Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра інформаційних систем та комп'ютерного моделювання
(повна назва кафедри (предметної, циклової комісії))

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему: **Розроблення CRM-системи для мережі барбершопів**

Виконав: студент *II* курсу групи *ІСТС-21*
спеціальності 126 "Інформаційні системи
і технології"
(шифр і назва напряму підготовки, спеціальності)

Буждиган С.М.
(прізвище та ініціали)

Керівник *Шабатура Ю.В.*
(прізвище та ініціали)

Рецензент *Карашевський В.П.*
(прізвище та ініціали)

Львів – 2024

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій

Кафедра інформаційних систем та комп'ютерного моделювання

Рівень вищої освіти перший (бакалаврський)

Спеціальність 126 "Інформаційні системи та технології"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІСКМ
_____Сторожук О.Л.
"06" 02 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Буджигану Сергію Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Розроблення CRM-системи для мережі барбершопів*

Керівник роботи *Шабатура Юрій Васильович, д.т.н., проф.*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "06" лютого 2024 року № - 87 .

2. Термін подання студентом роботи *10.06.2024 р.*

3. Вихідні дані до роботи *Постановка задачі та її формалізації. Аналіз сервісів створення засобів комунікації користувачів web-ресурсів.*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Вступ
Розділ 1. Опис предметної області
Розділ 2. Інформаційне та математичне забезпечення
Розділ 3. Програмно-технологічне забезпечення
Висновки
Список використаних джерел

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

6. Дата видачі завдання 7 лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ № з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Огляд літературних даних	06.02-30.02.2024	виконано
2	Розділ 1. Стан проблемної області	01.03-05.03.2024	виконано
3	Розділ 2. Інформаційне та математичне забезпечення	05.03-15.03.2024	виконано
4	Розділ 3. Програмне та технічне забезпечення	16.03-15.05.2024	виконано
5	Оформлення дипломної роботи	16.05-31.05.2024	виконано
6	Підготовка до захисту дипломної роботи, оформлення презентації	01.06-09.06. 2024	виконано

Студент Буждиган С. М. _____
(підпис)

Керівник д. т. н., проф. Шабатура Ю. В. _____
(підпис)

АННОТАЦІЯ

Дипломна робота містить 44 сторінок пояснювальної записки, 18 зображень, 15 джерел, 1 додаток.

В дипломній роботі спроектовано і реалізовано інформаційну веб систему відповідно до технічного завдання. Вона розроблена засобами мови програмування javascript з використанням pocketbase для управління моделлю даних що реалізує збереження інформації та отримання даних клієнтом, що в свою чергу виконаний на фулстек фреймворку sveltekit та компонентами бібліотеки svelteui. Тестування цієї програми показало її цілісність та структурованість, вона має властивості, що були поставлені в технічному завданні при проектуванні. Дана система призначена для автоматизації процесів обробки записів в барбершопі в зручному веб інтерфейсі.

Ключові слова: MPA, JavaScript, Sveltekit, SvelteUi, Pocketbase, Barbershop, CRM, Docker, Proxy.

ABSTRACT

The diploma project consists of a 44 pages explanatory report, 18 images, 15 references, 1 appendice.

The diploma project involves the design and implementation of an informational web system in accordance with the technical specification. It is developed using the JavaScript programming language, utilizing Pocketbase for managing the data model that handles information storage and client data retrieval. The client is built on the full-stack framework Sveltekit and incorporates components from the SvelteUi library. Testing of this program has demonstrated its integrity and structured nature, meeting the properties outlined in the technical specification during the design phase. The system is designed to automate the processes of handling records in a barbershop through a user-friendly web interface.

Keywords: MPA, JavaScript, Sveltekit, SvelteUi, Pocketbase, Barbershop, CRM, Docker, Proxy.

ТЕХНІЧНЕ ЗАВДАННЯ

В даній дипломній роботі потрібно вирішити наступні завдання:

- провести огляд літературних джерел, програмної документації, а також відповідних зразків веб шаблонів;
- розглянути і використати графічні рішення сучасних UI/UX практик, які лежать в основі логічної моделі інтерфейсу;
- визначити сучасні стан предметної області барбершопів та проаналізувати вимоги і потребу в них для кінцевого користувача
- спроектувати та розробити серверну та клієнтську частини засобами pocketbase та sveltekit;
- провести тестування розробленого програмного продукту у відповідності з поставленим завданням;

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

MPA – Multi Page Application.

REST – REpresentational State Transfer.

API – Application Programming Interface.

UI – User Interface.

UX – User eXperience.

SSR – Server Side Rendering.

Unix – формат представлення часу.

Proxy – архітектурний паттерн що представляє посередника в певному процесі.

SMTP – Simple Mail Transfer Protocol.

HTML – HyperText Markup Language.

