс-20-1 _____ Д. М. Бендій
Підпис Ініціали, прізвище

Керівник канд. пед. наук, доцент _____ О. Г. Онишко
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер канд. техн. наук, доцент _____ І. В, Гурман
Підпис Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії
програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

____ _____ 2023 р.

Хмельницький 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Інженерії програмного забезпечення

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Л. П. Бедратюк

05 02 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бендію Данилу Михайловичу

Прізвище, ім'я, по батькові студента

1. Тема роботи Веб-застосунок для автоматизації роботи салону краси та організації роботи з клієнтами

Керівник роботи Онишко Оксана. Григорівна.

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.03.2023 р. № 5

2. Строк подання студентом роботи на кафедру 01.06.2023 р.

3. Вихідні дані до роботи Завдання на дипломне проектування

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задачі, Проєктування програмного забезпечення, програмна реалізація, тестування програмного забезпечення

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Презентаційні матеріали (слайди, 30 шт.), Три креслення: UML –діаграма варіантів використання, UML – діаграма класів, UML – діаграма розгортання

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Гурман Іван Васильович, доцент	02.06.2023	02.06.2023
Антиплагіат	Гурман Іван Васильович, доцент	02.06.2023	02.06.2023

7. Дата видачі завдання « 02 » січня 2023р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) Кваліфікаційної роботи	Строк виконання етапів проекту (роботи)	Примітка
1 Ознайомлення з тематикою кваліфікаційної роботи (КР), визначення та узгодження індивідуальних тем КР	01.12– 31.12.2022	
2 Збір матеріалу за темою КР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	02.01 – 31.01.2023	
3 Проєктування програмного забезпечення	01.02 – 28.02 2023	
4 Програмна реалізація з використанням відповідних засобів розробки	01.03 – 10.04.2023	
5 Тестування програмного забезпечення	11.04 – 30.04.2023	
6 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КР згідно вимог	01.05 – 25.05.2023	
7 Попередній захист КР	Травень 2023 (згідно графіка)	
8 Перевірка КР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05.2023	
9 Здача КР на кафедру; підготовка КР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КР	з 01.06.2023	

Студент _____ Бендій. Д. М.
Підпис Ініціали, прізвище

Керівник проєкту (роботи) _____ Онишко О. Г
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Веб-застосунок для автоматизації роботи салону краси та організації роботи з клієнтами».

Автор проєкту: Бендій Данило Михайлович.

Керівник проєкту: Онишко Оксана Григорівна

Пояснювальна записка: 72 с.,34 рис.,3 табл., 4 дод., 47 джерел.

Графічна частина:30 слайдів.

Ключові слова: WEB, JAVASCRIPT, MONGODB, NODEJS, EXPRESS, JQuery

Кваліфікаційна робота присвячена розробці веб-застосунку для автоматизації роботи салону краси та покращення організації роботи з клієнтами. Метою роботи є створення інструменту, який допоможе салону краси зберігати інформацію про клієнтів, записувати їх на процедури, відстежувати історію візитів та надавати інформацію про послуги.

У кваліфікаційній роботі було проаналізовано предметну область задачі, з'ясовані причини та передумови створення застосунку, було проведено аналіз існуючих рішень альтернативних програмних застосунків. Також в записці були порівняні можливі архітектурні рішення для подібних застосунків та можливі патерни, які використовуються для розробки веб-застосунків, були визначені модулі для системи та здійснена їх програмна реалізація.

Впровадження даного застосунку дозволить салону краси зберігати інформацію про клієнтів та їх історію візитів, а також планувати та організовувати роботу співробітників.

В результаті роботи було розроблено веб-застосунок для автоматизації роботи салону красти та організації роботи з клієнтами.

01.06.2023
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ. 200120.01.03.ПЗ	Пояснювальна записка	72		
2	А4		Завдання на кваліфіка-	1		
			ційну роботу			
3	А4		Анотація	1		
			Графічні документи			
5	А3	КвРІПЗ. 200120.01.03.Е8	UML-діаграма варіантів	1		
			використання			
6	А3	КвРІПЗ. 200120.01.03..Е8	UML-діаграма	1		
			розгортання			
7	А3	КвРІПЗ. 200120.01.03.Е8	UML-діаграма класів	1		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ. 200120.01.03.ВД			
Виконав		Бендій Д. М.		2.06.	Веб застосунок для автоматизації роботи салону краси та організації роботи з клієнтами Відомість документів	Літ.	Арк.	Аркушів
Керівник		Онишко О.Г.		2.06				72
Рецензент						ХНУ, ІПЗс-20-1		
Н. контр.		Гурман І.В.		02.06.				
Зав. каф.		Бедратюк Л. П.		2.06				

ЗМІСТ

					КвРІПЗ. 200120.01.03.ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Виконав		Бендій Д. М.		2.06	Веб застосунок для автоматизації роботи салону краси та організації роботи з клієнтами	Літ.	Арк.	Аркушів
Керівник		Онишко О.Г.		2.06			4	72
Рецензент						ХНУ, ІПЗс-20-1		
Н. контр.		Гурман І.В.		02.06				
Зав. ка.		Бедратюк Л. П.		2.06				

Вступ

За останні роки можна спостерігати зростання інтересу до здорового способу життя та зовнішнього вигляду у більшості країн світу. Люди стають більш усвідомленими щодо того, що здоров'я тіла та душі є важливими для їхнього щастя та добробуту. Зокрема, зростає популярність різних видів фізичних вправ та спорту, здорового харчування, а також процедур та процесів догляду за зовнішнім виглядом.

Це також стимулює розвиток сфери краси та сприяє зростанню попиту на послуги салонів краси, де пропонуються різні процедури догляду за обличчям, тілом та волоссям. Відвідування салонів краси допомагає людям підтримувати свій зовнішній вигляд, поліпшувати самопочуття та підвищувати самооцінку. Зростання інтересу до здорового способу життя та зовнішнього вигляду може сприяти подальшому розвитку індустрії краси та стимулювати зростання кількості салонів краси.

Отже, здійснюючи аналіз сучасного бізнес-середовища, можна зробити висновок про те, що розробка веб-застосунку для автоматизації роботи з клієнтами є актуальною та важливою задачею для підприємств, які працюють у сфері обслуговування клієнтів, оскільки дозволяє підвищити ефективність та якість обслуговування клієнтів. Такий застосунок дозволить ефективніше взаємодіяти з клієнтами, а також підвищить якість та швидкість обробки запитів клієнтів.

Об'єктом дослідження даної роботи є процес взаємодії між клієнтами та компанією, яка надає їм послуги, а також взаємодії між персоналом всередині компанії. Конкретніше, дослідженням охоплюються способи та засоби взаємодії, що використовуються наразі, а також можливості їх поліпшення за допомогою розробки веб-застосунку для автоматизації роботи з клієнтами.

Предметом дослідження роботи є розробка веб-застосунку для автоматизації роботи з клієнтами. Тема передбачає створення програмного

забезпечення, що дозволить покращити ефективність роботи з клієнтами та підвищити якість обслуговування. В рамках проєкту будуть досліджені технології розробки веб-додатків, визначені основні функції та вимоги до застосунку, розроблена архітектура програмного забезпечення та реалізовано його функціонал.

Головна мета даної роботи- розробка веб-застосунку для автоматизації роботи з клієнтами та організації роботи салону краси з метою підвищення задоволеності клієнтів наданням якісних та швидких послуг та збільшення ефективності роботи салону краси.

Для досягнення мети роботи необхідно вирішити наступні завдання:

– провести детальний аналіз предметної області;

– проаналізувати переваги та недоліки вже існуючих рішень;

– на основі попереднього аналізу встановити вимоги до програмного продукту, на основі яких необхідно обрати найкращі та найефективніші методи і засоби для розробки програмного забезпечення;

– розробити програмне забезпечення відповідно до раніше встановлених вимог;

– провести тестові випробування.

1 Характеристика предметної області і постановка задачі

1.1 Аналіз предметної області

Веб-застосунок для автоматизації роботи підприємства та організації роботи з клієнтами - це програмне забезпечення, розроблене для полегшення та оптимізації бізнес-процесів підприємства. Цей веб-застосунок може включати різноманітні модулі, які дозволяють автоматизувати роботу різних департаментів та функціональних областей підприємства, таких як управління персоналом, фінансове планування, виробництво, складський облік тощо.

Організація роботи з клієнтами є також важливою складовою веб-застосунку для автоматизації роботи підприємства. Цей модуль дозволяє підтримувати базу даних клієнтів, їхні контактні дані та історію замовлень. Крім того, він може включати інструменти для розсилки повідомлень та сповіщень клієнтам, створення та управління замовленнями, планування та координацію записів на прийоми.

Загалом, веб-застосунок для автоматизації роботи підприємства та організації роботи з клієнтами є потужним інструментом, який може допомогти підприємству знизити витрати, підвищити ефективність та покращити задоволеність клієнтів.

Предметна область даного проєкту пов'язана з роботою салону краси та наданням послуг клієнтам. До предметної області належать процеси прийому клієнтів, запис на процедури, облік послуг та операцій з грошима, управління персоналом, а також маркетингові аспекти, пов'язані з просуванням послуг та приваблennям нових клієнтів.

Дана область є висококонкурентною та вимагає постійного вдосконалення та відслідковування трендів у галузі краси та здоров'я. Завдяки розвитку інтернет-технологій та електронної комерції, салони краси можуть забезпечувати клієнтам зручний та швидкий доступ до інформації про послуги та замовлення їх онлайн, що робить роботу більш ефективною та конкурентоспроможною.

У зв'язку зі зростанням популярності догляду за зовнішнім виглядом та здоров'ям, салони краси є важливим елементом індустрії краси та здоров'я та мають значний вплив на економіку та суспільство в цілому.

Розробка застосунку покликана вирішити певні проблеми, які виникають при управлінні таким підприємством як салон краси, а саме:

1. Необхідність організації інформаційного обліку. У салоні краси можуть виникнути проблеми зі зберіганням та обробкою інформації про клієнтів, пакети послуг, ціни та інші деталі. Впровадження відповідного веб-застосунку може розв’язати цю проблему.
2. Організація процесу запису клієнтів. Для забезпечення ефективної роботи салону краси необхідно мати ефективну систему запису клієнтів на процедури. Використання відповідного веб-застосунку для реєстрації клієнтів може зменшити кількість помилок та забезпечити більш швидке та точне записування клієнтів на процедури.
3. Організація роботи зі співробітниками. Салон краси може мати багато співробітників, що може призвести до проблем з організацією та контролем робочих годин, оплати праці та іншими аспектами управління персоналом. Використання відповідного веб-застосунку для організації роботи зі співробітниками може допомогти управляти цими процесами більш ефективно та зменшити кількість помилок.

У процесі роботи веб-застосунку для автоматизації роботи салону краси та організації роботи з клієнтами, комплекс вхідної інформації перетворюється у вихідну інформацію, що відображається на екрані користувача. Цей процес складається з кількох етапів.

Першим етапом є збір вхідної інформації, яка може бути отримана від користувачів, таких як деталі про замовлення, контактні дані клієнтів, розклад роботи майстрів та інша інформація, що відноситься до роботи салону краси.

Другим етапом є обробка цієї інформації, яка може включати перевірку на достовірність, валідацію та інші перевірки. Після цього інформація зберігається в базі даних для подальшої обробки та використання.

Третій етап полягає в перетворенні інформації відповідно до потреб користувачів та відображенні цієї інформації на екрані у вигляді зручного та легко зрозумілого інтерфейсу користувача.

Останнім етапом є забезпечення зручного та швидкого доступу користувачів до потрібної інформації через веб-застосунок, що дозволяє зменшити час на пошук та обробку інформації та підвищити продуктивність роботи салону краси.

1.2 Аналіз наявного програмно-технічного забезпечення

При проєктуванні програмного забезпечення для салону краси, важливо врахувати потреби користувачів, які шукають веб-застосунок для автоматизації роботи з клієнтами та організації роботи підприємства. На сьогоднішній день на ринку існують різноманітні додатки для салонів краси, однак, кожен з них має свої переваги та недоліки. Для успішного розроблення нового застосунку необхідно вивчити недоліки наявних рішень та знайти способи їх подолання.

Окремі приклади існуючих додатків для автоматизації роботи салонів краси та організації роботи з клієнтами включають такі:

1. **Shedul** - цей застосунок дозволяє створювати розклади роботи, приймати резервації, керувати фінансами та надавати інші функції для салонів краси.

Призначення програмного продукту: Shedul - це онлайн-система для управління салонами краси та перукарнями, яка допомагає організовувати роботу співробітників, записувати клієнтів на прийоми, керувати розкладом роботи та проводити аналітику діяльності салону.

Фірма-розробник: Shedul - це компанія, що розробляє програмні рішення для салонів краси та перукарень, заснована у 2012 році в Лондоні, Велика Британія.

Основні інтерфейсні вікна: Shedul має кілька основних інтерфейсних вікон, включаючи:

- Календар: де адміністратори можуть переглядати та керувати розкладом роботи співробітників, записувати клієнтів на прийоми, відміняти або змінювати прийоми, а також переглядати загальний огляд робочого часу.
- Форма запису на прийом: де клієнти можуть обирати доступні послуги, дати та часи прийому, вводити свої контактні дані та здійснювати запис на прийом.
- Налаштування: де адміністратори можуть налаштовувати робочий час, послуги, співробітників, ціни, відпустки та інші налаштування салону.
- Аналітика: де адміністратори можуть переглядати статистику та звіти про роботу салону, включаючи кількість прийомів, доходи, популярність послуг та інші аналітичні дані

Переваги:

- Безкоштовна версія застосунку містить велику кількість функціоналу, який задовольнить потреби багатьох салонів краси;
- Легкий інтерфейс дозволяє швидко зрозуміти, як працювати з застосунком;
- Дозволяє керувати розкладом роботи майстрів і прийомом клієнтів;
- Висока ступінь налаштовування робочого часу, нагадувань, оповіщень і т.д.;
- Надає можливість приймати оплату через PayPal і Stripe;
- Є можливість додавання фото клієнтів і їхніх контактних даних;
- Вбудована аналітика показує статистику роботи, продажів та іншу важливу інформацію.

Недоліки:

- Безкоштовна версія має обмеження в 50 записів на місяць, що може бути недостатньо для деяких салонів краси;
- Застосунок не має можливості відправки СМС-повідомлень клієнтам;
- Інтеграція з іншими застосункуми обмежена;
- Не дозволяє додавати інформацію про клієнта при бронюванні через веб-сайт, що може призвести до втрати потенційних клієнтів;
- Для відображення повного функціоналу застосунку необхідно використовувати десктопну версію, що не дуже зручно для користувачів мобільних пристроїв.

Рисунок 1.1. Логотип Shedul https://www.shedul.com/

2. **Mindbody** - цей застосунок надає можливість створювати розклади, приймати онлайн-резервації, керувати фінансами та відстежувати аналітику для салонів краси.

Mindbody - це програмне забезпечення для керування фітнес-студією, СПА-салоном, йога-центром, клубом з бойових мистецтв або будь-якою іншою сферою, де необхідний зручний інструментарій для керування розкладом класів, бронюванням, оплатою, зберіганням даних клієнтів, звітів та аналізу роботи.

Фірма-розробник: Mindbody було розроблено компанією Mindbody Inc. Компанія була заснована у 2001 році у Каліфорнії, США, і з тих пір стала однією з провідних компаній у галузі програмного забезпечення для фітнесу та здоров'я.

Основні інтерфейсні вікна: Mindbody має багато інтерфейсів для різних типів користувачів, таких як власники студій, менеджери, тренери та клієнти.

Головне вікно Mindbody містить такі функції як керування розкладом класів, бронювання, оплата, зберігання даних клієнтів, звіти та аналітика.

Переваги та недоліки:

1. Переваги:

- Mindbody має потужну функціональність та можливості для налаштування, що дозволяє власникам студій налаштовувати програму для своїх потреб.
- Mindbody має дуже зручний інтерфейс, що дозволяє користувачам легко користуватися програмою і забезпечує швидкий доступ до важливих функцій.
- Mindbody має можливість інтеграції з іншими програмами, такими як QuickBooks, Xero та інші, що забезпечує зручну роботу з фінансовими даними.

2. Недоліки:

- Вартість Mindbody може бути високою, особливо для невеликих студій або підприємств з обмеженим бюджетом.
- Деякі користувачі можуть вважати, що інтерфейс Mindbody може бути складним або заплутаним, особливо для новачків.
- Деякі функції Mindbody можуть бути обмеженими або відсутніми в певних пакетах або планах цін, що може обмежувати можливості користувачів.

Рисунок 1.2. Логотип Mindbody

https://simplechoicechiropractic.com/mindbody-logo-440-v2/

3. **Acuity Scheduling** - це онлайн-система для планування та керування записами клієнтів для бізнесу різних розмірів та сфер діяльності. Фірма-розробник - Squarespace, Inc.

Основні інтерфейсні вікна Acuity Scheduling:

- Календар: графік записів клієнтів, який відображає доступні часи для зустрічей та дозволяє легко додавати, редагувати та видаляти зустрічі.
- Форма бронювання: інтерактивна форма, яка дає клієнтам можливість самостійно записатись на прийом, вибравши зручний час, послуги та працівника.
- Особистий кабінет: забезпечує зручний доступ до перегляду та редагування персональної інформації, історії записів та оплат, а також можливість встановлення пріоритетів в графіку записів.

Переваги Acuity Scheduling:

- Простий та інтуїтивний інтерфейс, що дозволяє легко налаштовувати та використовувати систему навіть для користувачів без технічного досвіду.
- Широкі можливості інтеграції з іншими популярними програмами та сервісами, такими як Google Calendar, Zoom, Stripe та інші.
- Автоматичні повідомлення та нагадування, що зменшують кількість пропущених зустрічей та сприяють підвищенню лояльності клієнтів.

Недоліки Acuity Scheduling:

- Обмежені можливості кастомізації: не завжди можна налаштувати розклад так, як потрібно.
- Складний інтерфейс: інтерфейс Acuity Scheduling може здаватися складним для новачків, що може викликати труднощі при першому використанні.
- Висока вартість: Acuity Scheduling вимагає місячної плати за використання, що може бути високою для деяких користувачів, зокрема для невеликих бізнесів.

Рисунок 1.3. Логтип Acuity Scheduling https://www.inc.com/profile/acuity-scheduling

Недоліки цих додатків можуть включати складність використання, недостатню підтримку клієнтів, високу вартість підписки та обмежений функціонал. Розробка веб-застосунку для автоматизації роботи салону краси та організації роботи з клієнтами може вирішити ці проблеми та забезпечити більш ефективне та зручне управління салоном краси.

Таблиця 1.1. Порівняння ознак додатків-аналогів

Ознаки	**Acuity Scheduling**	**Shedul**	**Mindbody**
Призначення програмного продукту	Онлайн-система бронювання та планування зустрічей	Онлайн-система бронювання та планування зустрічей	Онлайн-платформа для управління бізнесом у сфері фітнесу, краси та здоров'я
Фірма-розробник	Acuity Scheduling	Shedul	Mindbody Inc.
Основні інтерфейсні вікна	Календар, форма бронювання, повідомлення	Календар, форма бронювання, повідомлення	Розклад класів, управління продажами, фінансами, аналітика, звіти
Переваги	Легкий у використанні, можливості інтеграції зі сторонніми програмами	Безкоштовна, простий інтерфейс, мобільний застосунок	Широкий функціонал, висока безпека, інтеграція зі сторонніми програмами, підтримка мультилокальності та мультиплатформеності
Недоліки	Обмежені можливості на безкоштовному плані, висока ціна на розширені функції	Обмежені можливості на безкоштовному плані, складність налаштування	Висока ціна на програмне забезпечення та послуги підтримки, відсутність налаштування інтерфейсу, складність в освоєнні для некваліфікованих користувачів

1.3 Вимоги до програмного продукту.

Для розробки веб-застосунку для автоматизації роботи салону краси та організації роботи з клієнтами потрібне наступне інформаційне та програмне забезпечення:

Інформаційне забезпечення:

- інформація про розклад роботи салону краси;
- перелік послуг, які надає салон краси;
- база даних клієнтів, яка повинна містити особисту інформацію, перелік послуг, які були надані та звіти по операціях з клієнтами;
- інформація про працівників салону краси, їхні контактні дані, розклад роботи, перелік послуг, які вони можуть надати.

Застосунок повинен бути доступним через веб-браузер і мати зручний та інтуїтивно зрозумілий користувацький інтерфейс. Клієнти можуть бронювати послуги у зручний для них час, персонал - легко планувати зустрічі та контролювати робочий час, а керівництво - отримувати звіти та аналітику про діяльність салону.

Щоб веб-застосунок працював належним чином, йому потрібен належний хостинг, який гарантує надійність програмного забезпечення та швидку роботу. Крім того, необхідне постійне з'єднання з Інтернетом і доступ до веб-сервера та бази даних MongoDB.

Для визначення вимог до програмного забезпечення була використана уніфікована мова моделювання UML, призначена для представлення, визначення, візуалізації та документування об'єктно-орієнтованих моделей програмного забезпечення.

Діаграма варіантів використання (сценарій поведінки, use case) є першим концептуальним представленням при проєктуванні та розробці системи. Ця діаграма складається з сутностей, варіантів використання та зв'язків між ними. При створенні діаграми можуть також використовуватися загальні символічні елементи - нотації та механізми розширення.

Суть цієї діаграми полягає в наступному. Система, що розробляється, представляється у вигляді групи акторів, які взаємодіють з системою через так звані варіанти використання. В даному випадку актор - це будь-який об'єкт, суб'єкт або система, що взаємодіє ззовні з модельованою системою. Самі варіанти використання є специфікаціями послуг (функцій), які система надає акторам.

Іншими словами, кожен варіант використання визначає набір дій, які виконує система при взаємодії з акторами. При цьому спосіб, у який ці дії реалізуються, в моделі не відображається.

Кінцевими користувачами веб-застосунку для автоматизації роботи салону краси та організації роботи з клієнтами є:

1. Працівники салону краси.
2. Клієнти салону краси.
3. Гості.

До основних інформаційних потреб користувачів можна віднести:

1. Клієнти:
   - інформація про послуги, їх ціни та тривалість;
   - інформація про наявність вільних місць на певний час;
   - історія замовлень та їх статус.
2. Працівники:
   - інформація про замовлення на їх послуги;
   - інформація про їх власні прийоми;
   - доступ до інформації про клієнтів та їхні замовлення.

   В залежності від посади функціонал працівника може розширюватись такими можливостями:
   - можливість керування користувачами в системі;
   - можливість налаштування параметрів системи та її оновлення.
3. Гості:
   - доступ до основної інформації про салон краси;
   - Реєстрація в системі.

Для забезпечення інформаційних потреб користувачі застосунок повинен забезпечувати наступний функціонал:

1. Працівник – в залежності від посади має різний функціонал «майстер» - ведення своєї роботи, запису на прийом, перегляду інформації про клієнтів, інформації про послуги, що продаються в салоні краси.

«Адміністратор» - керування системою, в залежності від доступних прав, додавання та видалення даних, редагування профілів користувачів і налаштування параметрів системи

2. Клієнти – можливість запису на прийом, перегляду свого профілю, перегляду послуг, які пропонуються в салоні краси, і їх замовлення.
3. Гості – перегляд основної інформації доступної на сайті без реєстрації, реєстрація на сайті..

Таблиця 1.2. Опис акторів

Актор	Короткий опис
Гість	Має можливість переглядати основну інформацію сайту, а також зареєструватись чи авторизуватись у системі як клієнт.
Клієнт	Має можливість переглядати інформацію про послуги на сайті, реєструватись, записуватись на прийом та скасовувати замовлення.
Майстер	Працівник салону краси, який веде свою роботу, включаючи запис на прийоми, перегляд інформації про клієнтів та послуги салону краси.
Адміністратор	В керувати даними працівників салону(змінювати, додавати нових працівників), також має дозвіл на керування даними клієнтів, можливістю редагувати замовлення(відміняти, закривати, видаляти), також має доступ до налаштування цін за послуги, а також додавання нових.