ВСТУП

Актуальність дипломної роботи. CRM система для барбершопів є актуальною і може принести значну користь бізнесу. Деякі з переваг використання CRM системи для барбершопу:

- Управління записами: CRM система дозволяє автоматизувати процес запису клієнтів, що звільняє час для інших завдань. Клієнти також можуть самостійно записуватися онлайн, що зручно для них.
- Управління відносинами з клієнтами: CRM система дозволяє зберігати інформацію про клієнтів, включаючи їхні контактні дані, історію обслуговування та уподобання. Це допомагає краще зрозуміти потреби клієнтів і надавати їм персоналізований сервіс.
- Маркетинг і продажі: CRM система дозволяє створювати та керувати маркетинговими кампаніями, а також відстежувати результати. Це допомагає залучати нових клієнтів і підвищувати лояльність існуючих.
- Аналіз даних: CRM система дозволяє збирати та аналізувати дані про бізнес, наприклад, про продажі, клієнтський потік і ефективність маркетингових кампаній. Це допомагає приймати обґрунтовані рішення щодо розвитку бізнесу.

Звичайно, ефективність CRM системи залежить від того, як вона використовується. Важливо правильно налаштувати систему і навчити персонал працювати з нею. Але в цілому, CRM система може бути цінним інструментом для підвищення ефективності барбершопу і зростання його бізнесу.

Об'єкт дослідження – процес автоматизованої обробка самостійних клієнтських записів.

Предмет дослідження – crm система для бізнесу у сфері надання послуг.

Мета роботи – розробка crm системи для автоматизації онлайн запису на процедури в барбершопі.

Завдання дипломної роботи:

- провести огляд літературних джерел та публікацій за темою розробки, та існуюче програмне забезпечення;

- проаналізувати архітектурні рішення що застосовуються при побудові подібних систем.
- спроектувати та імплементувати серверну частину системи з допомогою sveltekit
- розробити модель даних в системі pocketbase
- розробити сучасний дизайн з його подальшою верстkою засобами svelte та бібліотеки компонентів svelteui.
- описати розгортання системи в докерфайлі

Практична значимість – дана система для має переваги на фоні існуючих рішень:

- Автономність: Інформаційна система надає можливість клієнтам самостійно обирати дату\час запису та майстра, без участі менеджера.
- Доступність: Розроблене рішення доступне у вигляді відкритого вихідного коду з налаштованою збіркою, тож є дешевшим в використанні ніж існуючі crm для що пропонуються як saas рішення.
- Комфорт: Віддалений запис без необхідності контактувати з кимось по телефону – є зручним і затребуваним серед великої кількості сучасних людей, що уникають лишніх соціальних контактів.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Загальна інформація

CRM (Customer Relationship Management) - це система управління взаємовідносинами з клієнтами. CRM-системи допомагають компаніям збирати, зберігати та аналізувати інформацію про клієнтів, а також використовувати цю інформацію для покращення обслуговування клієнтів та підвищення продажів.

Основні функції CRM-систем:

- Управління клієнтами. CRM-системи дозволяють зберігати інформацію про клієнтів, включаючи їх контактні дані, історію взаємодій, уподобання та іншу важливу інформацію. Ця інформація може бути використана для персоналізації обслуговування клієнтів та підвищення їх лояльності.
- Управління замовленнями. CRM-системи дозволяють керувати записами на прийоми, бронювати послуги та продукти та відправляти нагадування клієнтам. Це допомагає забезпечити ефективну роботу компанії та мінімізувати ризик незапланованих відвідувань.
- Управління маркетингом. CRM-системи дозволяють створювати та керувати маркетинговими кампаніями, а також відстежувати їх ефективність. Це допомагає залучати нових клієнтів та підвищити продажі.

CRM-системи можуть використовуватися різними компаніями, незалежно від їхнього розміру та галузі. CRM-системи можуть бути корисними для компаній, які хочуть:

- Поліпшити обслуговування клієнтів
- Підвищення лояльності клієнтів
- Зменшити витрати на продажі та маркетинг
- Підвищення продажів

CRM-системи можуть бути хмарної або локальною. Хмарні CRM-системи розміщуються на серверах постачальника послуг, а локальні CRM-системи розміщуються на власних серверах компанії.

1.2 Огляд існуючих рішень

Більшість існуючих crm систем орієнтовані на сферу електронної комерції, а решта – узагальненого типу, з можливістю проектування під конкретну організацію. Самих по собі ентерпрйз crm спеціально для барбершопів немає, тільки невеликі ріішення. Наприклад Altegio – це хмарна CRM-система, призначена для автоматизації бізнес-процесів у сфері послуг. Вона допомагає компаніям підвищити ефективність роботи, збільшити продажі та поліпшити обслуговування клієнтів.

Altegio пропонує широкий спектр функцій, які дозволяють:

- Автоматизувати запис клієнтів. Клієнти можуть самостійно записатися на прийом через веб-сайт або мобільний додаток.
- Створювати та керувати записами. Менеджери можуть створювати, змінювати та відмовлятися від записів.
- Відстежувати статус записів. Менеджери можуть відстежувати статус записів, щоб бути в курсі того, які клієнти прийдуть на прийом.
- Надавати інформацію клієнтам. Менеджери можуть надсилати клієнтам повідомлення про їхні записи, а також інформацію про послуги та продукти компанії.
- Аналізувати дані про клієнтів. Менеджери можуть аналізувати дані про клієнтів, щоб отримати уявлення про їхні потреби та поведінку.

Altegio доступна в декількох тарифних планах, які підходять для компаній різного розміру.

Ось деякі з переваг використання Altegio:

- Покращене обслуговування клієнтів. Altegio допомагає компаніям забезпечити клієнтам кращий досвід обслуговування, автоматизуючи записи, надаючи інформацію та аналізуючи дані.
- Збільшені продажі. Altegio допомагає компаніям підвищити продажі, збираючи дані про клієнтів та надаючи їм персоналізовані пропозиції.
- Покращена ефективність роботи. Altegio допомагає компаніям автоматизувати рутинні завдання, звільняючи час для більш важливих завдань.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 JavaScript

JavaScript - це динамічна, об'єктно-орієнтована, прототипна мова програмування, яка використовується для створення інтерактивних веб-додатків і вважається однією з найбільш популярних у світі. Вона використовується як у фронтенді, так і у бекенді.

Фронтенд - це частина веб-додатку, яка взаємодіє з користувачем. Вона відповідає за відображення вмісту веб-сторінки, реакцію на дії користувача та надання інтерактивних функцій.

Бекенд - це частина веб-додатку, яка працює на сервері. Вона відповідає за обробку даних, виконання запитів користувачів та надання інформації фронтенду.

JavaScript є ідеальним вибором для розробки фронтенду, оскільки підтримується усіма веб-браузерами. Розробники можуть використовувати JavaScript для створення:

- Інтерактивних веб-сторінок, які реагують на дії користувача: наприклад, зміна кольору фону при наведенні курсора миші або відображення меню при натисканні на кнопку.
- Динамічного вмісту, що змінюється в залежності від даних або дій користувача: наприклад, відображення погоди або новин на веб-сторінці.
- Веб-додатків із функціями, такими як навігація, пошук і форми: наприклад, онлайн-магазин або соціальна мережа.

За допомогою середовища виконання JavaScript поза браузером, такого як Node.js, розробники можуть будувати швидкі та масштабовані серверні додатки. Node.js дозволяє використовувати JavaScript для обробки запитів користувачів, виконання баз даних та інших завдань, які зазвичай виконувалися б мовами програмування, такими як C++ або Java.

Виникнення нових стандартів, таких як ECMAScript 6 (ES6) і його наступних версій, додало багато нових можливостей для розробників, таких як:

- Стрілкові функції: простий і ефективний спосіб створення функцій.
- Деструктуризація: спосіб розділення об'єктів або масивів на їхні складові частини.
- Класи: об'єктно-орієнтована модель програмування, яка полегшує створення складних програм.
- Модулі: спосіб організації коду в логічні блоки.

У світі фронтенду одним із найпопулярніших фреймворків є React, який пропонує компонентний підхід до розробки. React дозволяє розробникам створювати веб-сторінки з використанням невеликих, повторно використовуваних компонентів. Інші популярні фреймворки для фронтенду включають Angular, Vue.js та Svelte.

У бекенді Express та Nest є популярними фреймворками для побудови серверів та маршрутизації запитів. Express - це простий і легкий фреймворк для Node.js, який дозволяє розробникам швидко створювати серверні додатки. Nest - це більш складний фреймворк, який пропонує більше можливостей та налаштування.

JavaScript активно використовується в сучасних стеках розробки, таких як MERN, MEAN, PERN та інші. Ці стеки дозволяють розробникам будувати повноцінні додатки на базі JavaScript, використовуючи інструменти, які ідеально поєднуються між собою.

Популярність JavaScript продовжує зростати, а розробники постійно досліджують нові інструменти та підходи, щоб працювати з цією мовою програмування ще більш продуктивно і ефективно. Хоча JavaScript є потужним і універсальним інструментом у сучасній веб-розробці, важливо враховувати, що вона може бути складною для вивчення та не завжди є оптимальним вибором для всіх видів завдань.

2.2 Pocketbase

PocketBase [13] – це відкрите програмне забезпечення для бекенду, яке складається з вбудованого репозиторію SQLite з підтримкою реального часу,

вбудованої системи управління файлами та користувачами, зручного інтерфейсу керування адміністратора та простого REST-подібного API.

PocketBase можна завантажити безпосередньо як автономну програму або використовувати як фреймворк/набір інструментів JS, який дозволяє створювати власну бізнес-логіку для конкретного додатка, при цьому в кінцевому підсумку отримується єдиний портативний виконуваний файл.

Основні особливості

- Реєстр даних реального часу: PocketBase використовує вбудований репозиторій SQLite, який забезпечує підтримку реального часу для змін даних. Це означає, що клієнти можуть отримувати доступ до найновіших даних в режимі реального часу, без необхідності постійно запитувати сервер.