Таблиця 1.3. – Опис варіантів використання програмної системи

Актор	Найменування ВВ	Опис ВВ
Гість	Реєстрація в системі	Користувач може зареєструватися в системі за допомогою своєї електронної пошти та паролю
	Пошук та перегляд інформації	Користувач може шукати та переглядати публікації
	Авторизація	Користувач може увійти у вже створений акаунт.
Клієнт	Пошук та перегляд інформації	Користувач може шукати та переглядати публікації
	Редагування профілю	Користувач може редагувати облікові дані особистого кабінету
	Запис на процедуру	Користувач може записатись на прийом до працівника на певну годину

Кінець Таблиці 1.3

	Відміна запису	Користувач може відмінити свій запис
Майстер	Редагування списку послуг	Працівник може додавати послуги до списку доступних у салоні
	Зміна статусу замовлення клієнта	Працівник може змінити статус замовлення зв’язаного з ним(на

		виконаний/прострочений, або відхилений)
Адміністратор	Модерування працівників	Адміністратор може керувати даними працівників(редагувати, додавати нових, змінювати розклад)
	Модерування клієнтів	Адміністратор може керувати даними клієнтів(редагувати, видаляти, відхиляти замовлення)
	Огляд та редагування замовлень	Адміністратор має доступ до всіх замовлень, та може змінювати їх статус
	Налаштування сайту	Адміністратор може змінювати ціни та списки послуг(видаляти, додавати нові)

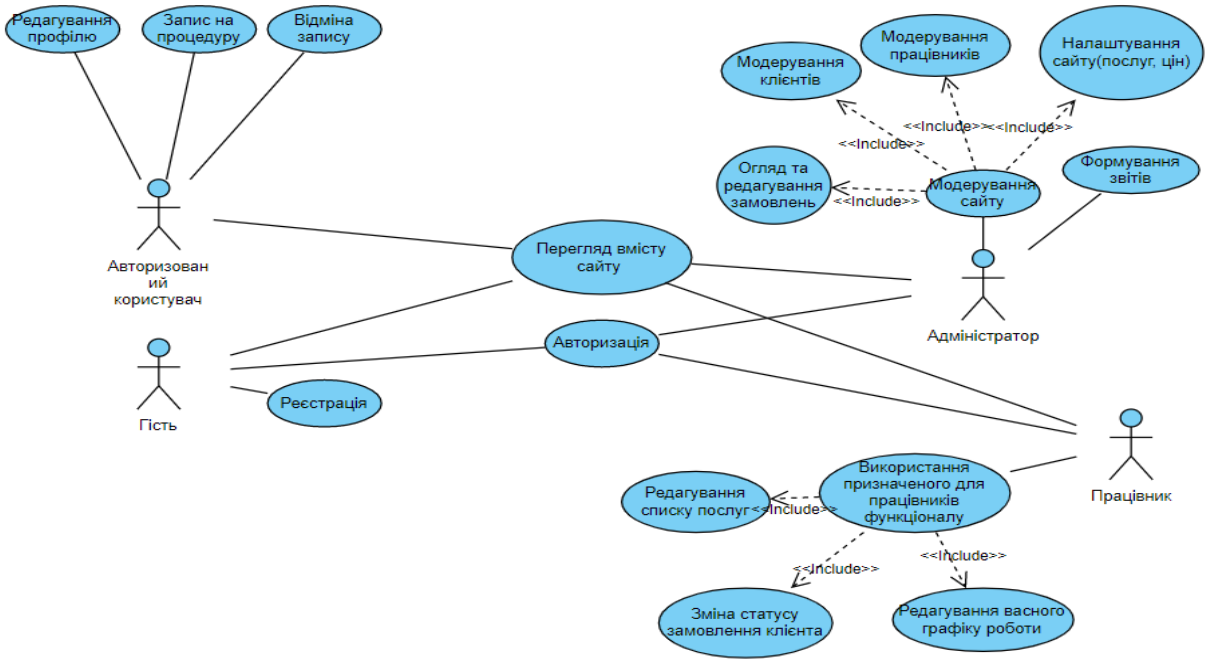


Рисунок 1.4 - Діаграма варіантів використання

1.4 Висновки. Постановка задачі.

У даному розділі були розглянуті особливості предметної області, виявлені проблеми, які існують в цій області, а також обговорені шляхи їх подолання. Були проаналізовані існуючі застосунки, виявлені переваги та недоліки..

Виконавши аналіз усіх вимог та предметної області кваліфікаційної роботи ми можемо створити список завдань які будуть стояти перед нами під час виконання кваліфікаційної роботи.

Список задач для кваліфікаційної роботи на тему «Веб-застосунок для автоматизації роботи салону краси та організації роботи з клієнтами»

- Аналіз вимог та формулювання функціональних та нефункціональних вимог до системи.
- Проєктування бази даних для зберігання інформації про користувачів, послуги, працівників та записи, використовуючи MongoDB.
- Розробка серверної частини застосунку з використанням популярного веб-фреймворку, наприклад, Node.js з Express.
- Реалізація клієнтської частини застосунку за допомогою HTML, CSS.
- Розробка функціонального застосунка який після розгортання зможе задовільняти усі потреби як власника так і користувачів.

Застосунок для автоматизації роботи салону краси та організації роботи з клієнтами повинен виконувати такі завдання:

- Реєстрацію та авторизацію користувачів
- Ведення бази даних клієнтів
- Керування робочим персоналом та облік записів.
- Бронювання та скасування послуг клієнтами

Висновки:

Після детального аналізу предметної області, пов'язаної зі салонами краси і організацією роботи з клієнтами, можна зробити висновок про великий потенціал розвитку цієї сфери. З ростом індустрії краси та зростанням

популярності послуг у цьому сегменті, автоматизація роботи салону краси стає дедалі більш важливою.

Аналіз існуючих рішень у цій галузі дозволив виявити переваги та недоліки аналогічних веб-застосунків. Цей аналіз стимулює розуміння попиту на інструменти для автоматизації та управління салонами краси. Індустрія має великі можливості для зростання та входу на ринок електронної комерції, навіть для невеликих салонів краси.

Розробка веб-застосунку для автоматизації роботи салону краси та організації роботи з клієнтами може стати успішним проєктом. Шляхом розробки технічного завдання, визначення основного функціоналу та його майбутнього розширення, можна створити потужний інструмент, що задовольнятиме потреби власників салонів краси та їх клієнтів.

Загальний висновок полягає в тому, що автоматизація роботи салону краси та організація роботи з клієнтами веб-застосунком має значний потенціал успіху. Зростання індустрії краси та збільшення попиту на ці послуги підтверджують важливість і потребу в ефективних інструментах для управління салонами краси.

2 Проєктування програмного забезпечення

2.1 Аналіз та вибір архітектури веб-застосунку

Аналіз та вибір архітектури веб-застосунку для автоматизації роботи салону краси та організації роботи з клієнтами є важливим кроком у процесі розробки застосунку. Для успішного вибору архітектури необхідно враховувати ряд критеріїв та особливостей, що стосуються функціональних та нефункціональних вимог до системи.

Один з ключових критеріїв - масштабованість. Салон краси може збільшувати свою клієнтську базу та розширювати набір послуг. Тому важливо

обрати архітектуру, яка легко масштабується та здатна витримати збільшене навантаження.

Ще один критерій - зручність розробки та підтримки. Архітектура повинна бути зрозумілою для розробників, щоб вони могли ефективно працювати над проєктом та вносити зміни в майбутньому. Крім того, важливо мати доступні інструменти та документацію, які сприяють зручній розробці та підтримці застосунку.

Також потрібно враховувати безпеку. Оскільки веб-застосунок буде містити конфіденційну інформацію про клієнтів та їх замовлення, важливо вибрати архітектуру, яка забезпечує високий рівень безпеки даних. Це може включати захист інформації, аутентифікацію та авторизацію користувачів.

Оскільки це онлайн-застосунок з вимогою до інтерактивності, обрано архітектуру Model-View-Controller (MVC).

Архітектура MVC передбачає розділення застосунку на три компоненти: модель (Model), представлення (View) та контролер (Controller). Кожен з цих компонентів виконує свої функції, що дозволяє забезпечити відокремленість бізнес-логіки, інтерфейсу користувача та обробки запитів.

Модель відповідає за бізнес-логіку та обробку даних. У випадку салону краси, модель може включати обробку записів клієнтів, розкладу роботи майстрів, управління складом продуктів та інші сутності, пов'язані з роботою салону.

Представлення відповідає за відображення даних користувачу та забезпечення інтерфейсу для взаємодії. Він може містити сторінки, форми, таблиці, графіки та інші елементи, які відображають дані та дозволяють користувачам взаємодіяти з системою.

Контролер служить посередником між моделлю та представленням. Він обробляє запити користувача, взаємодіє з моделлю для отримання необхідних даних та відправляє їх до представлення для відображення. Контролер також може включати логіку авторизації, аутентифікації та інші обробки запитів.

Обрана архітектура MVC має кілька переваг. Вона сприяє відокремленню бізнес-логіки від представлення, що полегшує розподіл ролей і відповідальностей між різними компонентами застосунку. Завдяки цьому, зміни в бізнес-логіці можуть вноситись незалежно від інтерфейсу користувача, а зміни в представленні не впливають на обробку даних та бізнес-правила.

Крім того, використання MVC дозволяє забезпечити легкість утримання та розширення застосунку. Оскільки кожен компонент виконує свої специфічні функції, можливість заміни або модифікації одного компонента не вимагає значних змін у інших. Наприклад, зміни в інтерфейсі користувача можуть бути впроваджені безпосередньо в представленні, не змінюючи бізнес-логіку та модель.

Додатковою перевагою архітектури MVC є можливість багаторазового використання компонентів. Наприклад, представлення може бути використане для різних типів клієнтських пристроїв (наприклад, веб, мобільні) без необхідності змін у бізнес-логіці. Також модель може бути використана в інших частинах системи, що спрощує розширення та використання різноманітних інтеграцій.

Загалом, обрана архітектура MVC є гнучкою, масштабованою та добре збалансованою для веб-додатків, що забезпечують автоматизацію роботи салону краси та організацію роботи з клієнтами. Вона дозволить ефективно розробляти, утримувати та розширювати застосунок з покращеною продуктивністю та забезпечить зручний інтерфейс користувача для клієнтів салону краси.

Крім того, використання архітектури MVC сприяє поділу роботи між розробниками. Кожен компонент (модель, представлення, контролер) може бути розроблений та тестований окремо, що полегшує колективну роботу над проєктом. Крім того, це дає можливість використовувати спеціалізовані фреймворки або бібліотеки для кожного компонента, що спрощує розробку та забезпечує високу якість коду.

За допомогою архітектури MVC можна ефективно реалізувати такі функціональності, як керування записами клієнтів, розклад роботи майстрів, управління запасами продуктів, статистика та звіти, онлайн-бронювання та інші функції, необхідні для автоматизації роботи салону краси та зручного взаємодії з клієнтами.

Враховуючи всі переваги архітектури MVC та вимоги до онлайн-застосунку для салону краси, її вибір є обґрунтованим і допоможе забезпечити якісний та ефективний розвиток проєкту. Однак, перед реалізацією архітектури необхідно провести детальний аналіз вимог, технічних обмежень, обрати відповідні технології та інструменти, а також розробити план реалізації та тестування, щоб забезпечити успішну розробку та впровадження веб-застосунку для салону краси.

2.2 Опис структури даних та моделі бази даних

Нижче наведена нормалізована схема бази даних для веб-застосунку автоматизації роботи салону краси, а також ER-діаграма, що візуалізує цю схему:

Таблиця: Clients

1. _id: ObjectId (ідентифікатор клієнта, PRIMARY KEY)
2. Name: String (ім'я клієнта)
3. Surname: String (прізвище клієнта)
4. Email: String (електронна пошта клієнта)
5. Phone: String (телефонний номер клієнта)
6. Appointments: Array (масив записів, пов'язаних з клієнтом)

Таблиця: Employees

1. _id: ObjectId (ідентифікатор працівника, PRIMARY KEY)
2. Name: String (ім'я працівника)
3. Surname: String (прізвище працівника)
4. Email: String (електронна пошта працівника)
5. Phone: String (телефонний номер працівника)

6. Position: String (посада працівника)
7. Schedule: Object (розклад роботи працівника)
8. Appointments: Array (масив записів, пов'язаних з працівником)

Таблиця: Services

1. _id: ObjectId (ідентифікатор послуги, PRIMARY KEY)
2. name: String (назва послуги)
3. price: Number (ціна послуги)

Таблиця: Appointments

1. _id: ObjectId (ідентифікатор запису на прийом, PRIMARY KEY)
2. client_id: ObjectId (ідентифікатор клієнта, зовнішній ключ до таблиці Clients)
3. employee_id: ObjectId (ідентифікатор працівника, зовнішній ключ до таблиці Employees)
4. service_id: Array (масив ідентифікаторів послуг, зовнішній ключ до таблиці Services)
5. date: Date (дата запису на прийом)
6. price: Number(загальна ціна)
7. status: String (статус запису)

У цій моделі бази даних таблиці Clients, Employees, Services Apointments, мають унікальні ідентифікатори (primary keys), які ідентифікують кожен запис у таблиці. Таблиця Appointments використовує зовнішні ключі для пов'язаності з іншими таблицями. Зовнішні ключі дозволяють встановлювати зв'язок між записами у різних таблицях.

Ця модель бази даних дозволяє зберігати інформацію про клієнтів, працівників, послуги та адміністраторів, а також зв'язувати їх у записах про записи на прийоми.

2.3 Проєктування серверної частини веб-застосунку

Серверна частина програмного продукту відіграє важливу роль, оскільки вона забезпечує більшість функціональності системи. Сервер відповідає за зв'язок між базою даних і тим, що відображається користувачеві на екрані, тобто клієнтською частиною.

Для створення та управління запитами використовується прикладний програмний інтерфейс (API) RESTful. RESTful API (Representational State Transfer) дозволяє компонентам клієнта та сервера взаємодіяти через мережу.

Запити відправляються на сервер та отримують інформацію з нього за допомогою чотирьох типів запитів:

- GET: для отримання даних;
- POST: для створення нових даних;
- PUT: для редагування існуючих даних;
- DELETE: для видалення даних.

Інформація з цих запитів передається на сервер у тілі запиту, а сервер здійснює зміни та зберігає їх у базі даних. Формат даних, які передаються та отримуються, визначений стандартними протоколами HTTP та HTTPS.

Одним із важливих аспектів є формат запитів та відповідей, який визначається стандартами протоколу HTTP.

На сьогодні виділяють два основні підходи до створення серверів: монолітна архітектура та мікросервісна архітектура.

Монолітна архітектура є найпоширенішою при розробці веб-додатків. Вона передбачає наявність єдиного пакета, що містить модулі, які взаємодіють між собою і утворюють багаторівневу архітектуру. Монолітна архітектура має рівні бізнес-логіки, доступу до сховища даних та представлення.

Переваги монолітної архітектури включають простоту розробки та розгортання, високу продуктивність і швидкий відгук на запити, оскільки всі компоненти працюють в межах одного процесу. Однак, у монолітної архітектури

можуть виникати проблеми з масштабованістю і модульністю, оскільки зростання розміру проєкту може призвести до складнощів у розробці і підтримці.

Мікросервісна архітектура, натомість, розбиває програмний продукт на невеликі, незалежні сервіси, кожен з яких виконує конкретні функції. Кожен сервіс може бути розгорнутий та масштабований окремо, що дозволяє більш гнучко пристосовуватись до змін і вимог системи. Комунікація між сервісами зазвичай здійснюється за допомогою мережевих протоколів, таких як HTTP або message queue.

Мікросервісна архітектура забезпечує високу модульність, масштабованість та незалежність компонентів системи. Однак, вона вимагає більшої складності управління декількома сервісами, а також збільшує навантаження на мережу та затримки при комунікації між сервісами.

Можна зробити висновок, що монолітну архітектуру доцільно використовувати при розробці простих та невеликих програмних систем, де обсяг обробки ресурсів не є дуже великим. З іншого боку, мікросервісна архітектура рекомендується для більш складних систем, де присутні багато непов'язаних між собою модулів.

У випадку теми кваліфікаційної роботи, обрання монолітної архітектури є розумним рішенням, враховуючи середню складність проєкту та обмежений час на розробку. Модулі даної системи будуть взаємопов'язані, тому декомпозиція їх на окремі сервіси стає непрактичною та вимагатиме значних зусиль для розгортання та тестування.

Архітектурним патерном розробки застосунку стане патерн MVC

MVC (Model-View-Controller) є архітектурним шаблоном, який широко використовується для розробки веб-застосунків. Він дозволяє відокремити логіку програми від її представлення та взаємодії з користувачем.

У контексті веб-застосунку для автоматизації роботи салону краси та організації роботи з клієнтами, використання MVC може бути корисним. Основні компоненти MVC включають:

1. Модель (Model): Модель представляє бізнес-логіку та дані застосунку. В цьому випадку модель може включати дані про клієнтів, розклади, послуги, запаси тощо. Вона відповідає за доступ до даних, їх оновлення та обробку.
2. Представлення (View): Представлення відповідає за відображення даних користувачу та інтерфейс користувача. Наприклад, це можуть бути сторінки веб-сайту, на яких клієнти можуть переглядати інформацію про послуги, розклади та здійснювати бронювання.
3. Контролер (Controller): Контролер обробляє вхідні події та взаємодіє з моделлю та представленням. Він приймає вхідні дані від користувача (наприклад, запити на бронювання) та виконує необхідні дії, використовуючи модель, після чого оновлює представлення.

MVC дозволяє розділити логіку, представлення та взаємодію на окремі компоненти, що полегшує розробку та підтримку коду. Він сприяє модульності та забезпечує більшу гнучкість у внесенні змін у систему.

Таким чином, використання архітектури MVC може бути корисним для розробки веб-застосунку салону краси, допомагаючи відокремити логіку бізнесу від інтерфейсу користувача та забезпечити кращу організацію та підтримку проєкту.

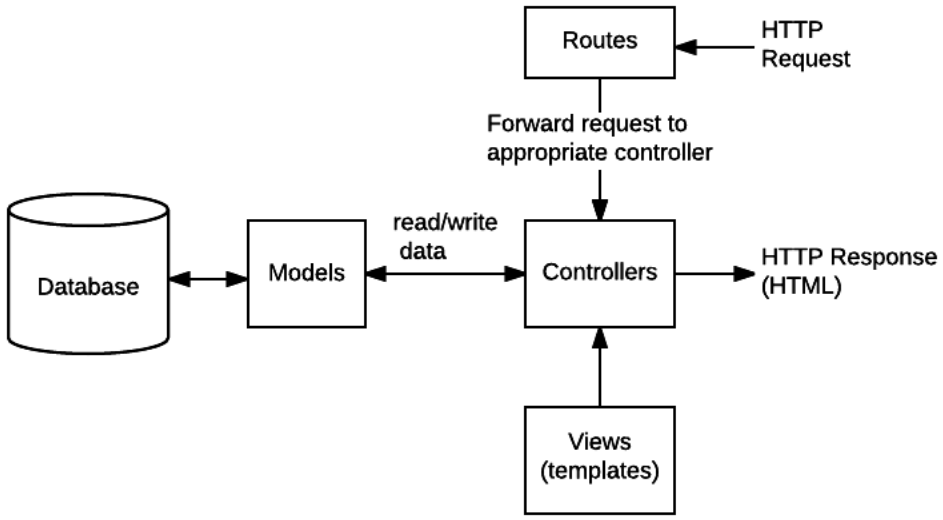


Рисунок 2.1 Структура застосунку розробленого за патерном MVC

2.4 Проєктування клієнтської частини веб-застосунку

Для розробки дизайну проєкту було вирішено використати метод прототипування. Прототипування - це етап розробки програмного забезпечення, коли створюється макет або прототип інтерфейсу, що дозволяє візуалізувати майбутнє програмне забезпечення і продемонструвати його замовнику на ранніх етапах розробки. Цей підхід допомагає уникнути багатьох ризиків на етапі проєктування.

На першому етапі необхідно створити верхню частину сторінки, яку називають «хедер». Цей блок повинен включати логотип, кнопку для повернення на головну сторінку, кнопку інформації про контакти, а також кнопки для авторизації та реєстрації (див. рисунок 2.2).

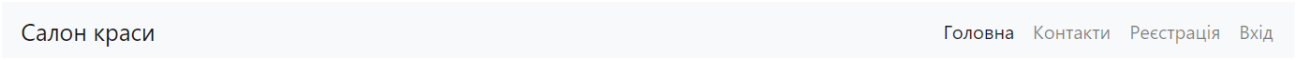


Рисунок 2.2 Хедер неавторизованого користувача

Таким чином, шляхом прототипування ми зможемо візуалізувати та продемонструвати основні елементи інтерфейсу замовнику, спрощуючи процес розробки та дозволяючи уникнути можливих проблем на етапі проєктування.

У авторизованого клієнта в хедері кнопки зміняться. Замість кнопок авторизації та реєстрації буде кнопка для виходу з облікового запису, а також з’являться кнопки переходу на корзину та на список попередніх замовлень (рисунок 2.3).

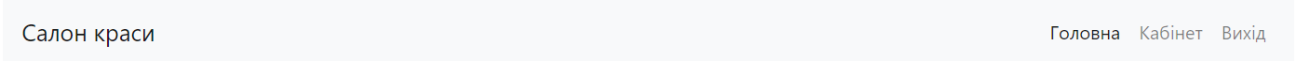


Рисунок 2.3 Хедер авторизованого клієнта

В адміністратора та майстра салону хедер буде містити кнопку, для переходу на сторінку панелі керування (рисунок 2.4).

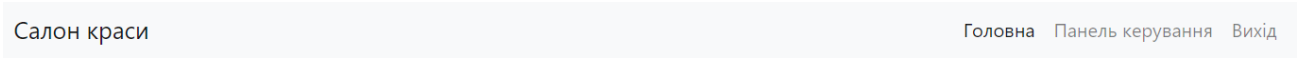


Рисунок 2.3 Хедер майстра та адміністратору салону

Далі необхідно провести проєктування екрану розроблюваного програмного продукту.

Макет головної сторінки представлено на рисунку 2.4.

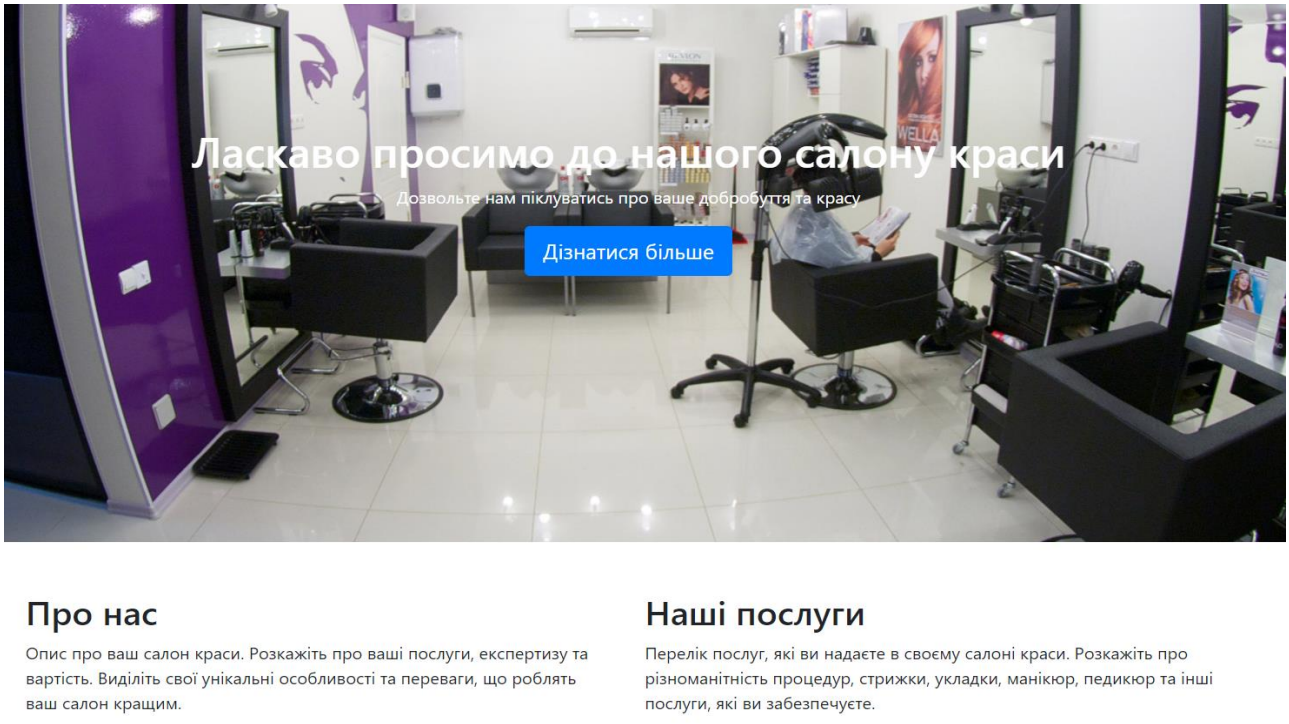


Рисунок 2.4 Макет головної сторінки

Для відображення послуг створимо представлення що буде отримувати дані про послуги і виводити їх у вигляді коротких інформаційних вікон.