- Вбудована система управління файлами: PocketBase також включає в себе вбудовану систему управління файлами, яка дозволяє зберігати файли на сервері. Це робить PocketBase ідеальним рішенням для додатків, які потребують зберігання файлів, таких як веб-додатки, мобільні додатки та програмне забезпечення для управління клієнтами.

- Вбудована система управління користувачами: PocketBase включає в себе вбудовану систему управління користувачами, яка дозволяє керувати обліковими записами користувачів, правами доступу та іншими аспектами безпеки. Це робить PocketBase ідеальним рішенням для додатків, які потребують захисту даних користувачів.

- Зручний інтерфейс керування адміністратора: PocketBase включає в себе зручний інтерфейс керування адміністратора, який дозволяє адміністраторам керувати сервером PocketBase. Цей інтерфейс дозволяє адміністраторам виконувати такі завдання, як створення облікових записів користувачів, налаштування прав доступу та відстеження активності

сервера.

- Простий REST-подібний API: PocketBase надає простий REST-подібний API, який дозволяє клієнтам отримувати доступ до даних і функціональності сервера. Цей API простий у використанні та дозволяє клієнтам швидко та легко розробляти додатки для PocketBase.

Використання

PocketBase можна використовувати для створення широкого спектру додатків, включаючи:

- Веб-додатки
- Мобільні додатки
- Програмне забезпечення для управління клієнтами
- Програмне забезпечення для обробки даних
- Програмне забезпечення для IoT

PocketBase – це потужний і універсальний інструмент, який можна використовувати для створення швидких, надійних та масштабованих додатків.

2.3 **SMTP**

SMTP [14] - це абревіатура від Simple Mail Transfer Protocol. Це протокол прикладного рівня TCP/IP, який використовується для передачі електронної пошти. SMTP використовує зв'язок TCP на порту 25.

SMTP працює в режимі клієнт-сервер. Клієнтський додаток електронної пошти, такий як Gmail або Outlook, використовує SMTP для надсилання електронних листів на сервер електронної пошти. Сервер електронної пошти, такий як Gmail або Exchange, використовує SMTP для доставки електронних листів на сервери інших постачальників електронної пошти.

Етапи процесу SMTP:

1. Авторизація: Клієнтський додаток електронної пошти авторизується на сервері електронної пошти, надавши ім'я користувача та пароль.

2. Формування повідомлення: Клієнтський додаток електронної пошти формує повідомлення електронної пошти, включаючи адреси відправника та одержувача, тему та текст повідомлення.

3. Передача повідомлення: Клієнтський додаток електронної пошти передає повідомлення серверу електронної пошти.

4. Доставка повідомлення: Сервер електронної пошти доставляє повідомлення на сервер одержувача.

SMTP використовує ряд команд для управління процесом передачі електронної пошти. Ось деякі з основних команд SMTP:

- HELO: Клієнтський додаток електронної пошти використовує команду HELO для інформування сервера електронної пошти про своє ім'я.
- MAIL FROM: Клієнтський додаток електронної пошти використовує команду MAIL FROM для визначення адреси відправника.
- RCPT TO: Клієнтський додаток електронної пошти використовує команду RCPT TO для визначення адрес одержувача.
- DATA: Клієнтський додаток електронної пошти використовує команду DATA для початку передачі повідомлення електронної пошти.
- QUIT: Клієнтський додаток електронної пошти використовує команду QUIT для завершення сеансу SMTP.

SMTP є відкритим протоколом, що означає, що будь-хто може перехопити та прочитати електронні листи, які передаються за допомогою SMTP. Для підвищення безпеки електронної пошти часто використовується SMTP з SSL/TLS. SSL/TLS шифрує повідомлення електронної пошти, щоб їх не можна було перехопити та прочитати.

SMTP може використовуватися для різних додаткових функцій, таких як:

- Розсилання повідомлень: SMTP можна використовувати для розсилки повідомлень електронної пошти великій кількості людей.

- Спам: SMTP часто використовується для розсилки спаму. Спам - це небажані електронні листи, які часто містять рекламу або інші небажані повідомлення.
- Інформаційні повідомлення: SMTP можна використовувати для надсилання інформаційних повідомлень, таких як повідомлення про оновлення програмного забезпечення або повідомлення про обслуговування.

SMTP є важливим протоколом, який використовується для передачі електронної пошти. Він простий у використанні та ефективний, але має обмеження безпеки.

2.4 Sqlite

SQLite [6, 7, 8, 9] - це реляційна система керування базами даних, яка зберігається в єдиному файлі на диску. Вона написана на C і є відкритим програмним забезпеченням. SQLite є найпопулярнішою базою даних у світі, і її використовують у широкому спектрі програм, включаючи веб-додатки, мобільні додатки, програмне забезпечення для управління клієнтами та програмне забезпечення для обробки даних.