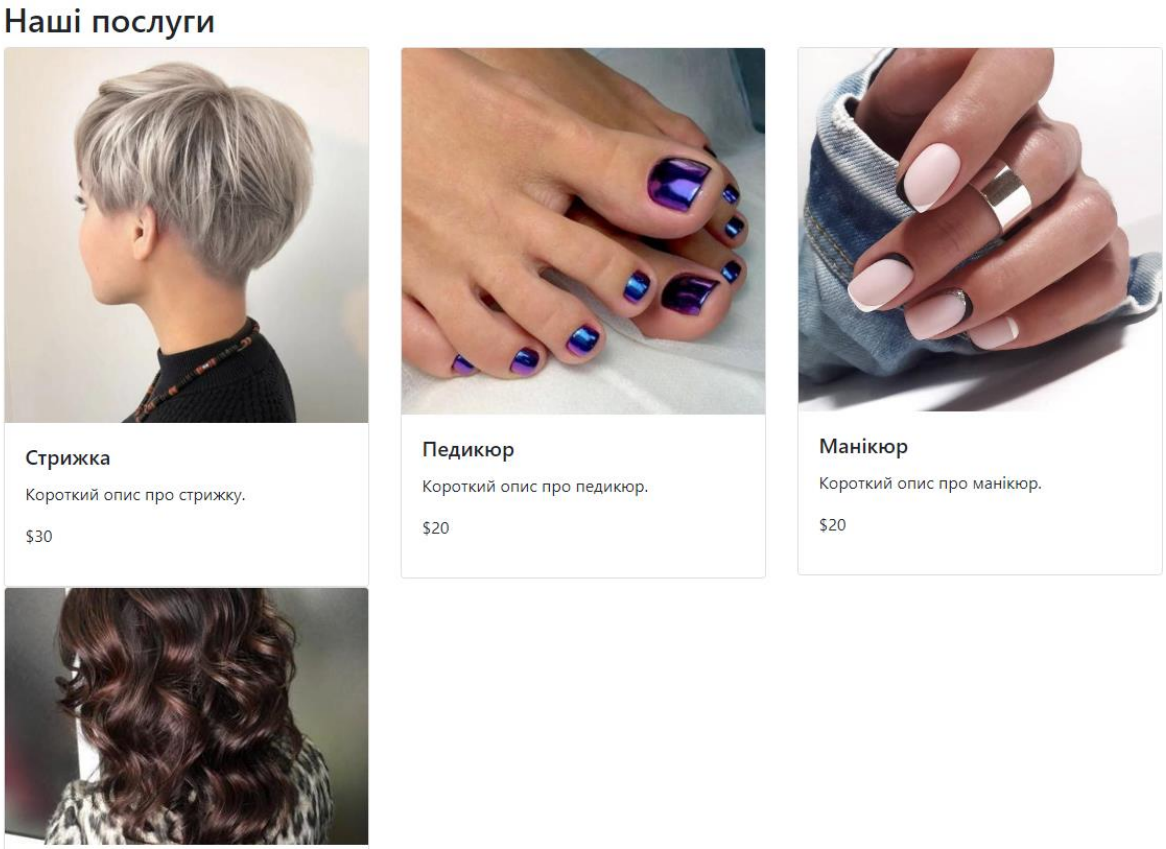


Рисунок 2.5 Сторінка послуг

Наступним кроком буде проєктування макету для панелі керування адміністратора. На ній будуть розміщені посилання на наступні сторінки управління даними такими як клієнти, працівники, прийоми, послуги. Макет сторінки панелі керування адміністратора зображено на рисунку 2.6.

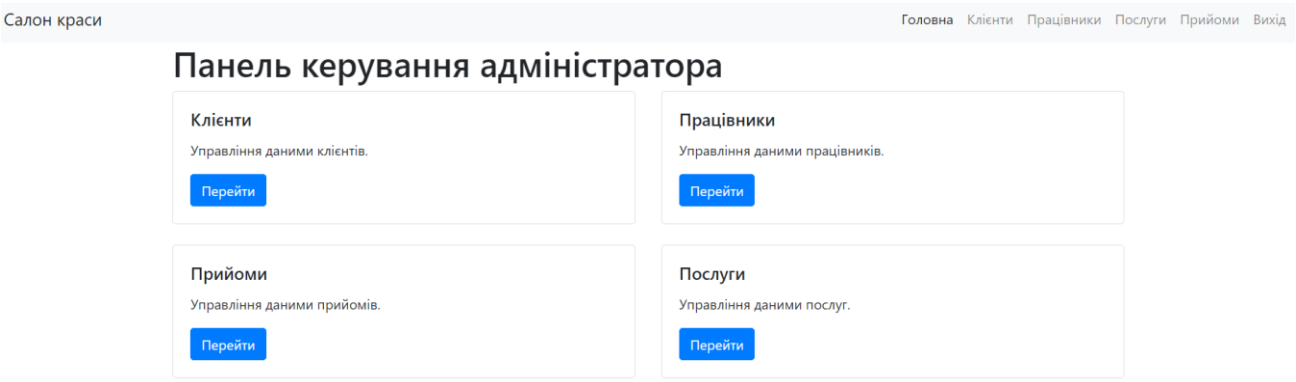


Рисунок 2.6 Макет панелі керування адміністратора

Після проєктування головної сторінки панелі керування адміністратора буде створена сторінка Клієнти, призначена для додавання видалення та редагування клієнтської інформації адміністратором. В правій частині буде

виводитись таблиця з основною інформацією про користувачів: іменем, прізвищем, поштою, номером телефону. Внизу знаходитимуться кнопки пагінації для перегляду наступних користувачів. Зліва на сторінці розташовуватиметься форма додавання нового клієнту Макет сторінки з детальної інформацією представлений на рисунку 2.7

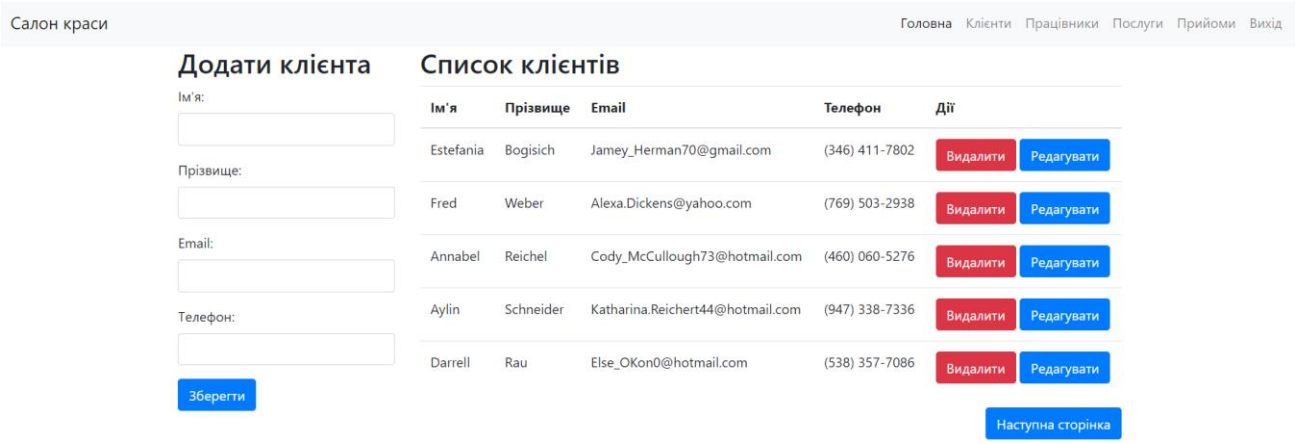


Рисунок 2.8 Макет сторінки Клієнти

Сторінка Працівники призначена для відображення списку працівників салону краси та надання можливості адміністратору виконувати дії над кожним працівником, такі як редагування або видалення. Макет сторінки містить наступні елементи: Заголовок сторінки «Працівники», Таблиця з переліком працівників, дії: кнопки «Редагувати» та «Видалити» для кожного працівника, кнопка «Додати нового працівника» для переходу на сторінку створення нового запису про працівника. Всі елементи сторінки стилізовані з використанням Bootstrap CSS для забезпечення зручного та привабливого інтерфейсу.

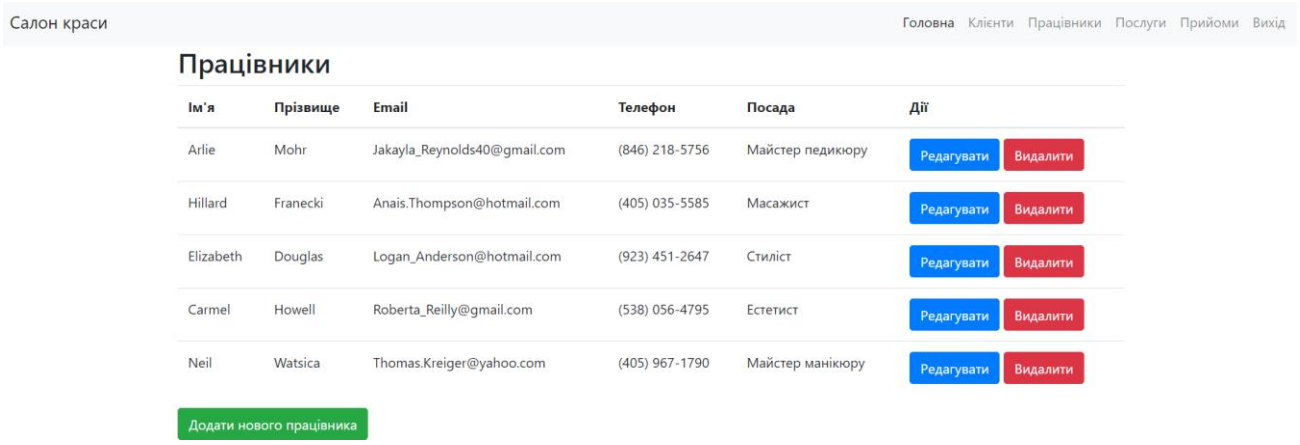


Рисунок 2.9 Макет сторінки Працівники

Сторінка «Послуги» відображає форму для додавання нової послуги в салоні краси. Форма містить поля для введення назви послуги, її ціни та опису. Після заповнення полів і натискання кнопки «Зберегти», нова послуга буде додана до списку послуг.

Також на цій сторінці відображається список існуючих послуг. Для кожної послуги в списку вказано її назву, ціну та опис. Крім того, для кожної послуги є кнопки «Видалити» та «Редагувати», які дозволяють видалити або редагувати відповідну послугу.

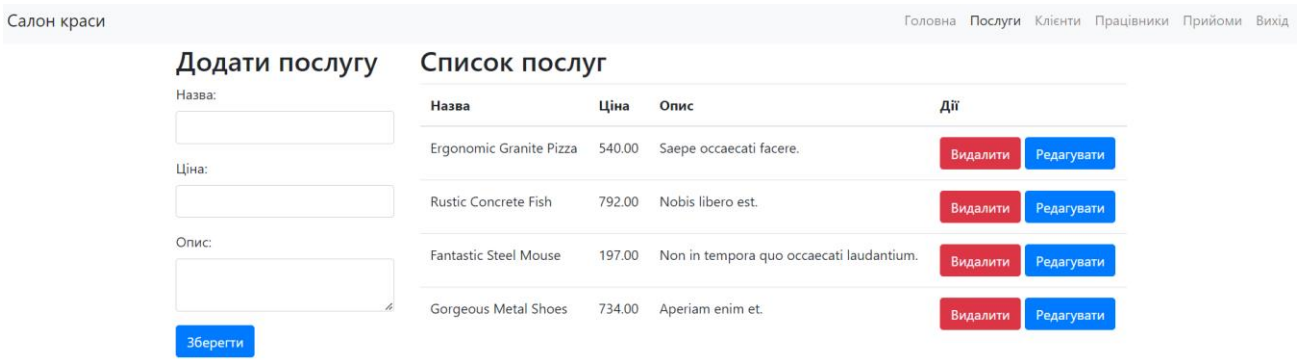


Рисунок 2.10 Макет сторінки Списку послуг

Наступна сторінка «Прийоми» відображає список прийомів у салоні краси. На сторінці виводиться таблиця з даними про прийоми, включаючи інформацію про клієнта, працівника, послуги, загальну ціну, дату та статус прийому.

Для кожного прийому в списку відображається ім'я клієнта, ім'я працівника та список послуг, які були обрані для цього прийому. Також виводиться загальна ціна прийому, дата та статус (наприклад, «Запланований» або «Підтверджений»).

Для кожного прийому також доступні кнопки «Редагувати» та «Видалити», які дозволяють адміністратору виконувати відповідні дії з прийомом, такі як редагування його деталей або видалення з системи.

Також є кнопка що викликає модальну форму для створення нового Прийому.

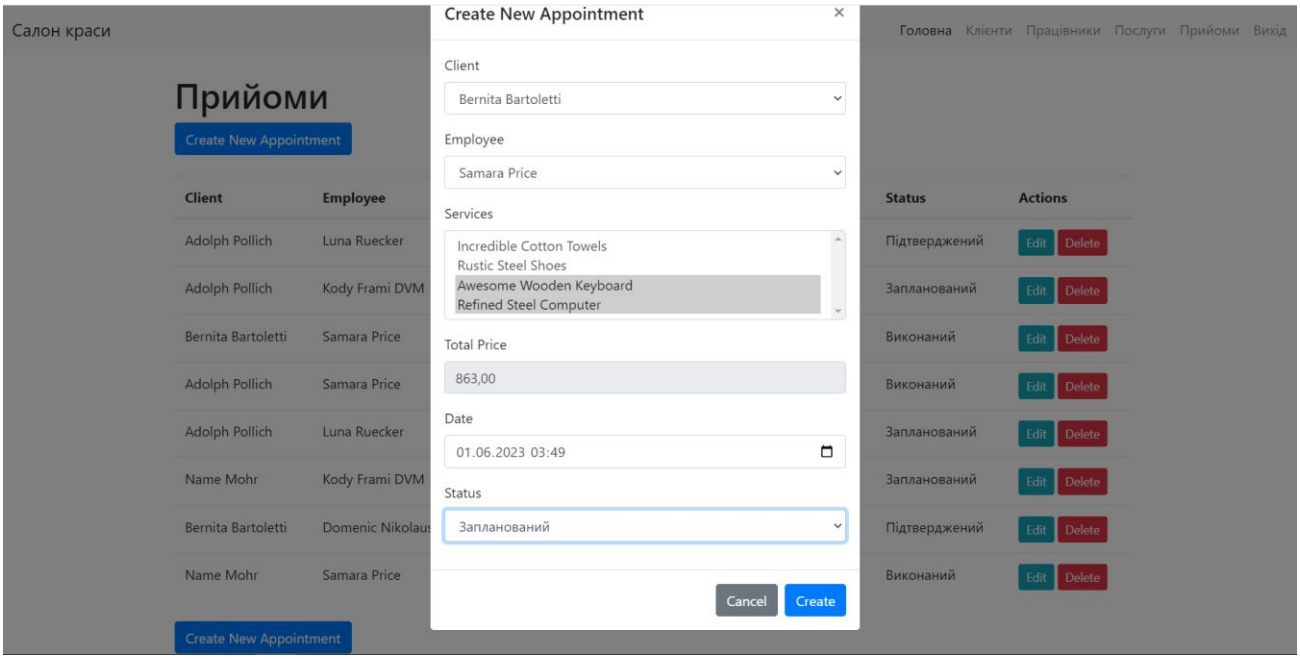


Рисунок 2.11 Макет сторінки Прийоми з модальним вікном створення прийому

Для роботи з незареєстрованими користувачами була створена форма контактів де користувач може створити запис без додаткових пунктів, які зможе обговорити персонально при відвідуванні салону чи з працівником який зв’яжеться з ним за номером телефону вказаним у заявці. Сторінка містить форму створення запису що включає в себе такі поля як: Ім’я, Прізвище, Номер телефону, та дату запису. З макетом сторінки «Контакти» можна ознайомитись на Рисунку 2.9

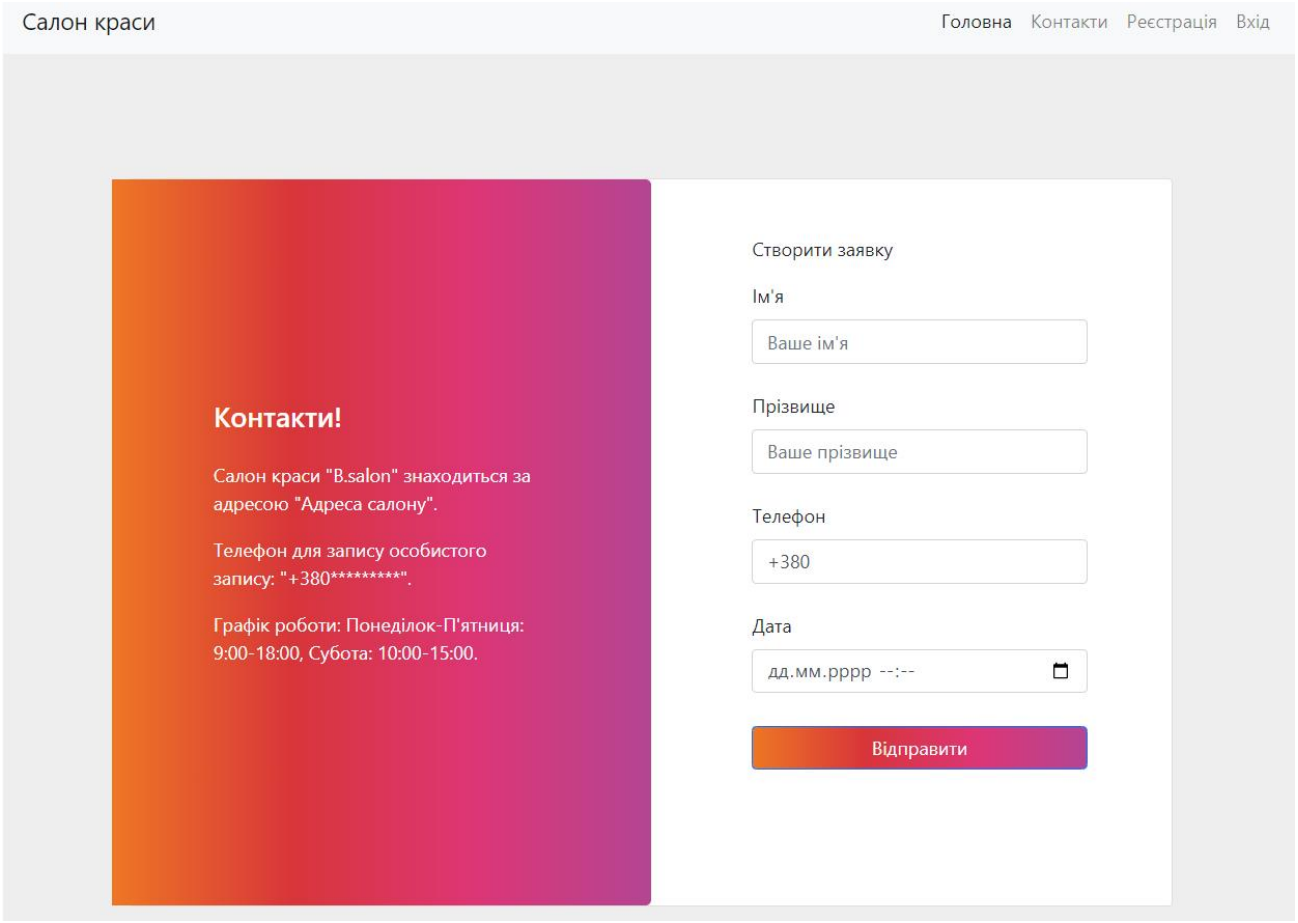


Рисунок 2.12 Макет сторінки «Конакти

Наступна сторінка є формою реєстрації для салону краси. Вона являє собою форму реєстрації, що складається з наступних полів:

- Ім'я: користувач вводить своє ім'я.
- Прізвище: користувач вводить своє прізвище.
- Email: користувач вводить свою електронну пошту.
- Номер телефону: користувач вводить свій номер телефону.
- Пароль: користувач вводить пароль.
- Підтвердження паролю: користувач повторно вводить пароль для підтвердження.
- Кнопка «Зареєструватися»: користувач натискає на кнопку для відправки форми реєстрації.

На цій сторінці користувач може заповнити форму реєстрації, введення буде перевірятися за допомогою скриптів на правильність та повинність полів.

Якщо поля заповнені неправильно або є помилки, користувач буде попереджений про це і форма не буде відправлена. Якщо дані введені правильно, форма може бути відправлена для обробки на сервері.

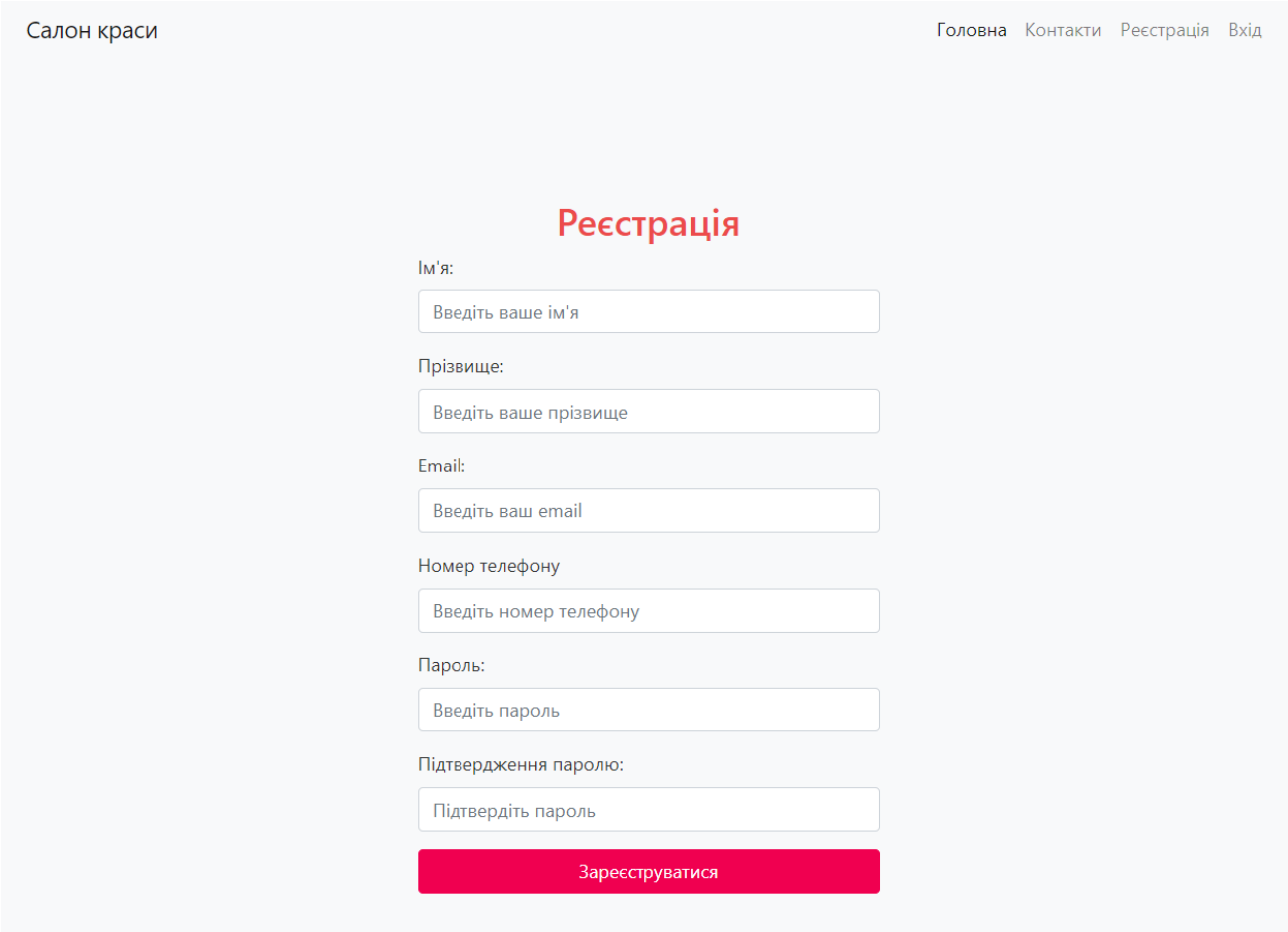


Рисунок 2.13 Макет сторінки «Реєстрація»

Наступною розробимо макет сторінки «Авторизація». Сторінка представляє собою форму авторизації для доступу до облікового запису на сайті салону краси. Ось результат, відображає: карту з формою авторизації, що розташована по центру сторінки. Ліва частина карти (колонка шириною 6): містить форму для введення даних авторизації:

- Заголовок «Увійти до акаунту».
- Поле введення «Email»: користувач вводить свою електронну пошту.
- Поле введення «Пароль»: користувач вводить свій пароль.

- Кнопка «Увійти»: користувач натискає на неї для надсилання форми авторизації.
- Рядок з посиланням «Ви не зареєстровані?» і кнопкою «Зареєструватися», які ведуть на сторінку реєстрації.
- Права частина карти (колонка шириною 6): містить інформаційний блок про салон краси: заголовок «Вітаємо!» та Короткий опис про салон краси та його послуги.

Ця сторінка використовує стилі Bootstrap для забезпечення привабливого вигляду та респонсивного дизайну. При натисканні на кнопку «Увійти» або «Зареєструватися», відповідні дані відправляються на сервер для подальшої обробки.

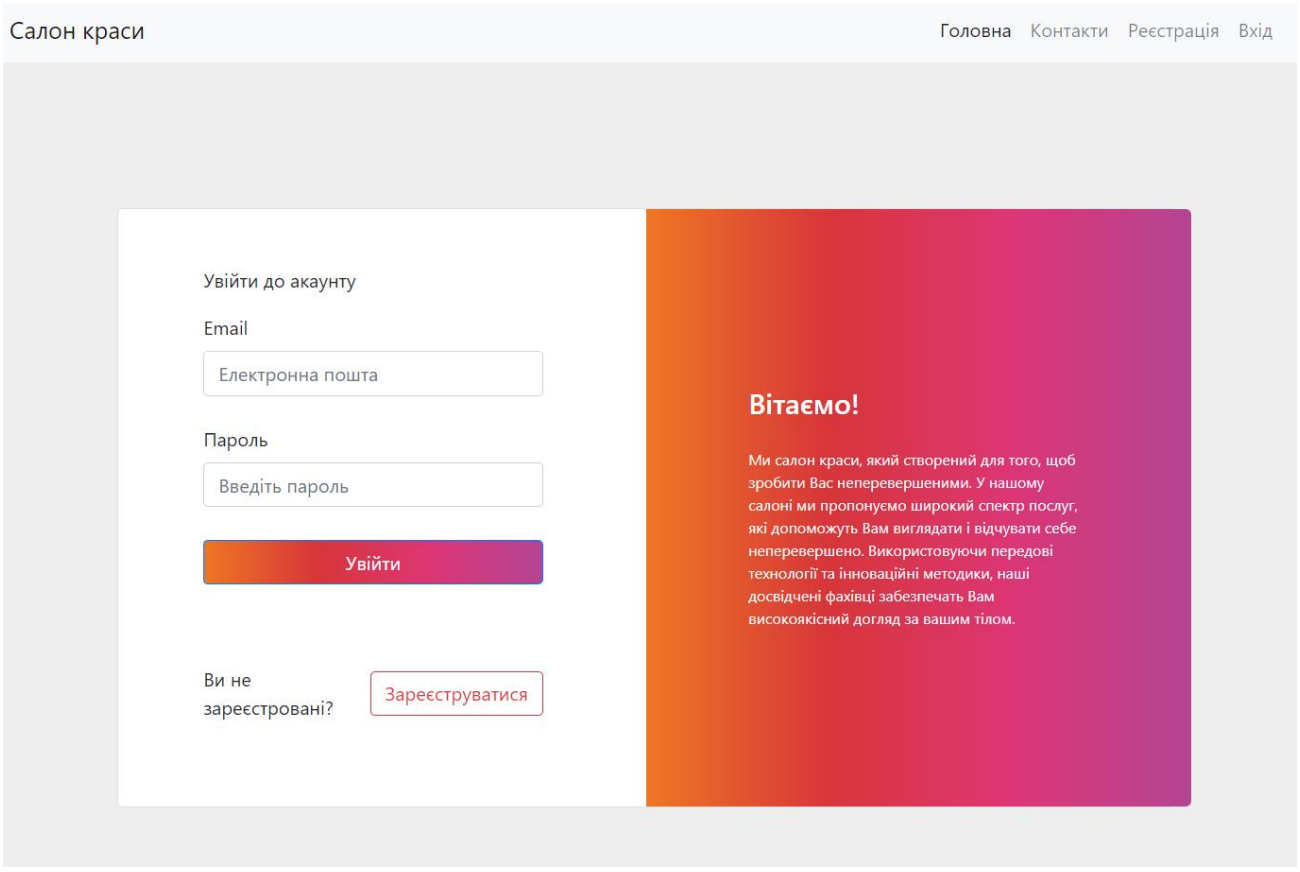


Рисунок 2.14 Макет сторінки «Авторизації»

2.5 Аналіз та вибір технологій і методів реалізації веб-застосунку

Проєктування програмного забезпечення вважається ключовим етапом розробки будь-якого інформаційного продукту. Веб-застосунок для автоматизації роботи салону краси та організації роботи з клієнтами вимагає особливої уваги на цьому етапі розробки, оскільки його ефективність залежить від правильного проєктування та реалізації функцій, які повинні бути доступні користувачам. У даному розділі будуть розглянуті основні етапи проєктування програмного забезпечення веб-застосунку, а також детально описано функції, які будуть доступні користувачам в рамках даного проєкту.

Відповідно до технічних вимог було обрано наступне програмне забезпечення та технології для реалізації.

Середовище розробки програмного забезпечення - Visual Studio Code.

Visual Studio Code - це середовище розробки Microsoft, яке має репутацію легкого, і цей програмний продукт підходить для реалізації проєкту, оскільки не містить зайвих функцій і дозволяє користувачам додавати та розширювати його, якщо вбудованих функцій недостатньо.

JavaScript має досить простий синтаксис, що дозволяє розробникам швидко створювати прототипи та розробляти програмне забезпечення. JavaScript також має велику кількість бібліотек і фреймворків, які забезпечують широку функціональність і швидку розробку.

Крім того, здатність JavaScript створювати динамічні та інтерактивні користувацькі інтерфейси робить її чудовим вибором для розробки веб- та мобільних додатків з використанням таких фреймворків, як React Native.

JavaScript є однією з найпопулярніших мов програмування на світі, яка має безліч технічних переваг, серед яких можна виділити наступні:

1. Клієнт-серверна архітектура: JavaScript виконується на стороні клієнта, що дозволяє мінімізувати навантаження на сервер та знизити час відповіді.
2. Швидкість: JavaScript дозволяє створювати динамічні веб-сторінки, які відображаються миттєво, без перезавантаження сторінки.

3. Широке застосування: JavaScript може використовуватися як на стороні клієнта, так і на стороні сервера, а також для розробки мобільних додатків та настільних застосунків.
4. Розширюваність: Багато бібліотек та фреймворків, таких як React, Angular та Vue, дозволяють розширювати можливості JavaScript, роблячи його більш функціональним та потужним.
5. Можливості взаємодії з користувачем: JavaScript дозволяє створювати взаємодію з користувачем в режимі реального часу, що забезпечує більш ефективну роботу з клієнтами та покращує їх враження від використання застосунку.
6. Підтримка відкритих стандартів: JavaScript базується на відкритих стандартах, що дозволяє розробникам створювати вільно розповсюджувані додатки та інструменти без необхідності використання пропрієтарного ПЗ.

Таким чином, вибір мови програмування JavaScript виправданий її популярністю, доступністю на різних платформах, простотою синтаксису та можливістю розробляти динамічні та інтерактивні інтерфейси.

Оскільки темою мого особистого проєкту є веб-додатки, то для його виконання я обрав середовище Node Js.

Node.js є популярною платформою для розробки веб-додатків і має ряд переваг, які роблять її придатною для розробки веб-додатків для автоматизації роботи салонів краси та організації роботи з клієнтами.

Node.js базується на движку V8, який використовується в браузері Google Chrome. Це дозволяє розробникам писати код, який працює дуже швидко і з мінімальним часом відгуку.

Node.js використовує мову JavaScript, яка є дуже популярною серед веб-розробників. JavaScript дозволяє розробникам створювати високоякісний код швидко та ефективно. Node.js надає ряд пакетів і модулів, які розробники можуть використовувати для розширення функціональності своїх додатків.

Node.js використовує асинхронні функції та події, що дозволяє застосункум ефективно працювати з багатьма запитами одночасно. node.js має велику та активну спільноту, що гарантує стабільність та постійну підтримку.

Тому вибір Node.js для розробки веб-додатків для автоматизації роботи салонів краси та організації роботи з клієнтами є логічним і перспективним.

Мовою програмування було обрано JavaScript. Причинами вибору стала її популярність і має велика спільноту розробників. Вона підтримується на багатьох платформах, включаючи веб-, мобільні та серверні середовища, і може використовуватися для створення широкого спектру додатків.

Також е роботі буде використано фреймворк Express.Js. Express.js є фреймворком для розробки веб-додатків на мові JavaScript для веб-сервера Node.js. Express дозволяє легко і швидко створювати веб-сервер, використовуючи модульний підхід до розробки веб-додатків. Express надає широкі можливості для маршрутизації, що дозволяє розподіляти запити користувачів на різні частини застосунку, а також вбудований рендеринг HTML-сторінок з шаблонів. Також він підтримує роботу зі статичними файлами, підключення різних модулів і пакетів, механізми обробки помилок і багато іншого. Express дозволяє розробникам зосередитися на бізнес-логіці свого застосунку, мінімізуючи зусилля на налаштування інфраструктури веб-сервера.

Також важливим кроком є вибір формату даних для отримання результатів запиту з серверу.

Найпопулярнішими форматами для відносно невеликих розмірів даних є XML і JSON. XML - строгість форматування та простота перевірки.

Порівняно з іншими форматами, XML виглядає дуже об'ємним. Кожен елемент даних повинен представитися детальною структурою, тому об'єм даних, що передається структурою виявляється малим.

Для аналізу даних такого формату також доводиться докладати чимало зусиль. Необхідно знати структури даних, а потім це все помістити в JS об'єкт. Тому це далеко не найкращий вибір формату даних для використання.

JSON - формат, який є невеликим у розмірах та простим в аналізі форматом даних.

Нижче наводиться приклад списку користувачів:

```
[
    {“i”: 1, “u”: “alice”, “e”: “alice@gmail.com”},
    {“i”: 2, “u”: “alex”, “e”: “alex@gmail.com”},
]
```

Як видно з прикладу список є масивом об'єктів, а кожен об'єкт – це користувач. Синтаксичний аналіз рядка очевидно буде швидшим і комфортнішим у використанні.

Для створення стилів застосунку, у зв’язку з його перевагами, було обрано Bootstrap — безкоштовний набір інструментів з відкритим кодом, призначений для створення вебсайтів та вебзастосунків, який містить шаблони CSS та HTML для типографіки, форм, кнопок, навігації та інших компонентів інтерфейсу, а також застосунку розширення JavaScript.

Основні інструменти:

- сітки - задані розміри колонок, які можно одразу ж використовувати;
- шаблони - фіксовані або резинові шаблони документів;
- медіа - надає можливість управляти відображеннями та відео;
- таблиці - засоби оформлення таблиць, навіть надання сортування;
- форми - класи для оформлення форм і деяких подій, що з ними відбуваються;
- навігація - класі для оформлення панелей, вкладок, переходів між сторінками, меню і панелі інструментів;
- алерті - оформлення діалогових вікон, підказок та спливаючих вікон;

Веб-застосунок для автоматизації роботи салону краси та організації роботи з клієнтами - це програмне забезпечення, яке використовується для зберігання та управління інформацією про клієнтів, послуги та персонал салону краси.

Визначившись з технологіями необхідними для створення застосунку необхідно сформувати основні програмні підсистеми веб-застосунку:

1. Підсистема управління клієнтами: дозволяє зберігати інформацію про клієнтів, їхні замовлення та історію відвідування салону. Клієнт може створювати профілі, редагувати свої персональні дані, записуватися на послуги та перевіряти статус замовлення.
2. Підсистема управління послугами: дозволяє створювати список послуг, які надає салон краси, та забезпечує можливість редагування цього списку. Клієнти можуть обирати послуги зі списку, записуватися на них та скасовувати.
3. Підсистема управління персоналом: дозволяє зберігати інформацію про працівників салону, їх розклад роботи, облік робочого часу та заробітної плати. Адміністратор може додавати нових працівників, редагувати їхні дані та призначати робочі години.

2.6 Висновки проектування програмного забезпечення застосунку

Після проведення проєктування програмного продукту можна зробити наступні висновки. Перш за все, було прийнято рішення використовувати клієнт-серверну архітектуру для створення застосунку після ретельного аналізу її переваг і недоліків. Також був проведений детальний аналіз і створена структура даних. Клієнтська частина застосунку була спроєктована таким чином, щоб повністю задовольняти потреби користувачів та клієнтів. Вона приваблива візуально і може залучити нових користувачів до відвідування салону краси. Після ретельного аналізу існуючих рішень були обрані найкращі технології для бази даних, клієнтської та серверної частин. Таким чином, обраний стек технологій і архітектура дозволять вирішити всі проблеми і завдання, що стоять перед розробкою програмного забезпечення.

3 Програмна реалізація

3.1 Розробка бази даних

Node.js дозволяє зручно взаємодіяти з базами даних MongoDB. Щоб це зробити, потрібно встановити драйвер MongoDB для Node.js за допомогою команди:

npm install mongodb

Для початку роботи з MongoDB, вам необхідно створити кластер MongoDB Atlas і завантажити зразки даних. MongoDB Atlas - це хмарний сервіс для керування базами даних MongoDB, який дозволяє легко створювати та керувати кластерами баз даних.

Ось приклад коду для підключення до кластера MongoDB Atlas за допомогою Node.js:

```
const { MongoClient } = require('mongodb');

const uri = «mongodb+srv://<username>:<password>@<your-cluster-url>/test?retryWrites=true&w=majority»;

const client = new MongoClient(uri);

async function connectToMongoDB() {

  try {

    await client.connect();

    console.log(«Connected to MongoDB Atlas»);

    // Далі ви можете виконувати запити до бази даних

      } catch (err) {

    console.error(«Error connecting to MongoDB Atlas:», err);

  } finally {

    await client.close();
```

```
    console.log(«Disconnected from MongoDB Atlas»);
  }
}
```

В цьому прикладі, username і password ви маєте замінити на свої облікові дані користувача, а <your-cluster-url> - на URL вашого кластера MongoDB Atlas.

Після виконання підключення, ви можете виконувати різні запити до бази даних за допомогою методів, що надаються бібліотекою mongodb. Наприклад, ви можете виконати операції вставки, оновлення, видалення документів, а також виконувати запити на вибірку даних.

Необхідно замінити <username>, <password> та <your-cluster-url> на відповідні дані вашого кластера MongoDB Atlas, щоб підключитися до вашої бази даних.

Створення бази даних MongoDB для веб-застосунку для автоматизації роботи салону краси та організації роботи з клієнтами потребує виконання наступних дій:

- Встановлення та налаштування MongoDB: У застосунку використано хмарний сервіс MongoDB, Atlas. Налаштовані параметри доступу та безпеки, які відповідають потребам застосунку.
- Створення бази даних: Запустіть MongoDB та створіть нову базу даних для вашого застосунку. Надайте їй ім'я, яке буде відображати призначення бази даних.
- Визначення колекцій: Колекції в MongoDB відповідають таблицям у реляційних базах даних. Визначте колекції, які будуть зберігати дані вашого веб-магазину, наприклад, колекцію «Послуги» для зберігання інформації про послуги салону краси.
- Визначення схеми даних: Визначте структуру даних для кожної колекції у MongoDB. Наприклад, для колекції «Послуги» - поля, такі як «назва», «ціна», «опис», тощо. Визначте типи даних та інші обмеження, які необхідні для кожного поля.

– Запис даних: Додайте записи до вашої бази даних, ввівши дані про послуги салону краси в колекцію «Послуги». Кожен запис буде представляти окрему послугу та її атрибути.

– Запити до бази даних: Виконуйте запити до бази даних MongoDB, щоб отримувати, оновлювати та видаляти дані. Наприклад, ви можете виконувати запити для пошуку послуг за назвою, сортування за ціною, підрахунку кількості послуг тощо.

– Індексування: У MongoDB діє система автоматичної генерації унікальних індексів для екземплярів колекцій, це забезпечує більшу унікальність даних і збільшує безпеку.

Захист даних: Забезпечте безпеку вашої бази даних MongoDB шляхом встановлення прав доступу, аутентифікації та інших заходів безпеки. Використовуйте ролі та дозволи, щоб обмежити доступ до бази даних лише для авторизованих користувачів. UML-діаграма класів зображена на рисунку 3.1

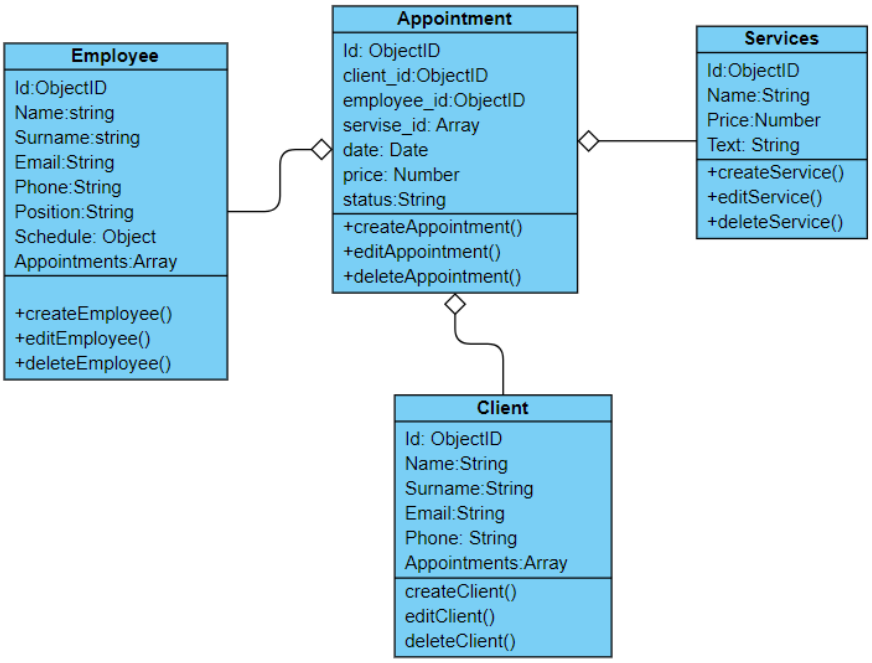


Рисунок 3.1 UML-діаграма класів

3.2 Розробка програмних модулів

Як було зазначено у розділі «2.4 Проєктування серверної частини застосунку патерном для розробки програми стане патерн MVC.

У парадигмі Model-View-Controller (MVC), застосунок Node.js може бути організований за допомогою наступних компонентів:

1. Моделі (Models): Моделі відповідають за управління даними застосунку. Вони включають у себе бізнес-логіку, взаємодіють з базою даних і забезпечують збереження, отримання та оновлення даних. Моделі можуть бути представлені в форматі об'єктів або класів, залежно від вибраної бібліотеки або фреймворка.
2. Представлення (Views): Представлення відповідають за відображення даних користувачу. Вони створюють HTML-сторінки, відправляють їх на клієнтську сторону і відповідають за взаємодію з користувачем (наприклад, обробка подій). Представлення можуть містити шаблони, які динамічно заповнюються даними з моделей перед відправкою користувачу.
3. Контролери (Controllers): Контролери координують взаємодію моделей та представлень. Вони обробляють запити від користувача, виконують відповідні дії, оновлюють моделі та вибирають відповідне представлення для відображення результатів. Контролери можуть приймати дані від користувача, валідувати їх та передавати до моделей для обробки.
4. Маршрутизатор (Router): Маршрутизатор визначає, які URL-шляхи (роути) відповідають певним контролерам. Він визначає, який контролер буде обробляти запити на певному шляху. Маршрутизатор дозволяє налаштувати маршрути для різних дій, таких як отримання, оновлення, видалення даних і т.д.

Розробку модулів почнемо з опису структури проєкту зображеної на рисунку 3.2

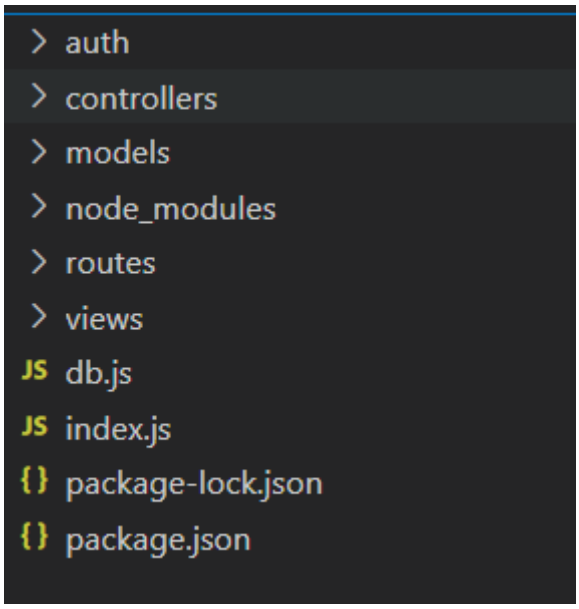


Рисунок 3.2 – Структура програми

Застосунок містить папки розподілені згідно архітектури MVC. Де папка models – містить моделі бізнес-логіки застосунку, папка controllers – відповідає за контролери, папка routes – за маршрутизацію проєкту, папка views – за представлення, а папка auth – за аутентифікацію користувачів.

Папка models, зображена на рисунку 3.3, створена для опису структури в об’єкта в базі даних. В проєкті є такі моделі як: appointment, client, employee, service, user(вона необхідна лише для аутентифікації).

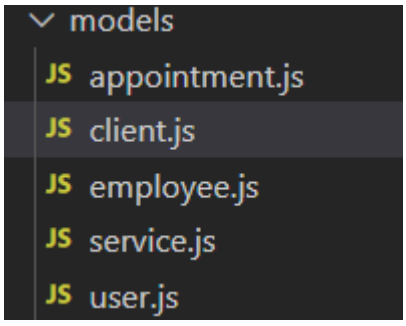


Рисунок 3.3 Папка models

Приклад, створення моделі Клієнта:

```
const mongoose = require('mongoose');
const clientSchema = new mongoose.Schema({
  name: {
```

```
    type: String,
    required: true,
  },
  surname: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
  },
  phone: {
    type: String,
    required: true,
  },
  password:{
    type: String,
    required: true,
  },
  appointments: [
    {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Order',
    },
  ],
});
const Client = mongoose.model('Client', clientSchema);
module.exports = Client;
```

Папка controllers містить усі контролери які використовує застосунок, усы контролери застосунку зображені на рисунку 3.4

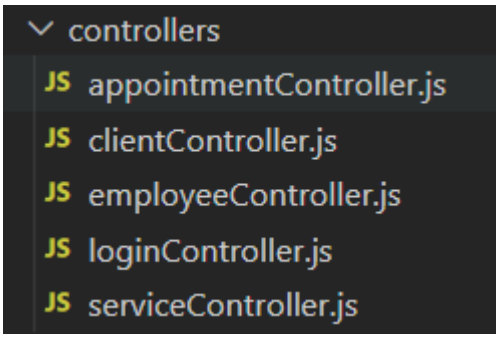


Рисунок 3.4 Контролери застосунку

Приклад – serviceController.js

```
// Підключення необхідних модулів та залежностей
const Service = require('../models/Service'); // Приклад моделі Service
// Оголошення та експорт об'єкта контролера
const serviceController = {
  // Метод для отримання всіх послуг
  getAllServices: async (req, res) => {
    try {
      // Отримання всіх послуг з бази даних
      const services = await Service.find();
      // Відправлення списку послуг у відповідь
      res.json(services);
    } catch (error) {
      console.error(error);
      res.status(500).json({ message: 'Помилка сервера' });
    }
  },
  // Метод для створення нової послуги
  createService: async (req, res) => {
    try {
      // Отримання даних з тіла запиту
      const { name, description, price } = req.body;
      // Створення нового об'єкту послуги
      const newService = new Service({
        name,
```

```
        description,
        price,
      });
      // Збереження нової послуги в базі даних
      const savedService = await newService.save();
      // Відправлення збереженої послуги у відповідь
      res.json(savedService);
    } catch (error) {
      console.error(error);
      res.status(500).json({ message: 'Помилка сервера' });
    }
  },
};
module.exports = serviceController;
```

Також у папці проєкту розміщено такі файли як index.js та db.js.

Файл index.js це основний файл проєкту. Цей файл є прикладом конфігурації застосунку Express.js з використанням різних модулів та налаштувань. Ось опис його структури та функціональності:

1. Підключення необхідних модулів:

   const express = require('express'); const app = express(); const session = require('express-session'); const bodyParser = require('body-parser'); const passport = require('passport'); const faker = require('faker');

2. Налаштування сесій:

   app.use(session({ secret: 'oneboy', saveUninitialized: true, resave: true }));

3. Налаштування шаблонізатора EJS:

   app.set('view engine', 'ejs');

4. Парсинг JSON-тіл запитів:

   app.use(bodyParser.json());

5. Парсинг URL-кодованих тіл запитів:

   app.use(bodyParser.urlencoded({ extended: true }));

6. Підключення до бази даних:

const connectDB = require('./db'); connectDB();

7. Налаштування шляху до папки з видами (шаблонами):

app.set('views', __dirname + '/views');

8. Ініціалізація та налаштування Passport для аутентифікації:

app.use(passport.initialize()); loginCheck(passport); app.use(passport.session());

loginCheck - це функція, яка налаштовує стратегії Passport для аутентифікації, а **passport.session()** дозволяє Passport працювати з сесіями користувача.

9. Підключення головного роутера:

const mainRouter = require('./routes/main'); app.use('/', mainRouter);

Цей код підключає роутер, що обробляє шляхи на головному рівні застосунку.

10. Запуск сервера на вказаному порті:

const port = 3000; app.listen(port, () => { console.log(`Server is running on port ${port}`); });

В даному випадку, сервер запускається на порті 3000.

Тоді як файл db.js це файл підключення до MongoDB за допомогою бібліотеки Mongoose. Ось опис його функціональності:

1. Підключення необхідних модулів:

const mongoose = require('mongoose');

2. Оголошення функції **connectDB**, яка виконує підключення до бази даних:

const connectDB = async () => { try { await mongoose.connect('mongodb+srv://dan:qwe1@cluster0.qjrtmiy.mongodb.net/Bsalon?retryWrites=true&w=majority', { useNewUrlParser: true, useUnifiedTopology: true }); console.log('Підключено до бази даних'); } catch (error) { console.log('Помилка підключення до бази даних:', error); } };

У цій функції використовується метод **mongoose.connect**, який приймає URL-рядок підключення до бази даних, а також застосунку параметри для налаштування підключення.