Основні особливості SQLite:

- Малий розмір: SQLite - це невелика база даних, яка займає лише кілька мегабайтів пам'яті. Це робить її ідеальною для використання в пристроях з обмеженими ресурсами, таких як мобільні телефони та планшети.
- Швидкість: SQLite є дуже швидкою базою даних. Вона може виконувати операції з базами даних набагато швидше, ніж деякі інші великі бази даних.
- Надійність: SQLite є дуже надійною базою даних. Вона має ряд функцій, які допомагають захистити дані від втрати або пошкодження, таких як журнали відновлення та підтримка резервних копій.
- Підтримка SQL: SQLite підтримує більшість основних функцій SQL, таких як створення таблиць, інсерція даних, вибір даних та оновлення даних.

SQLite можна використовувати для створення широкого спектру додатків, включаючи:

- Веб-додатки: SQLite часто використовується в веб-додатках для зберігання даних користувачів, таких як імена користувачів, паролі та інформація про облікові записи.
- Мобільні додатки: SQLite часто використовується в мобільних додатках для зберігання даних, таких як контакти, календарі та записи.
- Програмне забезпечення для управління клієнтами: SQLite часто використовується в програмному забезпеченні для управління клієнтами для зберігання інформації про клієнтів, таких як імена, адреси та номери телефонів.
- Програмне забезпечення для обробки даних: SQLite часто використовується в програмному забезпеченні для обробки даних для зберігання даних, таких як фінансові дані та статистичні дані.

SQLite має ряд переваг, які роблять її привабливою для розробників програмного забезпечення:

- Малий розмір: SQLite займає лише кілька мегабайтів пам'яті, що робить її ідеальною для використання в пристроях з обмеженими ресурсами.
- Швидкість: SQLite є дуже швидкою базою даних, що може бути важливою для додатків, які вимагають швидкого доступу до даних.
- Надійність: SQLite є дуже надійною базою даних, що може бути важливою для додатків, які містять важливі дані.
- Підтримка SQL: SQLite підтримує більшість основних функцій SQL, що робить її легкою для використання розробниками, які знайомі з SQL.

SQLite також має ряд недоліків, які слід враховувати перед використанням її в проекті:

- Максимальний розмір бази даних: Максимальний розмір бази даних SQLite обмежений розміром файлу, в якому вона зберігається. Це може бути проблемою для додатків, які потребують бази даних великого розміру.
- Безпека: SQLite не має вбудованих функцій безпеки. Розробники програмного забезпечення повинні самостійно забезпечити безпеку даних у базі даних SQLite.

2.5 Typed js

Svelte typed JS - це розширення для Svelte, яке додає тип безпеки до Svelte компонентів. Це робить код більш безпечним і економить час на розробку.

Svelte typed JS працює, додаючи типи до Svelte компонентів. Це можна зробити вручну, або використовуючи інструменти автоматичного генерування типів.

Щоб написати типізований Svelte компонент, необхідно додати типи до всіх змінних і функцій у компоненті. Наприклад, наступний компонент має тип `{ name: string }`:

HTML

```
<script lang="ts">
  export let name: string;
</script>

<div>
  Привіт, {{ name }}!
</div>
```

Існує кілька інструментів, які можна використовувати для автоматичного генерування типів для Svelte компонентів. Один із таких інструментів - `svelte-check`. `svelte-check` аналізує код Svelte і генерує типи для всіх компонентів.

Svelte typed JS має ряд переваг, включаючи:

- Покращена безпека: Типи допомагають запобігти помилкам, пов'язаним із типом даних. Наприклад, якщо ви намагаєтеся призначити числове значення змінній типу `string`, Svelte typed JS повідомить про помилку.
- Покращена продуктивність: Типи можуть допомогти оптимізувати продуктивність коду, оскільки компілятор може використовувати типи для більш ефективного генерування коду.
- Покращена читабельність: Типи можуть допомогти покращити читабельність коду, оскільки вони роблять код більш інформативним.

Svelte typed JS має деякі недоліки, включаючи:

- Додатковий код: Типи додають додатковий код до компонента, що може призвести до збільшення розміру компонента.
- Додатковий час компіляції: Типи можуть збільшити час компіляції компонента.

Загалом, Svelte typed JS - це потужний інструмент, який може покращити безпеку, продуктивність і читабельність Svelte компонентів.

2.6 SvelteKit

SvelteKit [15, 12, 1, 2, 3, 4, 5] - це фреймворк для швидкого розроблення надійних і високоефективних веб-додатків за допомогою Svelte. Якщо ви знайомі з React, SvelteKit схожий на Next.js.

SvelteKit допомагає розробляти веб-додатки, дотримуючись сучасних найкращих практик і надаючи рішення загальних проблем розробки. Він пропонує все, від базових функціональностей - наприклад, маршрутизатора, який оновлює ваш інтерфейс користувача, коли користувач натискає посилання - до більш просунутих можливостей.

- Швидкість: SvelteKit, що працює на Svelte і Vite, швидкість вбудовується в кожну щілину: швидка установка, швидка розробка, швидкі збірки, швидкі завантаження сторінок, швидка навігація.