3. Експорт функції **connectDB** для використання в інших частинах застосунку:

   module.exports = connectDB;

Це дозволяє імпортувати функцію **connectDB** в інші файли і використовувати її для підключення до бази даних MongoDB.

Файл **connectDB.js** виконує важну роль в застосунку, оскільки він відповідає за підключення до бази даних і дозволяє взаємодіяти з нею за допомогою Mongoose.

За зберігання представлень у застосунку відповідає папка views, у ній зберігаються усі представлення розбиті за групами користувачів такими як admin, client та employee структура папки зображена на рисунку 3.5

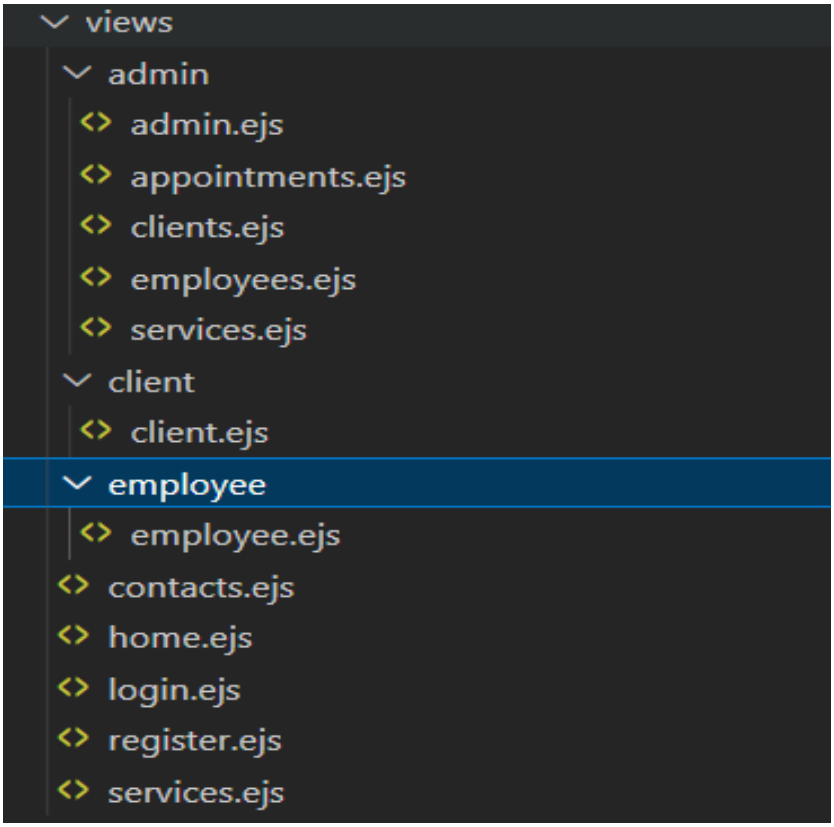


Рисунок 3.5 Структура папки views

3.3 Керівництво користувача

У цьому розділі надані покрокові інструкції щодо користування веб-застосунком для різних категорій користувачів, зокрема працівника-адміністратора, працівника-майстра салону, зареєстрованого користувача та гостя. Дотримуйтесь наведених нижче кроків, щоб успішно використовувати застосунок.

1. Для працівника-адміністратора:
    - Увійдіть до системи за допомогою своїх облікових даних адміністратора

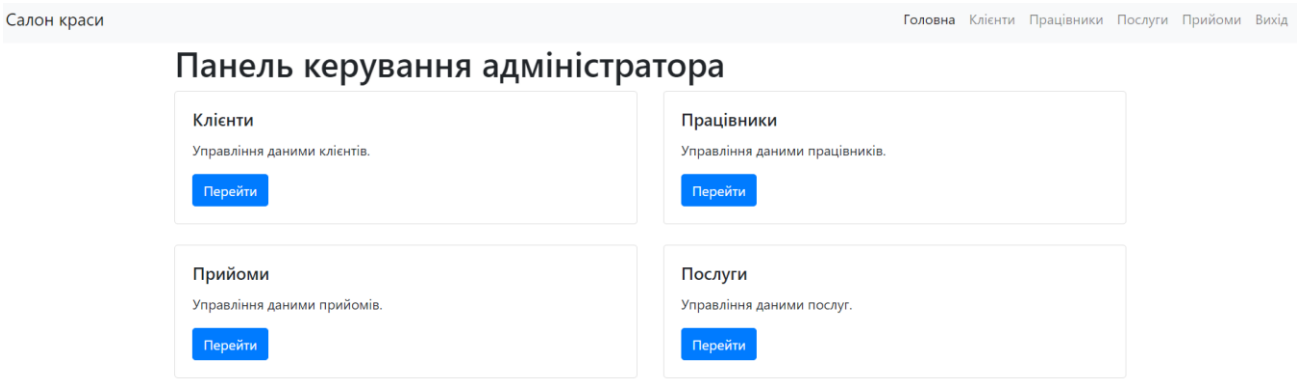


Рисунок 3.6 Панель керування адміністратора

- Перегляньте загальну інформацію про салон краси та його персонал.
- Додавайте або видаляйте працівників зі списку співробітників салону.

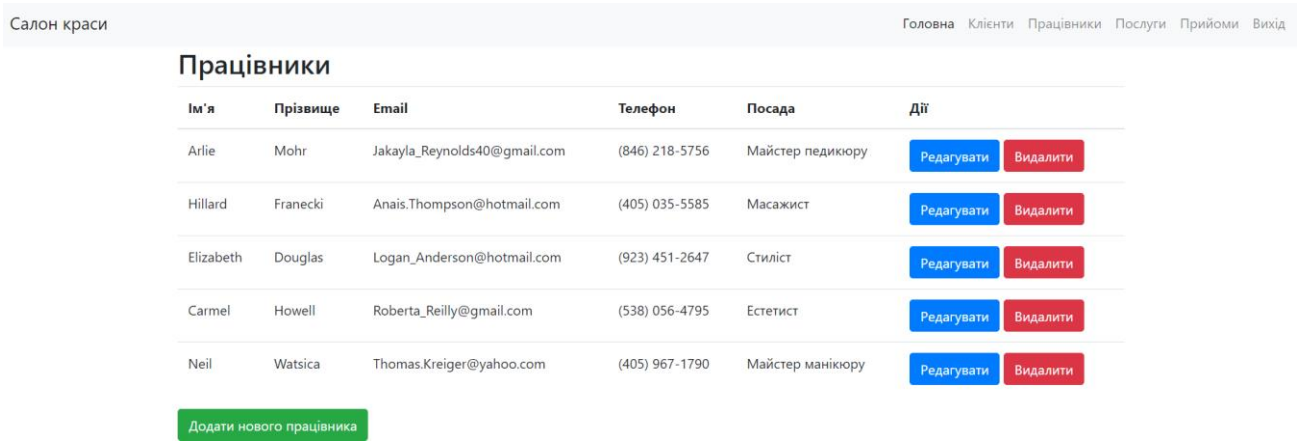


Рисунок 3.7 Сторінка керування працівниками

- Додавайте або видаляйте послуги, які надає салон краси.

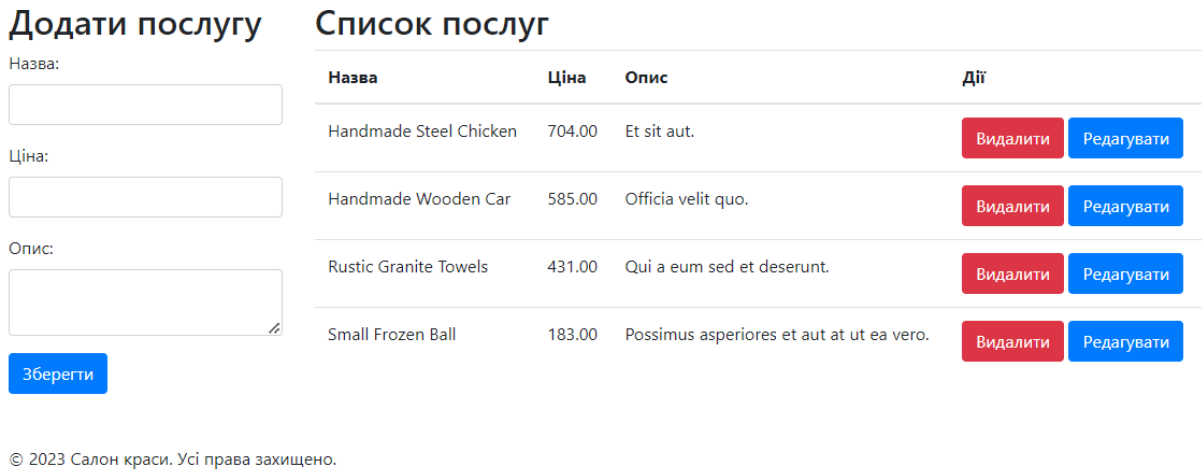


Рисунок 3.8 Сторінка керування послугами

- Додавайте або видаляйте клієнтів салону краси.

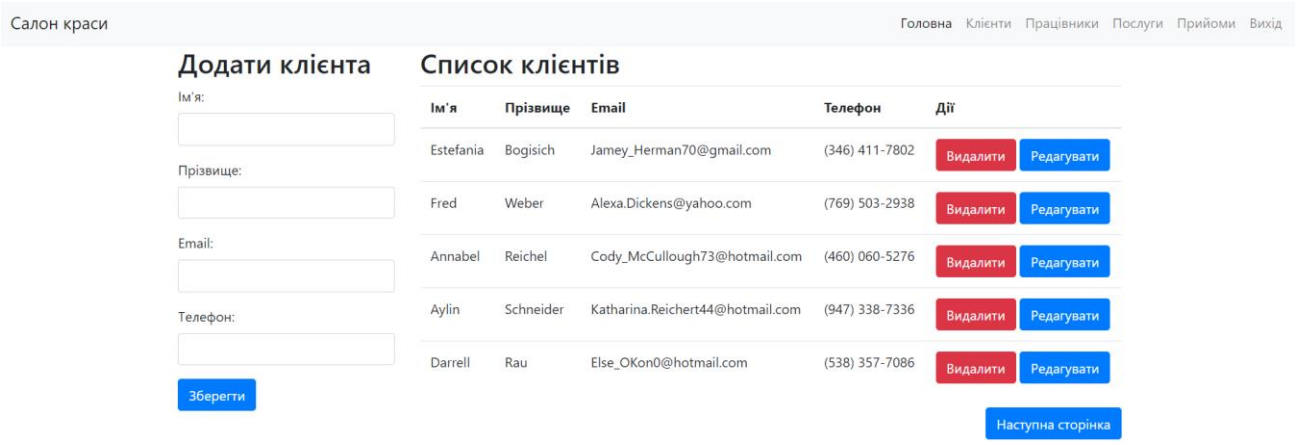


Рисунок 3.9 Сторінка керування клієнтами

- Додавайте або видаляйте записи на прийом.

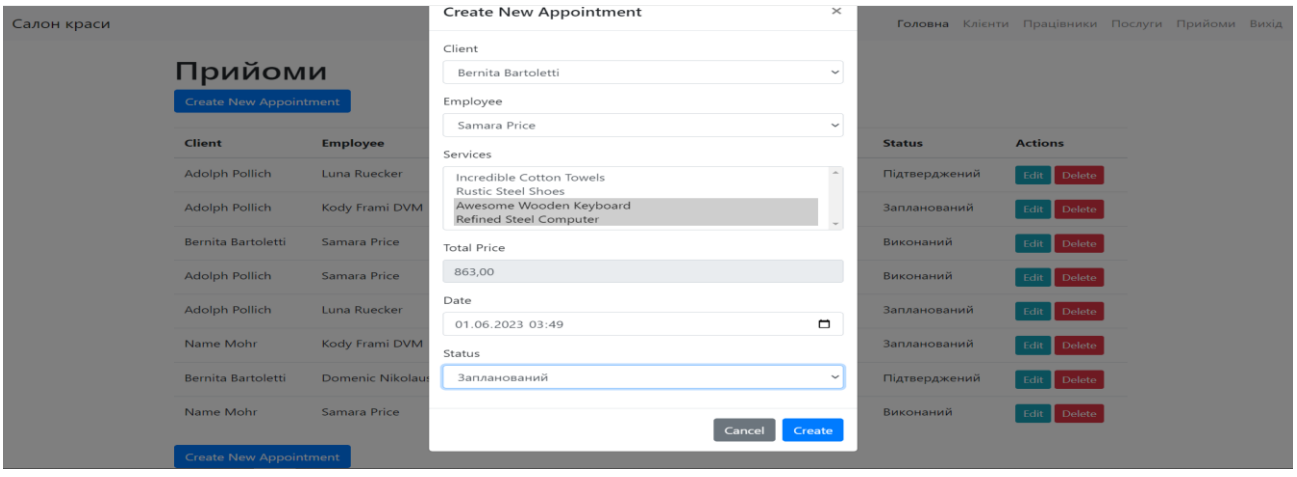


Рисунок 3.10 Сторінка керування записами на прийом

2. Для працівника-майстра салону:
    - Увійдіть до системи за допомогою своїх облікових даних.
    - Перегляньте свій розклад роботи та призначені вам клієнти та їхні послуги.
    - Записуйте нові апоінтменти для клієнтів та надавайте послуги.
    - Скасовуйте або переносьте існуючі апоінтменти у разі потреби.
3. Для зареєстрованого користувача:
    - Зареєструйтеся в системі, заповнивши обов'язкові поля форми реєстрації.

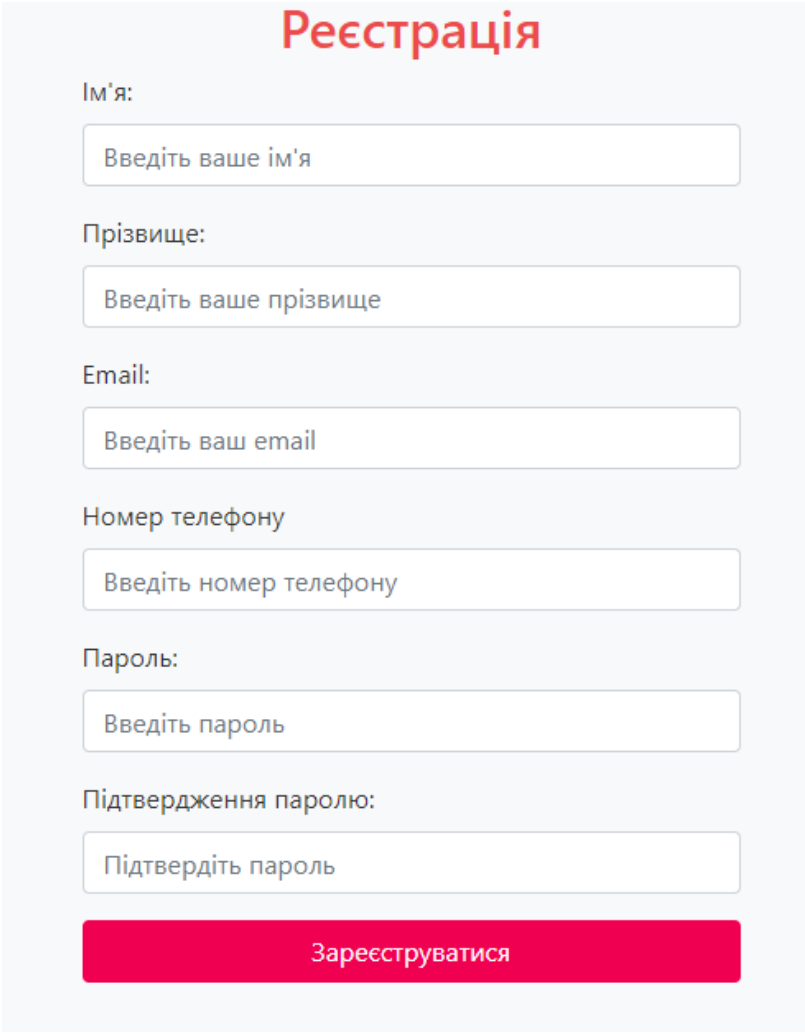


Рисунок 3.11 Сторінка реєстрації

- Увійдіть до системи за допомогою своїх облікових даних.

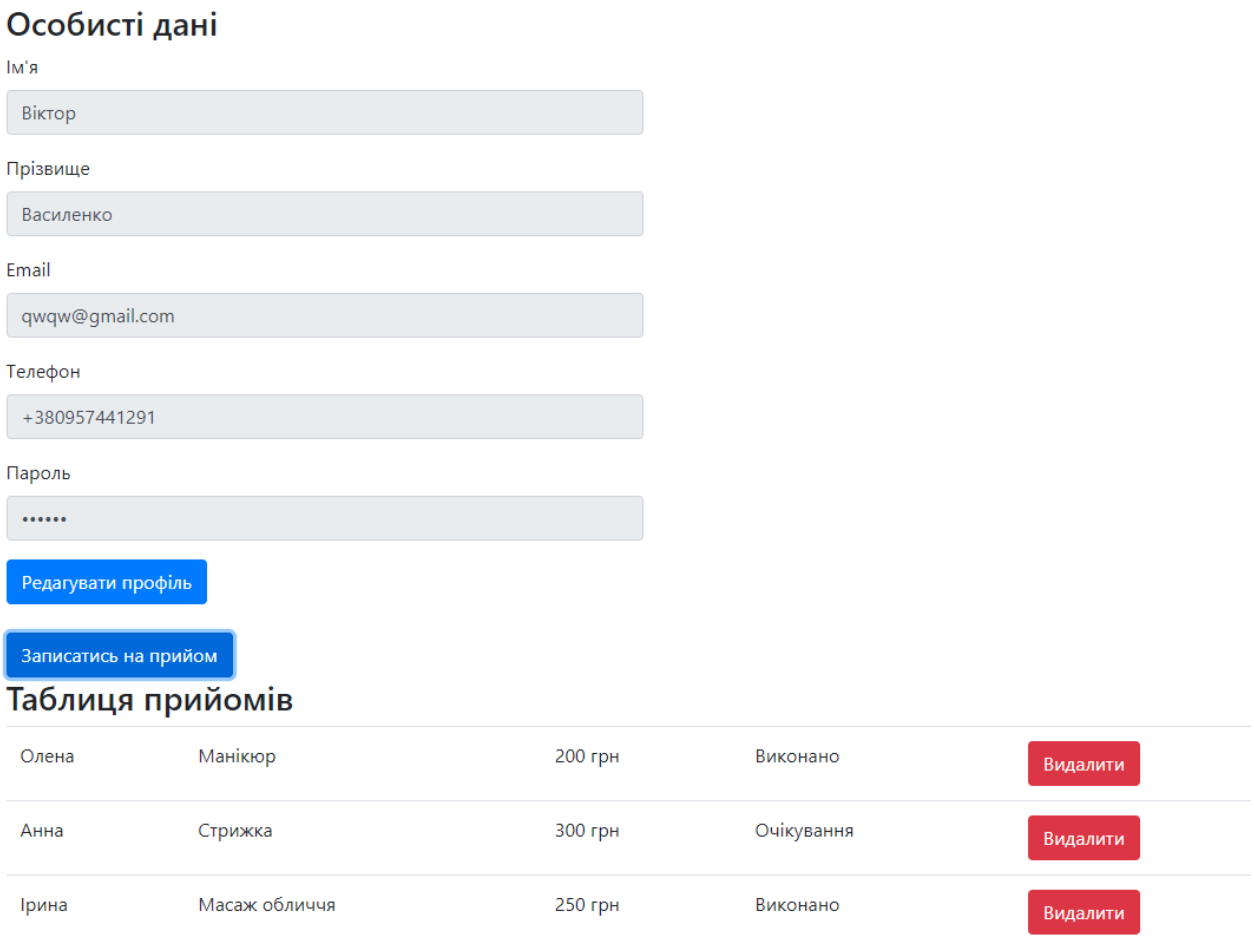


Рисунок 3.12 Особистий кабінет користувача

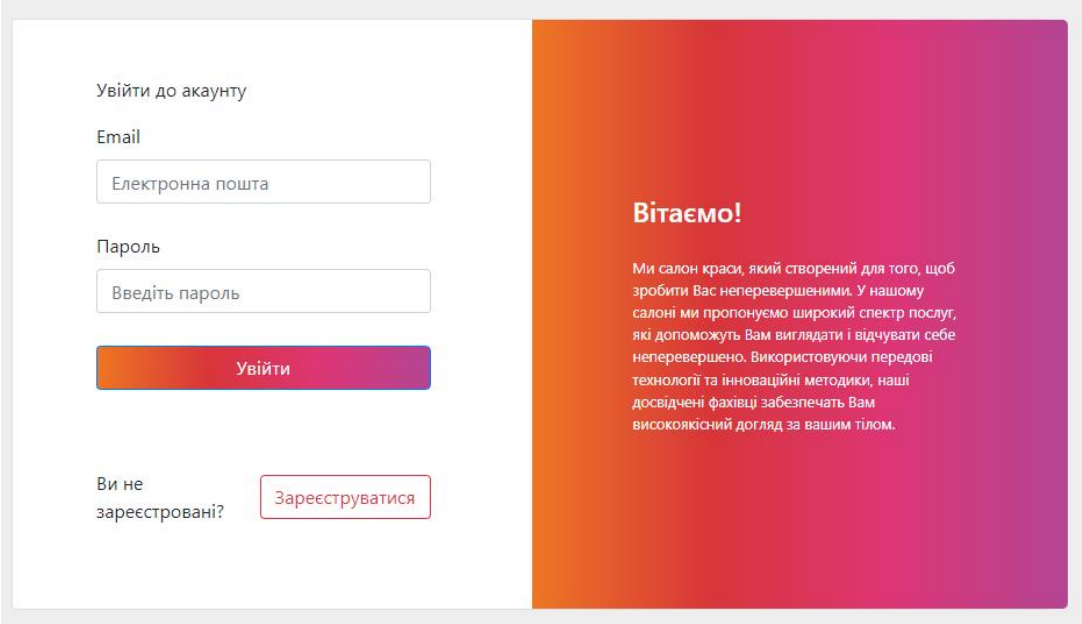


Рисунок 3.13 Сторінка авторизації

- Перегляньте список доступних послуг та їх ціни.

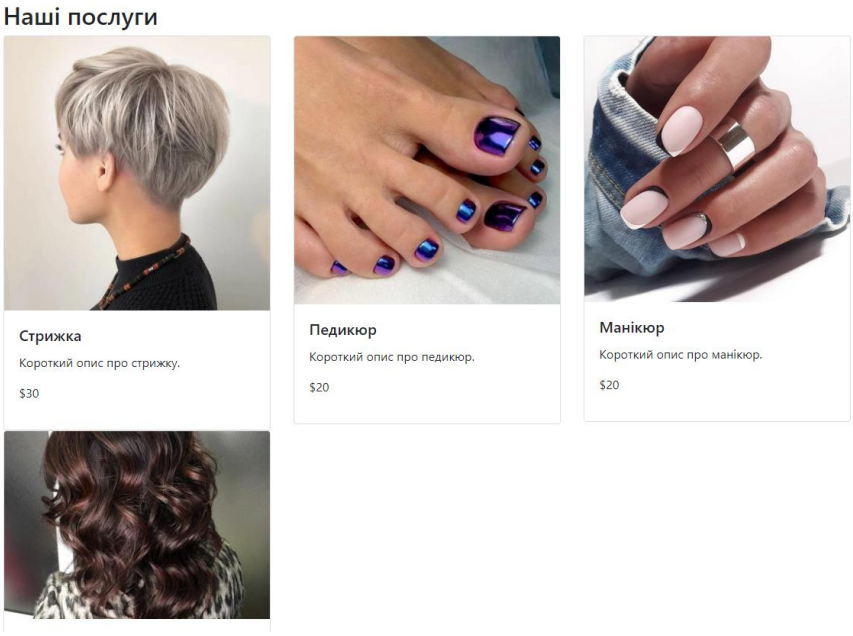


Рисунок 3.14 Сторінка послуг

- Запишіться на обрану послугу, вказавши зручний для вас час та дату.

4. Для гостя:

    - Відвідайте веб-застосунок без необхідності реєстрації або входу.
    - Перегляньте загальну інформацію про салон краси.

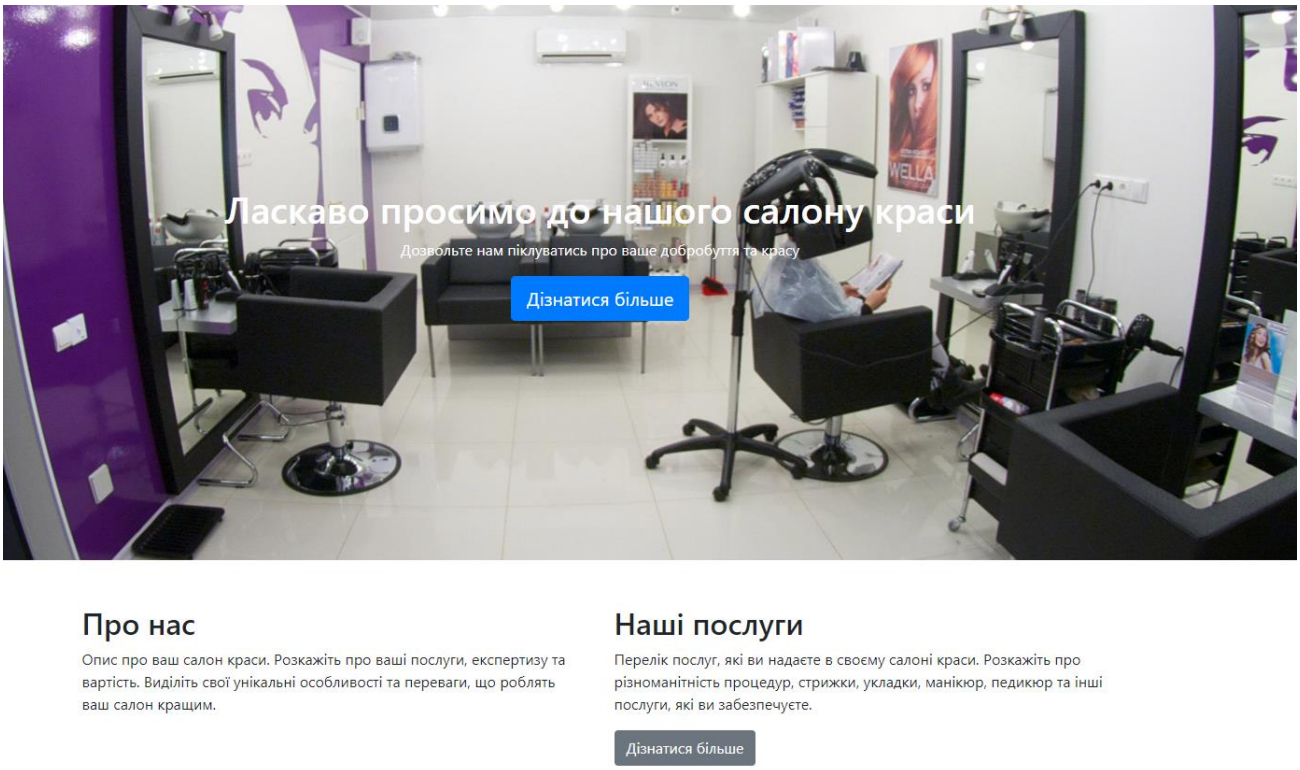


Рисунок 3.15 Головна сторінка з загальною інформацією

- Звертайтесь до адміністратора для отримання додаткової інформації або допомоги.

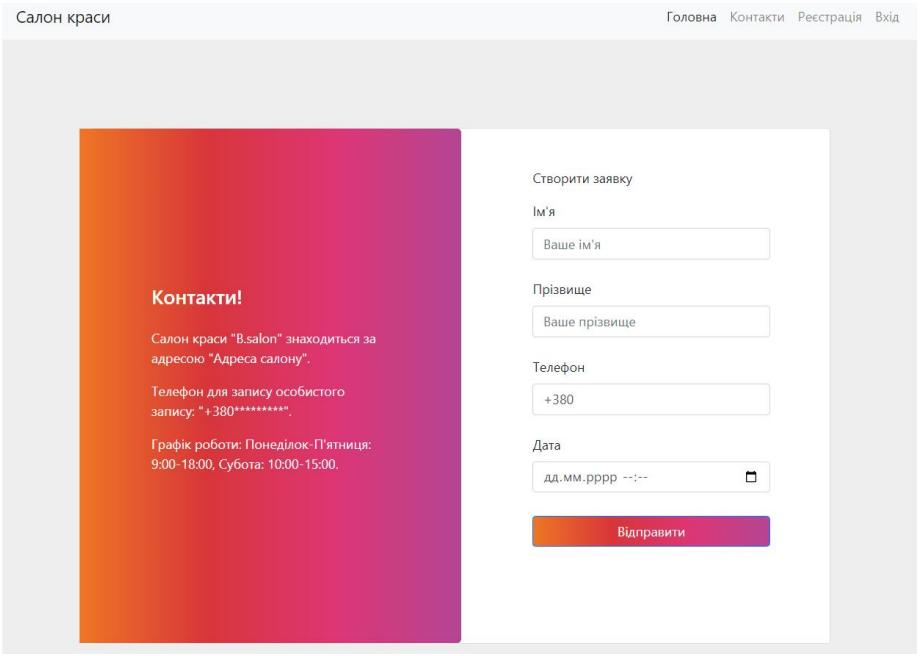


Рисунок 3.16 Сторінка контактів

3.4 Технічні характеристики веб-застосунку

Мінімальні технічні вимоги для пристрою для перегляду веб-застосунку включають:

- **Операційна система**: Пристрій повинен підтримувати сучасні операційні системи, такі як Windows, macOS, Linux, Android або iOS.
- **Веб-браузер**: Рекомендується використовувати останню версію сумісного веб-браузера, такого як Google Chrome, Mozilla Firefox, Safari або Microsoft Edge. Оновлений браузер забезпечить оптимальне відображення та функціональність веб-застосунку.
- **Процесор**: Мінімум двоядерний процесор з тактовою частотою не менше 1.8 ГГц.
- **Оперативна пам'ять (RAM)**: Рекомендується мати не менше 4 ГБ оперативної пам'яті для забезпечення плавної роботи веб-застосунку та уникнення зависань або повільного завантаження сторінок.
- **Дисплей**: За рахунок використання стилів бібліотеки Bootstrap додаток гарно масштабується для інших платформ.
- **Інтернет-підключення**: Необхідне стабільне та достатньо швидке Інтернет-підключення для завантаження сторінок, зображень та інших ресурсів веб-застосунку.
- **Підтримка JavaScript та вбудованих функцій**: Пристрій повинен підтримувати виконання JavaScript та мати активовані вбудовані функції, такі як відтворення звуку та відео, для повного функціоналу веб-застосунку.

Зазначені вимоги допоможуть забезпечити відповідну працездатність та відображення веб-застосунку на різних пристроях.

3.5 Розгортання та встановлення системи

Для успішного розгортання цього проєкту необхідно мати певні компоненти встановлені на вашому комп'ютері, зокрема Node.JS. Почніть з відкриття папки проєкту і відкриття двох терміналів. Для запуску застосунку виконайте наступні кроки:

- Перейдіть до папки з файлами застосунку в терміналі за допомогою команди "cd <dirname>". Де dirname – це шлях до папки застосунку.
- Виконайте команду "npm install" для встановлення залежностей проєкту.
- Запустіть команду "nodemon index.js" для запуску серверної частини.

Для перегляду клієнтської частини перейдіть за посиланням: localhost:3000/

3.6 Тестування веб-застосунку

Під час тестування Node.js застосунків використовуються різні підходи та інструменти. Ось декілька популярних підходів:

1. **Unit-тестування**: Це тестування окремих функцій, модулів або класів програми для перевірки правильності їхньої роботи. Для цього використовуються фреймворки, такі як Mocha, Jest або NodeUnit.
2. **Інтеграційне тестування**: Це тестування взаємодії між різними компонентами або модулями програми для перевірки правильності їхньої взаємодії. Для цього можуть використовуватись фреймворки, такі як Supertest або Chai HTTP.
3. **Функціональне тестування**: Це тестування поведінки програми як цілого, включаючи її функціональні можливості та взаємодію з користувачем або зовнішніми системами. Для функціонального тестування можна використовувати фреймворки, такі як Selenium або Puppeteer.

4. **Тестування API**: Це тестування API застосунку, перевірка правильності його запитів та відповідей. Для цього можна використовувати спеціалізовані бібліотеки, такі як Superagent або Axios.
5. **Тестування навантаження**: Це тестування реакції системи на великі навантаження або стресові умови. Для цього використовуються інструменти, такі як Apache JMeter або Artillery.
6. **Тестування безпеки**: Це тестування вразливостей та захисту додатку. Для цього використовуються інструменти, такі як OWASP ZAP або Burp Suite.

Найкращий підхід до тестування залежить від конкретного застосунку та його вимог. Зазвичай комбінація декількох підходів та інструментів дозволяє забезпечити широкий охоплення тестами та виявити різноманітні типи проблем у застосунку.

Для тестування застосунку було вибрано Unit тестування

Модульне тестування є підходом, при якому окремі модулі або компоненти програми тестуються незалежно від інших частин системи. У цьому випадку, окремі функції, класи або модулі програми тестуються для переконання в їх коректному функціонуванні.

Ці модульні тести використовують бібліотеку **assert** для перевірки очікуваних результатів. Кожен тест перевіряє певний аспект функціонування системи та порівнює його з очікуваним результатом за допомогою функцій **assert.strictEqual** для забезпечення, що значення співпадають.

Отже, код наведений вище є прикладом модульних тестів, які використовуються для тестування окремих модулів або компонентів програми

Деякі приклади тестів:

```
it('should add an employee to the salon', () => {
   const beautySalon = new BeautySalon();
   beautySalon.addEmployee(employee);

   assert.strictEqual(beautySalon.employees.length, 1);
   assert.strictEqual(beautySalon.employees[0], employee);
  });
```

```
  it('should remove an employee from the salon', () => {
    const beautySalon = new BeautySalon();
    beautySalon.addEmployee(employee);
    beautySalon.removeEmployee(employee);

    assert.strictEqual(beautySalon.employees.length, 0);
  });

  it('should add a service to the salon', () => {
    const beautySalon = new BeautySalon();
    beautySalon.addService(service);

    assert.strictEqual(beautySalon.services.length, 1);
    assert.strictEqual(beautySalon.services[0], service);
  });
```

1. **should add an employee to the salon**: Перевіряє, чи можна успішно додати працівника до салону. Після додавання працівника перевіряється, чи збігається кількість працівників в салоні та чи дорівнює перший працівник доданому працівнику.
2. **should remove an employee from the salon**: Перевіряє, чи можна успішно видалити працівника з салону. Після видалення працівника перевіряється, чи кількість працівників в салоні дорівнює нулю.
3. **should add a service to the salon**: Перевіряє, чи можна успішно додати послугу до салону. Після додавання послуги перевіряється, чи збігається кількість послуг в салоні та чи дорівнює перша послуга доданій послузі.
4. **should remove a service from the salon**: Перевіряє, чи можна успішно видалити послугу з салону. Після видалення послуги перевіряється, чи кількість послуг в салоні дорівнює нулю.
5. **should schedule an appointment**: Перевіряє, чи можна успішно запланувати запис на прийом. Після запланування запису перевіряється, чи збігається кількість записів в салоні та чи дорівнює перший запис доданому запису.

6. **should cancel an appointment**: Перевіряє, чи можна успішно скасувати запис на прийом. Після скасування запису перевіряється, чи кількість записів в салоні дорівнює нулю.
7. **should get the schedule of an employee**: Перевіряє, чи можна успішно отримати розклад роботи працівника. Після запланування декількох записів перевіряється, чи збігається кількість записів у розкладі та чи дорівнюють записи в розкладі доданим записам.

Ці тести допомагають забезпечити, що ваш застосунок "Beauty Salon" працює правильно та виконує очікувані операції з працівниками, послугами, клієнтами та записами на прийоми

3.7 Висновки програмної реалізації застосунку

Висновки

Під час розробки веб-застосунку для автоматизації роботи салону краси та організації роботи з клієнтами були виявлені кілька висновків. Незважаючи на деякі складнощі, програмна реалізація додатку була успішно виконана в повному обсязі, а також має потенціал для масштабування та додавання нового функціоналу в майбутньому.

Під час розробки була створена база даних та реалізована можливість авторизації користувача. Код для серверної та клієнтської частини додатку був розроблений, а також були впроваджені різні клієнтські анімації. Архітектура проєкту була створена з урахуванням вимог ефективності та розширюваності.

У розділі тестування застосунку були створені тестові набори, а також були написані юніт-тести. Це допомогло покращити безпеку та функціональність застосунку, а також створило базу для подальшого розширення та додавання нового функціоналу.

В цілому, розроблений веб-застосунок для автоматизації роботи салону краси та організації роботи з клієнтами є успішною програмною реалізацією з перспективами для подальшого розвитку та розширення.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було проведено дослідження предметної області, а також обґрунтована актуальність розробки веб-застосунку для автоматизації роботи салону краси та поліпшення взаємодії з клієнтами. Виявлено проблеми, з якими зіштовхуються люди, що не використовують веб-додатки у сфері краси, та визначено причини цих проблем.

На основі аналізу аналогічних застосунків було складено список їх переваг і недоліків, а також сформульовані функціональні та нефункціональні вимоги до розроблюваного веб-застосунку. З цих вимог було сформульовано технічне завдання та розроблено сценарії використання додатку.

У рамках кваліфікаційної роботи був розроблений веб-застосунок для автоматизації роботи салону краси та поліпшення взаємодії з клієнтами. Застосунок надає можливість клієнтам оглядати послуги, записуватись на процедури, здійснювати оплату та зв'язок з майстрами салону.

Для реалізації проекту було використано сучасні технології, зокрема фреймворк Angular для фронтенду та Node.js з використанням Express.js для бекенду. В якості бази даних була обрана MongoDB для зберігання інформації про послуги, майстрів та клієнтів.

Веб-застосунок має зручний та інтуїтивно зрозумілий інтерфейс, що дозволяє клієнтам швидко знайти потрібну інформацію, записатись на процедуру та здійснити оплату. Дизайн та взаємодія додатку були розроблені з урахуванням принципів зручності та задоволення користувачів.

В процесі розробки були проведені тести різних функціональних модулів, таких як запис на процедуру, оплата та комунікація з майстрами, для забезпечення якості та надійності програмного забезпечення.

Результатом кваліфікаційної роботи є функціональний веб-застосунок для автоматизації роботи салону краси та поліпшення взаємодії з клієнтами. Проект має потенціал для подальшого розширення, зокрема шляхом додавання

функціоналу аутентифікації користувачів, системи відгуків та оцінок, а також інтеграції з платіжними системами для онлайн-оплати.

Загалом, розроблений веб-застосунок є ефективним інструментом для автоматизації роботи салону краси та поліпшення взаємодії з клієнтами. Використання сучасних технологій розробки та увага до деталей дизайну дозволили створити продукт, який відповідає потребам салону та забезпечує зручне та задоволене використання для клієнтів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Макконнелл, С. (2004). "Code Complete: Практичний посібник з розробки програмного забезпечення." Київ: Видавництво "Дія".
2. Галицький, П., & Микуляк, А. (2015). "Node.js: Повний курс." Київ: Видавництво "Дія".
3. Заставний, А., & Козловський, О. (2018). "HTML та CSS: Дизайн і розробка веб-сайтів." Київ: Видавництво "Лоріс".
4. Абрамян, М., & Кузьмін, О. (2017). "MongoDB: Розробка і адміністрування." Київ: Видавництво "Пакт".
5. Фаулер, М. (2006). "Мова опису шаблонів UML 2.0." Київ: Видавництво "БІОСФЕРА".
6. Фаранда, І., & Калініченко, Є. (2016). "Програмування на JavaScript." Київ: Видавництво "Нова Книга".
7. Ковальов, А., & Красюк, О. (2019). "Веб-дизайн та розробка на HTML і CSS." Київ: Видавництво "Видавець".
8. Каплан, А., & Лерман, Г. (2007). "JavaScript і AJAX: Інтерактивні веб-додатки." Київ: Видавництво "Пакт".
9. Митник, О. (2018). "Front-End дизайн: Як створювати модерні веб-сайти." Київ: Видавництво "Інфопрес".
10. Мельник, С. (2017). "Розробка веб-додатків з використанням Node.js і MongoDB." Київ: Видавництво "Альтерпрес".
11. Шевченко, Д. (2015). "JavaScript для професіоналів." Київ: Видавництво "Діалектика".
12. Адамская, Л. (2016). "Web-дизайн: HTML, CSS, JavaScript." Київ: Видавництво "Дія".
13. Волкова, В. (2019). "Повний курс з MongoDB: від початку до високої доступності." Київ: Видавництво "Пакт".
14. Муравйова, О. (2018). "Front-End. З досвіду розробки." Київ: Видавництво "Лоріс".

15. Погорєлов, А., & Грушевський, І. (2015). "Node.js. Путівник по технології." Київ: Видавництво "Пакт".
16. Зайцев, О. (2017). "MongoDB: Повноцінна розробка веб-додатків." Київ: Видавництво "Пакт".
17. MDN Web Docs [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ (дата звернення – 15.02.2023).
18. W3Schools [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/ (дата звернення – 05.03.2023).
19. CSS-Tricks [Електронний ресурс] – Режим доступу до ресурсу: https://css-tricks.com/ (дата звернення – 21.01.2023).
20. Stack Overflow [Електронний ресурс] – Режим доступу до ресурсу: https://stackoverflow.com/ (дата звернення – 10.04.2023).
21. Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/ (дата звернення – 02.02.2023).
22. MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/ (дата звернення – 18.03.2023).
23. GitHub [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/ (дата звернення – 27.01.2023).
24. Codecademy [Електронний ресурс] – Режим доступу до ресурсу: https://www.codecademy.com/ (дата звернення – 09.04.2023).
25. Smashing Magazine [Електронний ресурс] – Режим доступу до ресурсу: https://www.smashingmagazine.com/ (дата звернення – 08.02.2023).
26. A List Apart [Електронний ресурс] – Режим доступу до ресурсу: https://alistapart.com/ (дата звернення – 14.03.2023).
27. Flanagan, D. (2011). JavaScript: The Definitive Guide. O'Reilly Media.
28. Duckett, J. (2014). HTML & CSS: Design and Build Websites. Wiley.
29. Duckett, J. (2014). JavaScript & jQuery: Interactive Front-End Web Development. Wiley.

30. Robbins, J., & Beech, J. (2012). Learning Node: Moving to the Server-Side. O'Reilly Media.
31. Wilson, J., & Wynne, R. (2013). Node.js the Right Way: Practical, Server-Side JavaScript That Scales. Pragmatic Bookshelf.
32. Dirolf, A., & Banker, D. (2013). MongoDB in Action. Manning Publications.
33. Copeland, M. (2010). MongoDB Applied Design Patterns. Packt Publishing.
34. Chodorow, K. (2010). MongoDB: The Definitive Guide. O'Reilly Media.
35. Peco, G. (2017). Mastering MongoDB 3.x: An expert's guide to building fault-tolerant MongoDB applications. Packt Publishing
36. Freeman, A., & Robson, E. (2017). Head First HTML and CSS: A Learner's Guide to Creating Standards-Based Web Pages. O'Reilly Media.
37. Duckett, J. (2011). HTML & CSS: Design and Build Websites. Wiley.
38. Powell, T. A., & Schneider, M. (2017). HTML5: The Complete Reference. McGraw-Hill Education.
39. Cederholm, D. (2010). CSS3 for Web Designers. A Book Apart.
40. Meyer, E. A. (2017). CSS: The Definitive Guide. O'Reilly Media.
41. Gamble, I. (2018). The Node.js Handbook. Flavio Copes.
42. Cantelon, M., & others. (2019). Node.js 8 the Right Way: Practical, Server-Side JavaScript That Scales. Pragmatic Bookshelf.
43. Hawke, B. (2017). MongoDB Basics. CreateSpace Independent Publishing Platform.
44. Plugge, E., Hawkins, P., & Membrey, P. (2016). The Definitive Guide to MongoDB: A Complete Guide to Dealing with Big Data Using MongoDB. Apress.
45. Dirolf, A. (2011). The Little MongoDB Book. Self-published.
46. Valeri, V. (2014). MongoDB Applied Design Patterns. Packt Publishing.
47. Membrey, P., Plugge, E., & Hawkins, T. (2010). The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing. Apress.

ДОДАТОК А

(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Введення

Робота виконується в рамках проекту розробки «Веб-застосунку для автоматизації роботи салону краси та організації роботи з клієнтами». Технічне завдання розроблено у відповідності до стандарту ГОСТ 19.201–78.

1 Підстава для розробки

Підставою для розробки є «Завдання на дипломний проект», затверджене завідувачем кафедри інженерії програмного забезпечення. Найменування розробки: «Веб-застосунок для автоматизації роботи салону краси та організації роботи з клієнтами»

2 Призначення розробки

2.1 Функціональне призначення

Функціональним призначенням додатку запис на прийом у салон краси та перегляд інформації сайту для звичайного користувача, а також облік замовлень для адміністратора веб-застосунку.

2.2 Експлуатаційне призначення

Програма повинна експлуатуватися на будь-яких пристроях, на яких є браузер. Кінцевим користувачем додатку може виступати будь-яка особа.

3 Вимоги до програми

3.1 Вимоги до функціональних характеристик

Для авторизованих користувачів:

- інформація про послуги, їх ціни та тривалість;
- інформація про наявність вільних місць на певний час;
- історія замовлень та їх статус.

Працівники:

- інформація про замовлення на їх послуги;
- інформація про їх власні прийоми;

- доступ до інформації про клієнтів та їхні замовлення.

В залежності від посади функціонал працівника може розширюватись такими можливостями:

- можливість керування користувачами в системі;
- можливість налаштування параметрів системи та її оновлення.

Для неавторизованих користувачів:

- доступ до основної інформації про салон краси;
- Реєстрація в системі.
- Авторизація

3.2 Вимоги до надійності

Веб-додаток повинен забезпечувати такі вимоги до надійності:

– обробляти невірні дії користувача і попереджати його про можливі наслідки;

– можливість самостійно відновлюватись у разі збою;

– можливість резервного копіювання бази даних.

3.3 Умови експлуатації та вимоги до технічних засобів

Веб-додаток повинен працювати на всіх пристроях, які мають браузер та стабільний доступ до мережі «Інтернет»: смартфонах, планшетах та комп’ютерах. Браузер може бути будь-яким: Safari, GoogleChrome, Opera, MozillaFirefox, тощо.

Для роботи платформи без збоїв, потрібно, щоб пристрій, на якому вона запускається задовольняв наступні мінімальні вимоги:

– 512 Мб внутрішньої пам’яті;

– 2 Гб оперативної пам’яті;

– 2-ядерний процесор;

– доступ до мережі «Інтернет» зі швидкістю мінімум5Мбіт/с.

3.4 Вимоги до інформаційної та програмної сумісності

При розробцівеб-додатку буде використовуватись високорівнева об’єктно-орієнтована мова JavaScript. Для розробки серверної частини використовуватиметься платформа Node.js та фреймворк Express. Для розробки клієнтської частини була використана бібліотека стилів Bootstrap. В якості СКБД буде використовуватись нереляційна база даних MongoDB.

3.5 Спеціальні вимоги

Програма повинна мати зручний, гарний та зрозумілий зовнішній інтерфейс користувача.

4 Вимоги до програмної документації

У момент здачі проекту замовнику надається наступний набір документів:

– текст програми;
– опис програми;
– технічне завдання;
– керівництво користувача.

5 Стадії та етапи розробки

Стадії та етапи розробки інтернет-платформи «Агентство подорожей»подані у таблиці А.1.

Таблиця А.1 – Стадії та етапи розробки проекту

Стадія розробки	Етапи робіт	Зміст робіт
1	2	3
Технічне завдання 02.01.22 – 31.01.22	Обґрунтування необхідності розробки програми	Коротка характеристика програмного забезпечення; підстава і призначення розробки; вимоги до програмної системи і документація; стадії і етапи розробки програми; порядок контролю і приймання

Ескізний проект 01.02.23 – 26.02.23	Розробка ескізного проекту	Попередня розробка структури вхідних і вихідних даних; уточнення середовища програмування; розробка і опис загальної алгоритмічної структури
Технічний проект 29.02.23 – 19.03.23	Розробка технічного проекту	Уточнення структури вхідних і вихідних даних; розробка докладного алгоритму; розробка структури програми
Робочий проект 20.03.23 – 15.04.23	Розробка програмного забезпечення	Реалізація програмного забезпечення; відлагодження; проведення попереднього тестування
Розробка програмної документації 16.04.23 – 22.04.23	Розробка документації до програмного забезпечення	Розробка необхідної документації, передбаченої технічним завданням
Тестування системи 23.04.23 – 30.04.23	Проведення тестування програмного забезпечення	Розробка методики тестування; проведення основних тестів; коректування програмного забезпечення
Впровадження	Підготовка і передача програми	Підготовка і розгортання програмного забезпечення

6 Порядок контролю та приймання

Контроль здійснюється кінцевими користувачами системи, підключеними на етапі тестування додатку.