- Розвага: Більше не потрібно витрачати дні на вирішення проблем із конфігурацією бунделера, маршрутизацією, SSR, CSP, TypeScript, налаштуваннями розгортання та іншими нудними речами.

- Гнучкість: SvelteKit може використовуватися для створення будь-якого типу веб-додатку, включаючи односторінкові додатки (SPA), багатосторінкові додатки (MPA) та гібридні додатки.

Щоб створити проект SvelteKit, можна скористатися командою `npx degit sveltejs/kit my-app`. Це створить новий проект у каталозі `my-app`.

Після створення проекту ви можете запустити його за допомогою команди `npm run dev`. Це запустить сервер на порту `5173`. Ви можете відкрити додаток у веб-браузері, перейшовши за адресою `http://localhost:5173`.

SvelteKit заснований на кількох основних принципах:

- Швидкість: SvelteKit спрямований на те, щоб бути якомога швидшим. Він використовує Svelte, який є компілятором, який генерує високоефективний JavaScript.
- Простота: SvelteKit має простий і інтуїтивно зрозумілий API. Він полегшує розробку веб-додатків за допомогою сучасних найкращих практик.
- Гнучкість: SvelteKit може використовуватися для створення будь-якого типу веб-додатку. Він надає все необхідне для створення односторінкових, багатосторінкових та гібридних додатків.

SvelteKit можна використовувати для створення будь-якого типу веб-додатку. Ось кілька прикладів того, як можна використовувати SvelteKit:

- Створення односторінкових додатків: SvelteKit ідеально підходить для створення односторінкових додатків. Він забезпечує швидке і ефективне виконання, а також простий і інтуїтивно зрозумілий API.

- Створення багатосторінкових додатків: SvelteKit також можна використовувати для створення багатосторінкових додатків. Він забезпечує підтримку маршрутизації та інших функцій, необхідних для створення багатосторінкових додатків.

- Створення гібридних додатків: SvelteKit можна використовувати для створення гібридних додатків, які працюють як в Інтернеті, так і в автономному режимі. Він забезпечує підтримку функцій, необхідних для створення гібридних додатків, таких як локальне зберігання та доступ до датчиків.

2.7 GitHub

GitHub — це вебсервіс для спільної розробки програмного забезпечення. Він базується на системі контролю версій Git і дозволяє розробникам зберігати, керувати та співпрацювати над кодом онлайн.

GitHub пропонує широкий спектр функцій, які допомагають розробникам у їхній роботі. До них належать:

- Сховища Git: GitHub дозволяє розробникам створювати та зберігати репозиторії Git. Репозиторій — це просто каталог, який містить код, файли конфігурації та інші пов'язані з проектом матеріали.
- Спільна робота: GitHub дозволяє розробникам співпрацювати над кодом в реальному часі. Це означає, що кілька розробників можуть одночасно працювати над одним і тим же проектом, не заважаючи один одному.
- Контроль версій: GitHub використовує систему контролю версій Git для відстеження змін у коді. Це дозволяє розробникам відстежувати історію своїх змін та легко повертатися до попередніх версій коду.
- Профілі та спільноти: GitHub дозволяє розробникам створювати профілі та приєднуватися до спільнот. Це допомагає розробникам знаходити інших розробників, обмінюватися ідеями та навчатися один в одного.

GitHub використовується для розробки широкого спектру програмного забезпечення, від невеликих веб-додатків до великих операційних систем. Він є популярним вибором для розробників, які працюють над відкритим вихідним кодом, а також для корпоративних розробників.

Ось деякі з переваг використання GitHub:

- Легкість використання: GitHub має простий інтерфейс, який легко освоїти.
- Масштабованість: GitHub може масштабуватися, щоб підтримувати великі команди розробників та проекти.
- Безпека: GitHub пропонує широкий спектр функцій безпеки для захисту коду від несанкціонованого доступу.

GitHub — це потужний інструмент, який може допомогти розробникам у будь-якій фазі розробки програмного забезпечення.

2.8 SvelteUI

SvelteUI - це безкоштовна та відкрита для використання бібліотека компонентів інтерфейсу користувача, написана на Svelte. Вона включає в себе понад 50+ налаштовувані компоненти, які відповідають стандартам WAI-ARIA.

SvelteUI має такі основні характеристики:

- Сумісність із SvelteKit і SSR.
- Підтримка TypeScript та типів даних, але необов'язкова.
- Всі компоненти доступні відповідно до стандартів WAI-ARIA.
- Включає темний режим, а також API для налаштування тем.
- Мінімальне використання сторонніх залежностей.
- Легкий налаштування - нульова конфігурація.

SvelteUI можна використовувати для створення різних типів веб-додатків, включаючи веб-сайти, веб-інтерфейси додатків і навіть десктопні програми.