ДОДАТОК Б

(обов’язковий)

ДІАГРАМИ

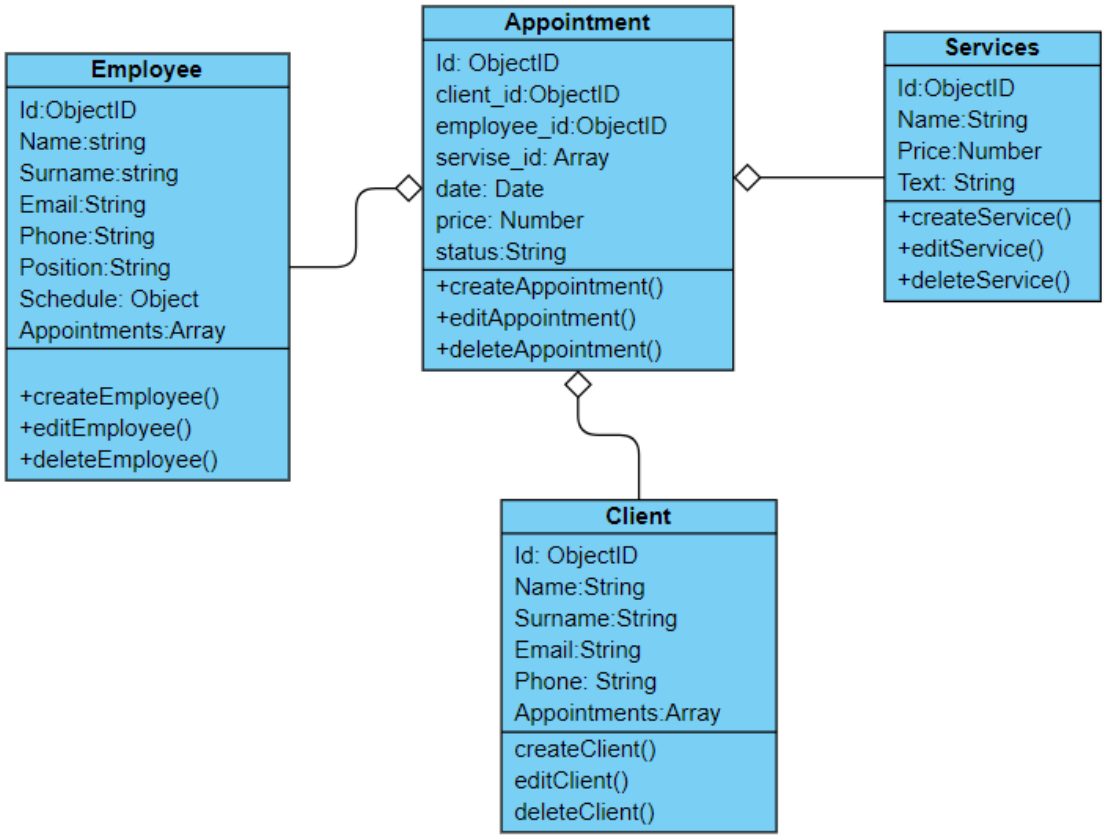


Рисунок Б.1 - Схема бази даних

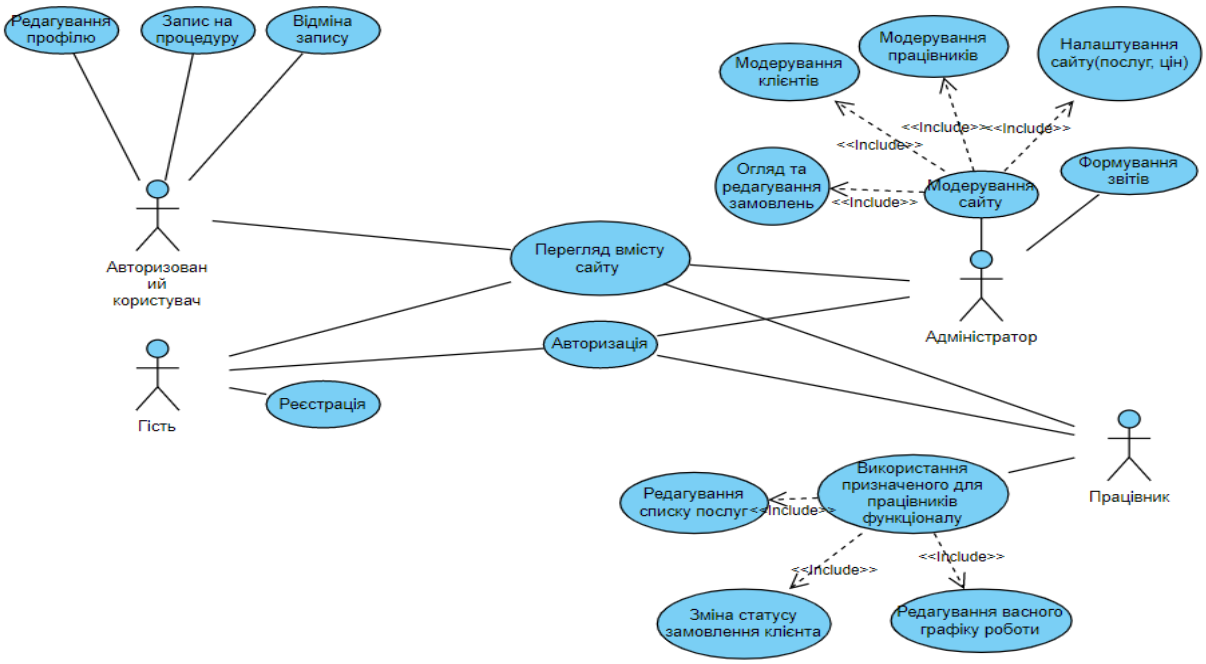


Рисунок Б.2 - Діаграма варіантів використання

ДОДАТОК В

(обов’язковий)

КОД (ЛІСТИНГ) ПРОГРАМИ

index.js

```
const express = require('express');

const app = express();

const session = require('express-session');

const bodyParser = require('body-parser');

const port = 3000;

const passport = require('passport');

const faker = require('faker');

// Other app configurations

const { loginCheck } = require("./auth/passport");

app.use(passport.initialize());

loginCheck(passport);


app.use(session({

  secret: 'oneboy',

  saveUninitialized: true,

  resave: true

}));

app.set('view engine', 'ejs');

// Parse JSON bodies

app.use(bodyParser.json());


// Parse URL-encoded bodies

app.use(bodyParser.urlencoded({ extended: true }));

const connectDB = require('./db');

connectDB();

app.set('views', __dirname + '/views');

const mainRouter = require('./routes/main');

app.use(passport.session());
```

```
app.use('/', mainRouter);


app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

product.js

```
const express = require('express');
const router = express.Router();
const { Product } = require("../models/Product");
const multer = require('multer');

const { auth } = require("../middleware/auth");

var storage = multer.diskStorage({
    destination: (req, file, cb) => {
        cb(null, 'uploads/')
    },
    filename: (req, file, cb) => {
        cb(null, `${Date.now()}_${file.originalname}`)
    },
    fileFilter: (req, file, cb) => {
        const ext = path.extname(file.originalname)
        if (ext !== '.jpg' || ext !== '.png') {
            return cb(res.status(400).end('only jpg, png are allowed'), false);
        }
        cb(null, true)
    }
})
```

```
var upload = multer({ storage: storage }).single("file")

router.post("/uploadImage", auth, (req, res) => {

    upload(req, res, err => {
        if (err) {
            return res.json({ success: false, err })
        }
        return res.json({ success: true, image: res.req.file.path, fileName: res.req.file.filename })
    })

});

router.post("/uploadProduct", auth, (req, res) => {

    const product = new Product(req.body)

    product.save((err) => {
        if (err) return res.status(400).json({ success: false, err })
        return res.status(200).json({ success: true })
    })

});

router.post("/getProducts", (req, res) => {

    let order = req.body.order ? req.body.order : "desc";
    let sortBy = req.body.sortBy ? req.body.sortBy : "_id";
    let limit = req.body.limit ? parseInt(req.body.limit) : 100;
    let skip = parseInt(req.body.skip);

    let findArgs = {};
    let term = req.body.searchTerm;
```

```
for (let key in req.body.filters) {

    if (req.body.filters[key].length > 0) {
        if (key === "price") {
            findArgs[key] = {
                $gte: req.body.filters[key][0],
                $lte: req.body.filters[key][1]
            }
        } else {
            findArgs[key] = req.body.filters[key];
        }
    }
}

console.log(findArgs)

if (term) {
    Product.find(findArgs)
        .find({ $text: { $search: `\"${term}\"` } })
        .populate("writer")
        .sort([[sortBy, order]])
        .skip(skip)
        .limit(limit)
        .exec((err, products) => {
            if (err) return res.status(400).json({ success: false, err })
            res.status(200).json({ success: true, products, postSize: products.length })
        })
} else {
    Product.find(findArgs)
        .populate("writer")
        .sort([[sortBy, order]])
        .skip(skip)
```

```
            .limit(limit)
            .exec((err, products) => {
                if (err) return res.status(400).json({ success: false, err })
                res.status(200).json({ success: true, products, postSize: products.length })
            })
    }

});

router.get("/products_by_id", (req, res) => {
    let type = req.query.type
    let productIds = req.query.id

    console.log("req.query.id", req.query.id)

    if (type === "array") {
        let ids = req.query.id.split(',');
        productIds = [];
        productIds = ids.map(item => {
            return item
        })
    }

    console.log("productIds", productIds)

    Product.find({ '_id': { $in: productIds } })
        .populate('writer')
        .exec((err, product) => {
            if (err) return res.status(400).send(err)
            return res.status(200).send(product)
        })
});
```

```
module.exports = router;
```

main.js

```
const express = require('express');

const router = express.Router();

const faker = require('faker');

// Import the necessary controllers

const {  registerView, loginView,  registerUser,  loginUser} = require('../controllers/loginController');

const {clientView,  clients} = require('../controllers/clientController')

const {getAllServices} = require('../controllers/serviceController')

const {generateAppoint} = require('../controllers/appointmentController')

const {generateScheduleData} = require('../controllers/employeeController')

const { protectRoute, allowIf } = require('../auth/protect');

router.get('/', (req, res) => {

  const authenticated = req.isAuthenticated();

  res.render("home", { authenticated });

});

router.get('/contacts', (req, res) => {

  const authenticated = req.isAuthenticated();

  res.render("contacts", { authenticated });

});

router.get('/services', (req, res) => {

       res.render("services" );

  });

router.get('/register', allowIf, registerView);

router.post('/register', allowIf, registerUser);

router.get('/login', allowIf, loginView);

router.post('/login', allowIf, loginUser);
```

```
router.get('/client', protectRoute, clientView);

//router.post('/client/routerointments', protectRoute, createrouterointment);

router.get('/admin', (req, res) => {
  res.render('admin/admin', { title: 'Панель керування адміністратора' });
});


// Сторінка клієнтів
router.get('/admin/clients', (req, res) => {


  res.render('admin/clients', { title: 'Клієнти', clients });
});
// Сторінка працівників
router.get('/admin/employees', (req, res) => {
    // Генерування випадкових даних працівників
      res.render('admin/employees', { title: 'Адміністратор - Працівники', employees });
  });
// Сторінка прийомів
router.get('/admin/appointments', (req, res) => {
    // Генерування випадкових даних прийомів
      res.render('admin/appointments', { title: 'Прийоми', generateAppoint});
  });
// Сторінка послуг
router.get('/admin/services', (req, res) => {
  // Генерування випадкових даних послуг
   res.render('admin/services', { title: 'Послуги', getAllServices });
});
// Сторінка працівників
router.get('/employee', (req, res) => {
  const employees = generateScheduleData();
```

```
  res.render('employee/employee', { title: 'Адміністратор - Працівники', generateScheduleData, breaks: [] });
});
// ...
module.exports = router;
```

appointment.js

```
const mongoose = require('mongoose');

const appointmentSchema = new mongoose.Schema({
  client_id: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Client',
    required: true,
  },
  employee_id: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Employee',
    required: true,
  },
  service_id: [
    {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Service',
      required: true,
    },
  ],
  date: {
    type: Date,
    required: true,
  },
```

```
  status: {
    type: String,
    required: true,
  },
});

const Appointment = mongoose.model('Appointment', appointmentSchema);

module.exports = Appointment;
```

client.js

```
const mongoose = require('mongoose');

const clientSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  surname: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
  },
  phone: {
    type: String,
    required: true,
  },
```

```
  password:{
    type: String,
    required: true,
  },
  appointments: [
    {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Order',
    },
  ],
});

const Client = mongoose.model('Client', clientSchema);

module.exports = Client;
```

employee.js

```
const mongoose = require('mongoose');

const employeeSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  surname: {
    type: String,
    required: true,
  },
  email: {
    type: String,
```

```
  required: true,
},
phone: {
  type: String,
  required: true,
},
position: {
  type: String,
  required: true,
},
schedule: {
  type: Object,
  required: true,
  properties: {
    workDays: {
      type: Array,
      required: true,
      items: {
        type: String,
      },
    },
    workHours: {
      type: Array,
      required: true,
      items: {
        type: Object,
        properties: {
          startTime: {
            type: String,
            required: true,
```

```
      },
      endTime: {
        type: String,
        required: true,
      },
    },
  },
},
breaks: {
  type: Array,
  items: {
    type: Object,
    properties: {
      startTime: {
        type: String,
        required: true,
      },
      endTime: {
        type: String,
        required: true,
      },
    },
  },
},
availability: {
  type: Array,
  items: {
    type: Object,
    properties: {
      date: {
```

```
            type: String,
            required: true,
          },
          startTime: {
            type: String,
            required: true,
          },
          endTime: {
            type: String,
            required: true,
          },
          booked: {
            type: Boolean,
            required: true,
          },
        },
      },
    },
  },
  appointments: [
    {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Appointment', // Replace 'Order' with the correct model name for appointments
    },
  ],
});

const Employee = mongoose.model('Employee', employeeSchema);
```

```
module.exports = Employee;
```

passport.js

```
const bcrypt = require("bcryptjs");

LocalStrategy = require("passport-local").Strategy;

// Завантаження моделей

const Employee = require("../models/employee");

const Client = require("../models/client");


const loginCheck = passport => {

  passport.use(

    new LocalStrategy({ usernameField: "email" }, (email, password, done) => {

      // Перевірка співробітника

      Employee.findOne({ email: email })

        .then(user => {

          if (user) {

            // Порівняння паролів

            bcrypt.compare(password, user.password, (error, isMatch) => {

              if (error) throw error;

              if (isMatch) {

                return done(null, { ...user._doc, userType: "employee" });

              } else {

                console.log("Неправильний пароль");

                return done(null, false);

              }

            });

          } else {

            // Перевірка адміністратора

            Employee.findOne({ email: email })
```

```
.then(user => {
  if (user) {
    // Порівняння паролів
    bcrypt.compare(password, user.password, (error, isMatch) => {
      if (error) throw error;
      if (isMatch) {
        return done(null, { ...user._doc, userType: "administrator" });
      } else {
        console.log("Неправильний пароль");
        return done(null, false);
      }
    });
  } else {
    // Перевірка клієнта
    Client.findOne({ email: email })
      .then(user => {
        if (user) {
          // Порівняння паролів
          bcrypt.compare(password, user.password, (error, isMatch) => {
            if (error) throw error;
            if (isMatch) {
              return done(null, { ...user._doc, userType: "client" });
            } else {
              console.log("Неправильний пароль");
              return done(null, false);
            }
          });
        } else {
          console.log("Користувач не знайдений");
          return done(null, false);
```

```
            }
          })
          .catch(error => console.log(error));
        }
      })
      .catch(error => console.log(error));
  })
);

passport.serializeUser((user, done) => {

  done(null, { id: user.id, userType: user.userType });
});

passport.deserializeUser(async (serializedUser, done) => {
  const { id, userType } = serializedUser;

  let model;

  switch (userType) {
    case "employee":
      model = Employee;
      break;
    case "administrator":
      model = Administrator;
      break;
    case "client":
```

```
        model = Client;
        break;
      default:
        return done("Недійсний тип користувача");
    }

    try {
      const user = await model.findOne({ _id: id }).exec();
      done(null, user);
    } catch (error) {
      done(error, null);
    }
  });
};

module.exports = {
  loginCheck,
};
```

protect.js

```
const protectRoute = (req, res, next) => {
    if (req.isAuthenticated()) {
      const userType = req.user.userType;
      let redirectUrl;

      switch (userType) {
        case "employee":
          redirectUrl = "/employee";
          break;
```

```
      case "administrator":
        redirectUrl = "/admin";
        break;
      case "client":
        redirectUrl = "/client";
        break;
      default:
        return res.redirect("/login");
    }

    return res.redirect(redirectUrl);
  }

  console.log("Будь ласка, увійдіть, щоб продовжити");
  res.redirect("/login");
};

const allowIf = (req, res, next) => {
  if (!req.isAuthenticated()) {
    return next();
  }

  const userType = req.user.userType;
  let redirectUrl;

  switch (userType) {
    case "employee":
      redirectUrl = "/employee";
      break;
    case "administrator":
```

```
        redirectUrl = "/admin";
        break;
      case "client":
        redirectUrl = "/client";
        break;
      default:
        return res.redirect("/client");
    }

    res.redirect(redirectUrl);
  };

  module.exports = {
    protectRoute,
    allowIf,
  };
```

loginController.js

```
const Client = require("../models/client");
const bcrypt = require("bcryptjs");
const passport = require("passport"); // Require passport module

const registerView = (req, res) => {
  res.render("register.ejs");
};

const loginView = (req, res) => {
  res.render("login");
};
```

```
const loginUser = (req, res, next) => {
    const { email, password } = req.body;

    if (!email || !password) {
      console.log("Please fill in all the fields");
      res.render("login", { email, password });
    } else {
      passport.authenticate("local", (error, user, info) => {
      // Authentication code...

      req.logIn(user, (error) => {
        if (error) {
          console.log(error);
          return next(error);
        }

        user.authenticated = true; // Set authenticated to true

        const { userType } = req.user;

        let redirectUrl;

        switch (userType) {
          case "employee":
            redirectUrl = "/employee";
            break;
          case "administrator":
            redirectUrl = "/admin";
            break;
          case "client":
            req.session.clientData = {
              name: user.name,
```

```
            surname: user.surname,
            email: user.email,
            phone: user.phone,
            appointments: user.appointments,
          };

          redirectUrl = "/client";
          break;
        default:
          return res.redirect("/login");
      }

      res.redirect(redirectUrl);
    });
  })(req, res, next);
 }
};

//Post Request that handles Register
const registerUser = (req, res) => {
 const { name, surname, email, phone, password, confirm } = req.body;
 console.log({ name, surname, email, phone, password, confirm });
 if (!name || !email || !password || !confirm) {
  console.log("Fill empty fields");
 }
 //Confirm Passwords
 if (password !== confirm) {
  console.log("Password must match");
 } else {
  //Validation
  Client.findOne({ email: email }).then((user) => {
   if (user) {
    console.log("email exists");
```

```
        res.render("register", {
          name,
          email,
          password,
          confirm,
        });
      } else {
        //Validation
        const newUser = new Client({
          name,
          surname,
          email,
          phone,
          password,
        });
        //Password Hashing
        bcrypt.genSalt(10, (err, salt) =>
          bcrypt.hash(newUser.password, salt, (err, hash) => {
            if (err) throw err;
            newUser.password = hash;
            newUser
              .save()
              .then(res.redirect("/login"))
              .catch((err) => console.log(err));
          })
        );
      }
    });
  }
};

module.exports = {
  registerView,
```

```
  registerUser,
  loginUser,
  loginView,
};
```

clientController.js

```
const Appointment = require('../models/appointment');
const Employee = require('../models/employee');

// Функція створення замовлення
const createAppointment = async (req, res) => {
  try {
    const { clientId, employeeId, serviceId, date } = req.body;

    // Перевірка доступності працівника в обраний час
    const isEmployeeAvailable = await Employee.findOne({
      _id: employeeId,
      appointments: {
        $elemMatch: { date: new Date(date) },
      },
    });

    if (isEmployeeAvailable) {
      res.status(400).send('Обраний працівник не доступний у вказаний час');
      return;
    }

    // Створення замовлення
```

```
    const appointment = new Appointment({
      client_id: clientId,
      employee_id: employeeId,
      service_id: serviceId,
      date: new Date(date),
      status: 'pending',
    });

    // Збереження замовлення
    await appointment.save();

    res.redirect('/client');
  } catch (error) {
    console.log(error);
    res.redirect('/client');
  }
};

const clientView = (req, res) => {
    const clientData = req.session.clientData;

    if (!clientData) {
      console.log("Undefined client");
      return res.redirect("/login");
    }
    res.render("clientView", { clientData });
  };
const clients = (req, res) => {
   // Генерування випадкових даних клієнтів
   const client = [];
```

```
  for (let i = 0; i < 20; i++) {
    const cliento = {
      id: faker.random.uuid(),
      name: faker.name.firstName(),
      surname: faker.name.lastName(),
      email: faker.internet.email(),
      phone: generateRandomPhoneNumber(),
    };
    clients.push(cliento);

  }
  const page = parseInt(req.query.page) || 1;
  const limit = 5; // Кількість клієнтів на сторінку

  // Генерування випадкових даних клієнтів
const startIndex = (page - 1) * limit;
const endIndex = page * limit;

const paginatedClients = clients.slice(startIndex, endIndex);




  // Визначення параметрів для кнопок перемикання
  const totalPages = Math.ceil(clients.length / limit);
  const nextPage = page < totalPages ? page + 1 : null;
  const prevPage = page > 1 ? page - 1 : null;
  return {client: paginatedClients, nextPage, prevPage }
}
```

```
module.exports = {
  createAppointment,
  clientView,
  clients,
};
```

serivceController.js

```
// Підключення необхідних модулів та залежностей
const Service = require('../models/Service'); // Приклад моделі Service


// Оголошення та експорт об'єкта контролера
const serviceController = {
  // Метод для отримання всіх послуг
  getAllServices: async (req, res) => {
    try {
      // Отримання всіх послуг з бази даних
      const services = await Service.find();


      // Відправлення списку послуг у відповідь
      res.json(services);
    } catch (error) {
      console.error(error);
      res.status(500).json({ message: 'Помилка сервера' });
    }
  },
getAllServices: async (req, res) => {
  const services = [];
  for (let i = 0; i < 4; i++) {
    const service = {
```

```
    id: faker.random.uuid(),
    name: faker.commerce.productName(),
    price: faker.commerce.price(),
    description: faker.lorem.sentence(),
  };
  services.push(service);
}
return services
},
// Метод для створення нової послуги
createService: async (req, res) => {
  try {
    // Отримання даних з тіла запиту
    const { name, description, price } = req.body;

    // Створення нового об'єкту послуги
    const newService = new Service({
      name,
      description,
      price,
    });

    // Збереження нової послуги в базі даних
    const savedService = await newService.save();

    // Відправлення збереженої послуги у відповідь
    res.json(savedService);
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Помилка сервера' });
```

```
    }
  },

  // Інші методи контролера (оновлення, видалення послуг і т.д.)
};

module.exports = serviceController;
```

employeeController.js

```
const faker = require('faker');

function generateRandomPhoneNumber() {
  const digits = '0123456789';
  let phoneNumber = '(';

  for (let i = 0; i < 3; i++) {
    phoneNumber += digits[Math.floor(Math.random() * 10)];
  }

  phoneNumber += ') ';

  for (let i = 0; i < 3; i++) {
    phoneNumber += digits[Math.floor(Math.random() * 10)];
  }

  phoneNumber += '-';

  for (let i = 0; i < 4; i++) {
    phoneNumber += digits[Math.floor(Math.random() * 10)];
  }
```

```
  return phoneNumber;
}

function generateScheduleData() {
    const Employee = require('./models/Employee');

    async function generateScheduleData() {
      try {
        const employees = await Employee.find().limit(5); // Fetching only 5 employees

        return employees;
      } catch (error) {
        console.error('Error fetching employees:', error);
        return [];
      }
    }


  return employees;
}

function clientView(req, res) {
  res.render('client', { title: 'Client View' });
}

function createAppointment(req, res) {
  // Handle creating an appointment
}
```

```
module.exports = {
  generateScheduleData,
  clientView,
  createAppointment,
};
```

Service.js

```
const mongoose = require('mongoose');

const serviceSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  price: {
    type: Number,
    required: true,
  },
  description: {
    type: String,
    required: true,
  },
});

const Service = mongoose.model('Service', serviceSchema);

module.exports = Service;
```

db.js

```
const mongoose = require('mongoose');


const connectDB = async () => {
try {
    await mongoose.connect('mongodb+srv://dan:qwe1@cluster0.qjrtmiy.mongodb.net/Bsalon?retryWrites=true&w=majority', {
    useNewUrlParser: true,
    useUnifiedTopology: true
    });
    console.log('Підключено до бази даних');
} catch (error) {
    console.log('Помилка підключення до бази даних:', error);
}
};


module.exports = connectDB;
```

tests.js

```
const assert = require('assert');
const { Employee, Appointment, Service, Client } = require('./your_application_module');


describe('Beauty Salon', () => {
  let employee;
  let service;
  let client;


  beforeEach(() => {
    employee = new Employee('John', 'Doe', 'john@example.com', '1234567890', 'Hair Stylist');
    service = new Service('Haircut',30,'Cutting hair');
    client = new Client('Jane', 'Smith', 'jane@example.com', '0987654321');
```

```
});

it('should add an employee to the salon', () => {
  const beautySalon = new BeautySalon();
  beautySalon.addEmployee(employee);

  assert.strictEqual(beautySalon.employees.length, 1);
  assert.strictEqual(beautySalon.employees[0], employee);
});

it('should remove an employee from the salon', () => {
  const beautySalon = new BeautySalon();
  beautySalon.addEmployee(employee);
  beautySalon.removeEmployee(employee);

  assert.strictEqual(beautySalon.employees.length, 0);
});

it('should add a service to the salon', () => {
  const beautySalon = new BeautySalon();
  beautySalon.addService(service);

  assert.strictEqual(beautySalon.services.length, 1);
  assert.strictEqual(beautySalon.services[0], service);
});

it('should remove a service from the salon', () => {
  const beautySalon = new BeautySalon();
  beautySalon.addService(service);
  beautySalon.removeService(service);
```