Ось деякі з найбільш популярних компонентів SvelteUI:

- Button - кнопка
- Input - текстове поле
- Checkbox - прапорець
- RadioButton - радіокнопка
- Select - випадаючий список
- List - список
- Card - картка
- Layout - макет

SvelteUI є потужним інструментом, який може допомогти розробникам створювати красиві та доступні веб-додатки швидше та легше.

Ось кілька прикладів того, як можна використовувати SvelteUI:

- Для створення простого веб-сайту можна використовувати такі компоненти, як Button, Input і Layout.
- Для створення веб-інтерфейсу додатків можна використовувати такі компоненти, як Checkbox, RadioButton і Select.

- Для створення десктопної програми можна використовувати такі компоненти, як List і Card.

SvelteUI - це чудовий вибір для розробників, які хочуть використовувати Svelte для створення веб-додатків. Він пропонує широкий вибір компонентів, які можна використовувати для створення різноманітних типів додатків.

2.9 VITE

Vite - це інструмент для створення веб-додатків, який поєднує в собі швидкість, простоту використання та підтримку сучасних технологій. Він був створений Еваном Ю, творцем Vue.js.

Vite має такі основні характеристики:

- Швидкість. Vite використовує Rollup і esbuild для швидкого компіляції коду. Він може генерувати готовий до використання додаток за лічені секунди.
- Простота використання. Vite має мінімальну конфігурацію. Він готовий до використання одразу після установки.
- Підтримка сучасних технологій. Vite підтримує TypeScript, CSS Modules, SCSS, PostCSS, Webpack 5, Vue 3, React 18 та інші сучасні технології.

Vite можна використовувати для створення різних типів веб-додатків, включаючи веб-сайти, веб-інтерфейси додатків і навіть десктопні програми.

Ось кілька прикладів того, як можна використовувати Vite:

- Для створення простого веб-сайту можна використовувати такі технології, як HTML, CSS і JavaScript.
- Для створення веб-інтерфейсу додатків можна використовувати такі технології, як Vue.js, React або Svelte.
- Для створення десктопної програми можна використовувати такі технології, як Webpack Dev Server і Electron.

Vite - це потужний інструмент, який може допомогти розробникам створювати веб-додатки швидше та легше. Він особливо підходить для розробників, які хочуть використовувати сучасні технології та отримати максимальну продуктивність.

Ось кілька додаткових деталей про Vite:

- Він використовує Rollup і esbuild для швидкого компіляції коду. Rollup і esbuild - це сучасні інструменти для об'єднання модулів JavaScript. Вони дозволяють Vite компілювати код у кілька разів швидше, ніж традиційні інструменти, такі як Webpack.
- Він має мінімальну конфігурацію. Vite готовий до використання одразу після установки. Потрібно лише створити новий проект за допомогою команди vite create.
- Він підтримує TypeScript, CSS Modules, SCSS, PostCSS, Webpack 5, Vue 3, React 18 та інші сучасні технології. Vite підтримує широкий спектр сучасних технологій, що робить його ідеальним вибором для розробників, які хочуть використовувати найновіші можливості.

Vite - це швидко зростаючий інструмент, який стає все більш популярним серед веб-розробників. Він пропонує ряд переваг порівняно з традиційними інструментами, такими як Webpack, зокрема швидкість, простоту використання та підтримку сучасних технологій.

2.10 Prettier

Prettier – це інструмент для форматування коду, який використовує статичний аналіз для автоматичного форматування коду відповідно до встановлених правил. Він підтримує широкий спектр мов програмування, включаючи JavaScript, TypeScript, Python, CSS, HTML та інші.

Prettier має ряд переваг, у тому числі:

- Збільшує читабельність коду. Форматований код легше читати та розуміти, що може допомогти розробникам швидше знаходити помилки та розбиратися в коді інших людей.
- Збільшує узгодженість коду. Prettier використовує однакові правила форматування для всього коду, що може допомогти покращити загальний вигляд і відчуття проекту.

- Може бути автоматизовано. Prettier можна інтегрувати з багатьма редакторами коду, щоб форматування коду відбувалося автоматично при збереженні файлів.

Prettier можна використовувати як окрему програму або через плагіни для редакторів коду. Щоб використовувати Prettier як окрему програму, потрібно виконати наступні дії:

1. Встановіть Prettier.
2. Виберіть файли, які потрібно відформатувати.
3. Запустіть Prettier.

Щоб використовувати Prettier через плагін для редактора коду, потрібно виконати наступні дії:

1. Встановіть плагін для редактора коду, який підтримує Prettier.
2. Налаштуйте плагін, щоб він відповідав вашим потребам.
3. Форматуйте код за допомогою плагіна.

Налаштування Prettier можна виконувати за допомогою файлу конфігурації. Файл конфігурації Prettier називається `.prettierrc` і може зберігатися в кореневому каталозі проекту або в каталозі кожного файлу. Файл конфігурації Prettier містить список правил форматування, які потрібно використовувати.