```
  assert.strictEqual(beautySalon.services.length, 0);
});

it('should schedule an appointment', () => {
  const beautySalon = new BeautySalon();
  beautySalon.addEmployee(employee);
  beautySalon.addService(service);

  const appointment = new Appointment(employee, client, service, new Date(), '10:00', '11:00');
  beautySalon.scheduleAppointment(appointment);

  assert.strictEqual(beautySalon.appointments.length, 1);
  assert.strictEqual(beautySalon.appointments[0], appointment);
});

it('should cancel an appointment', () => {
  const beautySalon = new BeautySalon();
  beautySalon.addEmployee(employee);
  beautySalon.addService(service);

  const appointment = new Appointment(employee, client, service, new Date(), '10:00', '11:00');
  beautySalon.scheduleAppointment(appointment);
  beautySalon.cancelAppointment(appointment);

  assert.strictEqual(beautySalon.appointments.length, 0);
});

it('should get the schedule of an employee', () => {
  const beautySalon = new BeautySalon();
```

```
    beautySalon.addEmployee(employee);
    beautySalon.addService(service);

    const appointment1 = new Appointment(employee, client, service, new Date(), '10:00', '11:00');
    const appointment2 = new Appointment(employee, client, service, new Date(), '12:00', '13:00');
    beautySalon.scheduleAppointment(appointment1);
    beautySalon.scheduleAppointment(appointment2);

    const schedule = beautySalon.getEmployeeSchedule(employee);

    assert.strictEqual(schedule.length, 2);
    assert.strictEqual(schedule[0], appointment1);
    assert.strictEqual(schedule[1], appointment2);
  });
});
```

ДОДАТОК Г

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Хмельницький Національний Університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

Кваліфікаційна робота, на тему:
«Веб-додаток для автоматизації роботи салону краси та організації роботи з клієнтами»

Студент: Бендій Данило Михайлович
Керівник: канд. пед. наук., доцент Онищко О.Г.

Вступ

- За останні роки можна спостерігати зростання інтересу до здорового способу життя та зовнішнього вигляду у більшості країн світу. Люди стають більш усвідомленими щодо того, що здоров'я тіла та душі є важливими для їхнього щастя та добробуту. Зокрема, зростає популярність різних видів фізичних вправ та спорту, здорового харчування, а також процедур та процесів догляду за зовнішнім виглядом.
- Це також стимулює розвиток сфери краси та сприяє зростанню попиту на послуги салонів краси, де пропонуються різні процедури догляду за обличчям, тілом та волоссям. Відвідування салонів краси допомагає людям підтримувати свій зовнішній вигляд, поліпшувати самопочуття та підвищувати самооцінку. Зростання інтересу до здорового способу життя та зовнішнього вигляду може сприяти подальшому розвитку індустрії краси та стимулювати зростання кількості салонів краси.

Актуальність

- Загальний ріст індустрії краси:
  - За останні роки індустрія краси переживає стійкий ріст, який прогнозується продовжуватись і надалі.
  - За даними Global Industry Analysts, Inc., світовий ринок косметики та особистої гігієни очікується досягти значення понад 800 мільярдів доларів США до 2023 року.
- Зростання популярності веб-застосунків у сфері краси:
  - За даними Statista, кількість завантажень мобільних додатків, пов'язаних з красою, у 2020 році перевищила 3 мільярди.
  - За даними App Annie, веб-застосунки для бронювання послуг краси займають високі позиції у списку найпопулярніших додатків у категорії "Стиль життя" на платформах iOS та Android.

Мета та завдання проєкту

- Головна мета даної роботи- розробка веб-додатку для автоматизації роботи з клієнтами та організації роботи салону краси з метою підвищення задоволеності клієнтів наданням якісних та швидких послуг та збільшення ефективності роботи салону краси.
- Для досягнення мети були сформовані такі завдання:

- Провести детальний аналіз предметної області.

- Проаналізувати наявне програмне забезпечення в цій галузі.

- На основі попереднього аналізу встановити вимоги до програмного продукту, на основі яких необхідно обрати найкращі та найефективніші методи і засоби для розробки програмного забезпечення.

- Спроєктувати структуру веб-додатку та розробити базу даних.

- Розробити програмне забезпечення відповідно до раніше встановлених вимог та провести тестові випробування

Призначення

- Додаток покликаний вирішити певні проблеми, які виникають при управлінні таким підприємством як салон краси.
- Завдання які вирішуються додатком:

1. Організація інформаційного обліку.
2. Організація процесу запису клієнтів.
3. Організація роботи зі співробітниками.

Аналіз існуючого програмного забезпечення

- При проєктуванні програмного забезпечення для салону краси, важливо врахувати потреби користувачів, які шукають веб-додаток для автоматизації роботи з клієнтами та організації роботи підприємства. На сьогоднішній день на ринку існують різноманітні додатки для салонів краси, однак, кожен з них має свої переваги та недоліки. Для успішного розроблення нового додатку необхідно вивчити недоліки наявних рішень та знайти способи їх подолання.
- Окремі приклади існуючих додатків для автоматизації роботи салонів краси та організації роботи з клієнтами включають такі: Shedul, Mindbody, Acuity Scheduling
- Основними недоліками додатків є складність інтерфейсу, висока вартість та обмежений функціонал.

Логотипи додатків

shedul.

Acuity Scheduling

mindbody

Порівняльна таблиця

Ознаки	Acuity Scheduling	Shedul	Mindbody
Призначення програмного продукту	Онлайн-система бронювання та планування зустрічей	Онлайн-система бронювання та планування зустрічей	Онлайн-платформа для управління бізнесом у сфері фітнесу, краси та здоров'я
Фірма-розробник	Acuity Scheduling	Shedul	Mindbody Inc.
Основні інтерфейсні вікна	Календар, форма бронювання, повідомлення	Календар, форма бронювання, повідомлення	Розклад класів, управління продажами, фінансами, аналітика, звіти
Переваги	Легкий у використанні, можливості інтеграції зі сторонніми програмами	Безкоштовна, простий інтерфейс, мобільний застосунок	Широкий функціонал, висока безпека, інтеграція зі сторонніми програмами, підтримка мультилокальності та мультиплатформеності
Недоліки	Обмежені можливості на безкоштовному плані, висока ціна на розширені функції	Обмежені можливості на безкоштовному плані, складність налаштування	Висока ціна на програмне забезпечення та послуги підтримки, відсутність налаштування інтерфейсу, складність в освоєнні для некваліфікованих користувачів

Постановка задачі та вимоги до розробки

Групи користувачів
Гість
Зареєстрований користувач
Працівник-майстер
Працівник-адміністратор

Основні функції
Реєстрація та авторизація користувачів
Ведення бази даних клієнтів
Керування робочим персоналом та облік записів
Бронювання та скасування послуг клієнтами

Функціональні вимоги

- Працівник – в залежності від посади має різні функціональні можливості:
- «майстер» -ведення своєї роботи, запису на прийом, перегляду інформації про клієнтів, інформації про послуги, що продаються в салоні краси.
- «Адміністратор» - керування системою, в залежності від доступних прав, додавання та видалення даних, редагування профілів користувачів і налаштування параметрів системи
- Клієнти – можливість запису на прийом, перегляду свого профілю, перегляду послуг, які пропонуються в салоні краси, і їх замовлення.
- Гості – перегляд основної інформації доступної на сайті без реєстрації, реєстрація на сайті.

Вибір системи керування даними

MongoDB є нереляційною (NoSQL) базою даних, яка має кілька переваг:

- Гнучкість: MongoDB дозволяє зберігати дані у вигляді документів, які можуть мати різну структуру.
- Швидкодія: MongoDB пропонує високу швидкодію завдяки своїй архітектурі.
- Потужні можливості запитів: MongoDB надає потужний механізм запитів, включаючи підтримку індексів та розширені можливості для фільтрації, сортування та агрегації даних.

Це все спрощує роботу з базою даних та забезпечує ефективну обробку даних у веб-додатку.

Розробка структури інформаційної ситсеми

Clients	Employees	Appointments	Services
_id(Object_ID)	_id(Object_ID)	_id(Object_ID)	_id(Object_ID)
Name(String)	Name(String)	client (ObjectId)	Name(String)
Surname(String)	Surname(String)	Service(String)	Price(Number)
email (String)	email(string)	Price(Number)	Desription(String)
Phone(string)	Phone(string)	Date(Date)	
Adress(String)	position(string)	Status(string)	
Appoinmetns(Array)	Schedule(String)		
	Appoinmetns(Array)		

Вибір шаблону проєктування

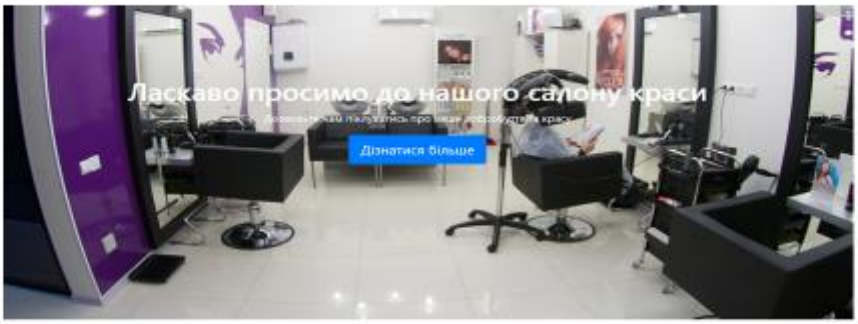
MVC (Model-View-Controller) є шаблоном проєктування, який широко використовується для розробки програмного забезпечення. В MVC модель додатку розбивається на три основні компоненти:

- Модель (Model): Відповідає за управління даними та бізнес-логікою додатку.
- Представлення (View): Відповідає за візуалізацію даних та взаємодію з користувачем.
- Контролер (Controller): Відповідає за обробку вхідних подій, таких як взаємодія користувача з представленням.

MVC шаблон дозволяє розділити логіку додатку на компоненти, що підвищує його читабельність, повторне використання коду та тестованість. Крім того, він сприяє поділу ролей у команді розробників, дозволяючи фахівцям займатися своїми областями відповідальності.

Використані технології

Реалізація проєкту. Головна сторінка



Про нас

Опис про ваш салон краси. Розкажіть про ваші послуги, експертизу та вартість. Виділіть свої унікальні особливості та переваги, що роблять ваш салон кращим.

Наші послуги

Перелік послуг, які ви надаєте в своєму салоні краси. Розкажіть про різноманітність процедур, стрижки, укладки, манікюр, педикюр та інші послуги, які ви забезпечуєте.

Реалізація проєкту. Сторінка Послуги

Наші послуги

Стрижка

Короткий опис про стрижку.

$30

Педикюр

Короткий опис про педикюр.

$20

Манікюр

Короткий опис про манікюр.

$20

Реалізація проєкту. Панель керування

Салон краси

Головна Клієнти Працівники Послуги Прийоми Вихід

Панель керування

Клієнти

Управління даними клієнтів.

Перейти

Працівники

Управління даними працівників.

Перейти

Прийоми

Управління даними прийомів.

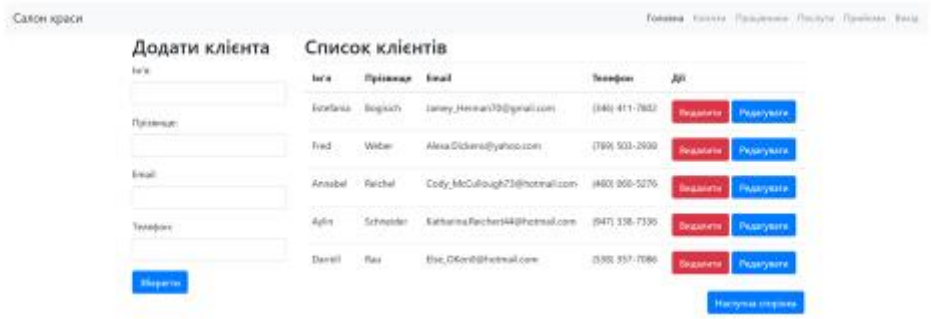
Перейти

Послуги

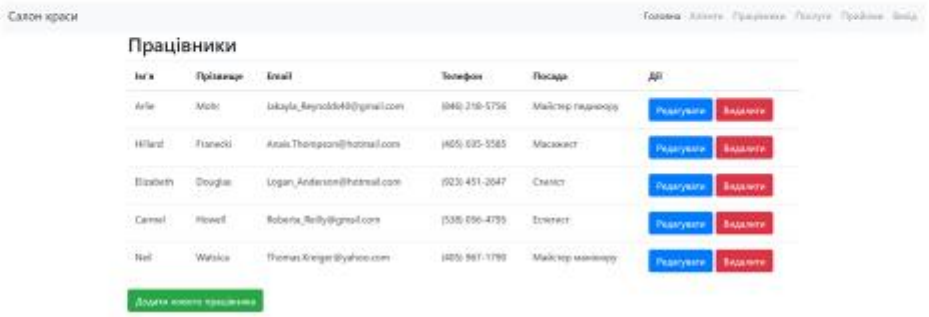
Управління даними послуг.

Перейти

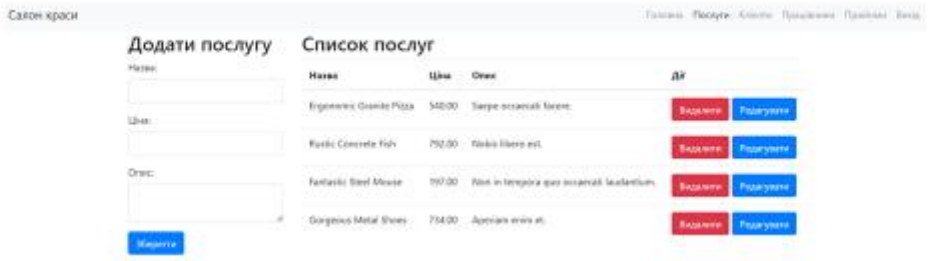
Реалізація проєкту. Сторінка Клієнти



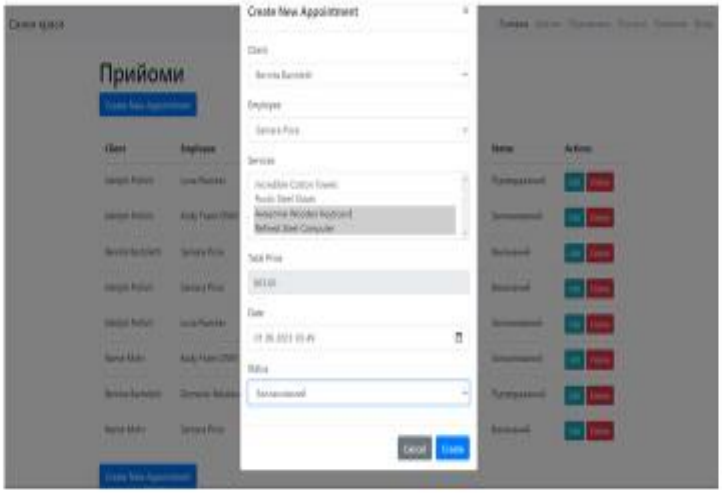
Реалізація проєкту. Сторінка працівників



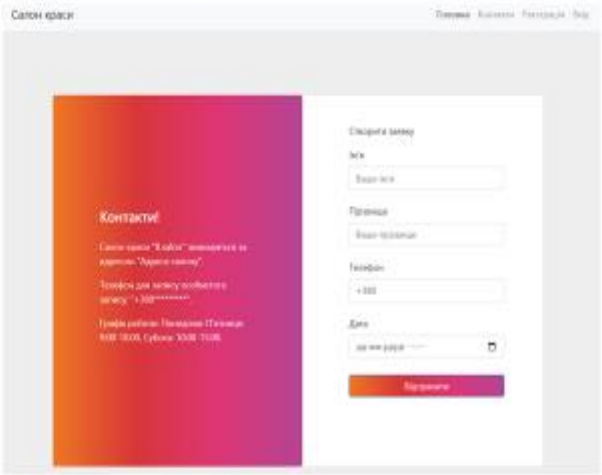
Реалізація проєкту. Сторінка послуги



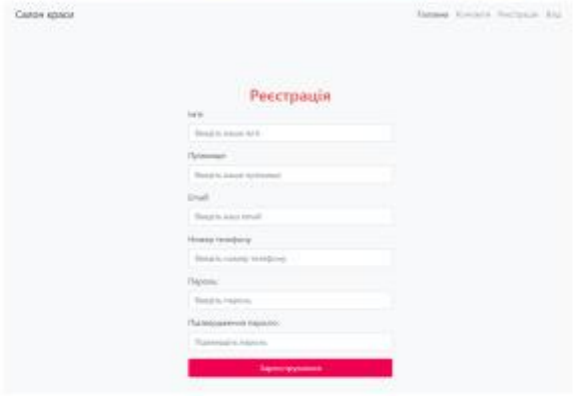
Реалізація проєкту. Сторінка прийомів



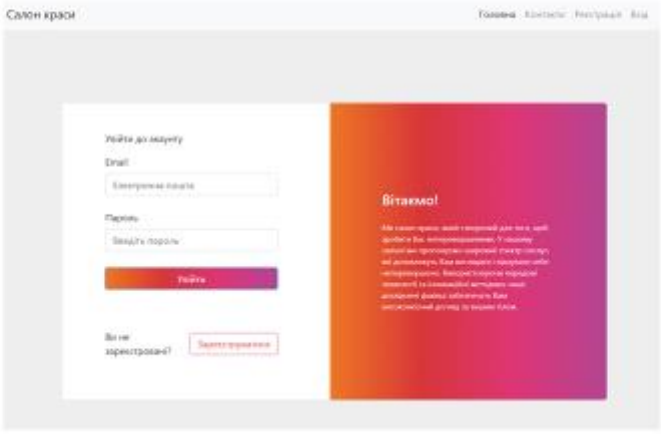
Реалізація проєкту. Сторінка Контакти



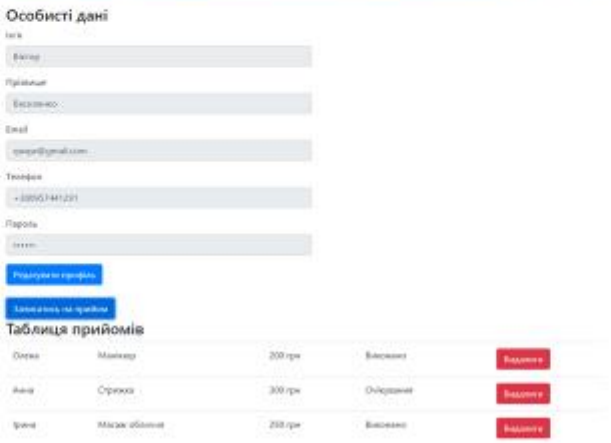
Реалізація проєкту. Сторінка реєстрації



Реалізація проєкту. Сторінка авторизації



Реалізація проєкту. Особистий кабінет клієнта



Реалізація проєкту. Форма створення прийому

Підсумок розробки веб-додатку

- Розроблений веб-додаток забезпечує користувачам весь зазначений функціонал в залежності від їх ролей, та вирішує усі поставлені перед ним завдання.
- Організація інформаційного обліку
- Користувачі отримують доступ до різноманітної інформації, такої як:
  1. Інформація про послуги.
  2. Інформація про замовлення.
  3. Дані клієнтів та працівників салону
- Організація процесу запису клієнтів. Клієнти та працівники салону отримують можливість створювати подію пов'язану з виконанням певним працівником салону певного набору послуг.
- Організація роботи зі співробітниками. Працівники та адміністрація салону отримує можливість керувати даними та списками персоналу.

Висновок

- Одним з найкращим способом збільшити прибутки салону краси є створення додатку для автоматизації роботи салону краси.
- Це допоможе покращити організацію та ефективність роботи салону, спростить процес роботи з клієнтами, забезпечить зручний доступ до інформації і допоможе підвищити задоволення клієнтів і подарувати їм приємний досвід взаємодії з салоном.

Дякую за увагу!

ГРАФІЧНА ЧАСТИНА

(обов’язкова)

Завідувачу кафедри
інженерії програмного забезпечення
проф. Бедратюку Л. П.
студента групи ІПЗс-20-1
Бецція Д.М.
Прізвище, ініціали

ЗАЯВА

Прошу закріпити за мною тему кваліфікаційної роботи освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення»:

Веб-застосунок для автоматизації роботи салону краси та організації роботи з клієнтами

(керівник роботи – Онишко Оксана Григорівна)
Прізвище, ім'я, по батькові

05.02.2023р
Дата

Підпис студента

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти
Бендія Д. М.

Прізвище, ініціали

факультет ІТ, 3 курс, група ІПЗс-20-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності в Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений . Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та/або Anti-Plagiarism) і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

30.05.2023
дата

підпис

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ДЕКЛАРАЦІЯ УЧАСНИКА ОСВІТНЬОГО ПРОЦЕСУ

щодо дотримання академічної доброчесності

Цією декларацією я, Бендій Данило Михайлович
Прізвище, ім'я, по батькові

студент III курсу факультету інформаційних технологій, кафедри інженерії
здобувач вищої освіти (шифр та назва спец-ті, курс, академічна група)/ науково-педагогічний працівник (назва кафедри)

програмного забезпечення
назва факультету

підтверджую, що ознайомився (- лась) з Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті та Кодексом академічної доброчесності і **зобов'язуюсь** дотримуватися їх вимог під час освітнього процесу, проведення наукової діяльності, виконання організаційно-адміністративних функцій тощо.

Усвідомлюю, що у разі порушення мною принципів академічної доброчесності нестиму відповідальність перед академічною спільнотою ХНУ згідно з нормами, визначеними Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, законодавства України.

«_05_» __лютого_ 2023 р.

Підпис

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1015388702

Дата перевірки:
02.06.2023 11:57:42 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
02.06.2023 12:01:53 EEST

ID користувача:
100005589

Назва документа: **КвР_Бендій_ІПЗс_20_1_для_антиплагіату**

Кількість сторінок: **69** Кількість слів: **11065** Кількість символів: **87397** Розмір файлу: **5.82 MB** ID файлу: **1015053554**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

12.1%
Схожість

Найбільша схожість: **3.82%** з джерелом з Бібліотеки (ID файлу: **1015017362**)

7.48% Джерела з Інтернету	584	Сторінка 71
6.99% Джерела з Бібліотеки	101	Сторінка 75

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування	14 сторінок

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 3.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 11%**

	Документ		Сумарний збіг по Базі Даних	
ID: 114559 Назва: БКР Веб-застосунок для автоматизації роботи салону краси та організації роботи з клієнтами Додано в БД: 2023-06-02 Автора: Бендій Д.М. Керівники: Онишко О.Г. Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	70044	633	4862 (7%)	61 (10%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: «Веб застосунок для автоматизації роботи салону краси та організації роботи з клієнтами»
Автор: Бендій Данило Михайлович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Онишко Оксана Григорівна, кандидат педагогічних наук, доцент
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, відомість документів), у структурі змісту, назвах розділів/підрозділів тощо, у назвах публікацій у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посиланання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 12,1% і адресується до 584 джерел, що, з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Оксана ОНИШКО

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Бендій Данило Михайлович

Тема Веб-застосунок для автоматизації роботи салону краси та організації роботи з клієнтами

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 ; кількість сторінок записки 130

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі було досліджено і проаналізовано предметну область, визначено усі функціональні та нефункціональні вимоги. Був проведений аналіз існуючих програм на ринку, розглянуто їх переваги і недоліки, та доведено актуальність розробки нового програмного забезпечення. Розглянуто інструменти для реалізації спроектованих рішень, в результаті чого створено програмне забезпечення. Також було проведено тестування програми, за результатами якого доведено, що розроблене програмне забезпечення працює коректно та готове до експуатації.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі доведено актуальність теми, визначено мету та завдання дипломного проектування. У першому розділі проведено аналіз предметної області, розглянуто існуючі рішення та визначені функціональні і нефункціональні вимоги до розроблюваного програмного забезпечення. У другому розділі проведено аналіз сучасних архітектур, розглянуто їх переваги і недоліки та визначено, що система буде відповідати монолітній архітектурі та моделі клієнт-сервер. У третьому розділі підготовлено всі залежності для написання коду та виконано практичну розробку програмних модулів і описано їх особливості, в результаті чого створено програмний продукт, а також виконано модульне тестування системи та проведено його у відповідності до функціональних вимог, в результаті було підтверджено коректну роботу програми.

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є актуальною, оскільки популярність сфери краси стрімко зростає і буде довго зберігати тенденцію росу. Також було застосовано новітні технології для побудови програмного продукту . та актуальні архітектурні рішеня

5. Негативні сторони роботи У роботі було використано достатньо складні в супроводжені технології. Безпеці передачі файлів медіа матеріалів, в межах системи, не було приділено належної уваги.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки. Матеріал пояснювальної записки структурований, послідовний, чіткий та простий, що дозволяє чітко зрозуміти викладений матеріал у рамках тематики проектування. Графічний матеріал дає можливість наочно побачити деталі проектування системи.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «добре».

РЕЦЕНЗЕНТ Професор кафедри Комп'ютерної інженерії та інформаційних технологій, д. т. н. професор Лисенко Сергій Михайлович

"2" червня 2023 р. _____
(підпис)