Приклад файлу конфігурації Prettier:

```
JSON
{
  "tabWidth": 2,
  "useTabs": false,
  "semi": true,
  "trailingComma": "all",
  "bracketSpacing": true,
  "arrowParens": "always"
}
```

Цей файл конфігурації встановлює такі правила:

- Відступи розміром 2 символи.

- Не використовувати табуляції.
- Використовувати крапки з комою в кінці операторів.
- Використовувати пробіли між дужками.
- Використовувати пари дужок навколо стрілочних функцій.

Prettier – це потужний інструмент, який може допомогти покращити читабельність, узгодженість та загальний вигляд вашого коду. Завдяки простоті використання та широкому спектру підтримуваних мов Prettier є чудовим вибором для будь-якого розробника.

2.11 Docker

Docker — це платформа контейнеризації, яка дозволяє розробникам створювати, запускати та масштабувати додатки в ізольованих середовищах, які називаються контейнерами. Контейнери — це невеликі, самодостатні програми, які включають в себе все необхідне для їх роботи, включаючи операційну систему, бібліотеки, код та дані.

Docker працює за допомогою двох основних компонентів:

- Docker Engine — це програмне забезпечення, яке забезпечує основну функціональність Docker, включаючи створення, запуск, зупинку та видалення контейнерів.
- Dockerfile — це текстовий файл, який використовується для опису того, як створювати контейнери. Dockerfile містить список команд, які повинні виконуватися для створення контейнера, включаючи установку програмного забезпечення, копіювання файлів та налаштування конфігурації.

Docker пропонує ряд переваг перед традиційною моделлю віртуалізації, яка використовує гіпервізори для створення віртуальних машин. Контейнери більш економні, оскільки вони не потребують власної операційної системи. Вони також більш мобільні, оскільки можуть запускатися на будь-якому хості з встановленим Docker Engine.

Docker широко використовується в розробці програмного забезпечення, оскільки він дозволяє розробникам швидко та легко створювати та запускати

додатки. Docker також використовується в операційних системах, оскільки він дозволяє легко розгортати та масштабувати додатки.

Ось деякі з основних переваг використання Docker:

- Економія ресурсів: Контейнери більш економні, оскільки вони не потребують власної операційної системи. Це може призвести до значних заощаджень на витратах на хостинг і електроенергію.
- Мобільність: Контейнери можуть запускатися на будь-якому хості з встановленим Docker Engine. Це робить їх ідеальними для хмарних інфраструктур.
- Швидкість і простота: Docker дозволяє швидко та легко створювати, запускати та масштабувати додатки. Це може значно скоротити час розробки та розгортання.
- Безпека: Контейнери ізольовані один від одного, що допомагає захистити додатки від взломів.

Docker є потужним інструментом, який може допомогти розробникам та операційним системам ефективніше створювати, запускати та масштабувати додатки.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Розробка моделі представлення даних

Для роботи з даними в реляційній системі бази даних слід описати їх моделі. Це було виконано в графічному редакторі що надається pocketbase. В таблиці barber визначені поля (рис. 3.1) : ім’я, фото, рівень, електронна пошта, зв’язок з таблицею послуг, які надає конкретно цей працівник, ставка в відсотковому форматі від базової, час в який працює.

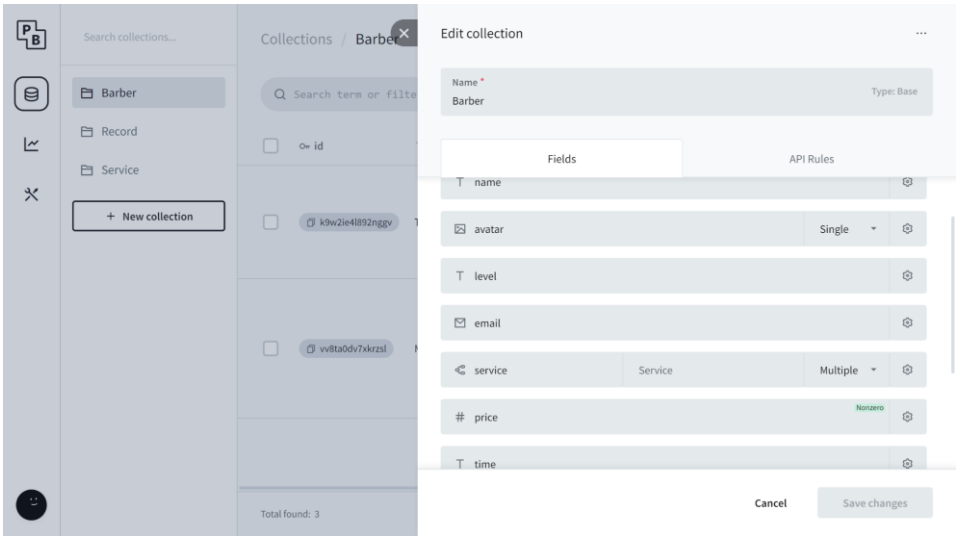


Рисунок 3.1 – Модель даних barber

Таблиця сервіси (рис. 3.2) має поля назва послуги та базова вартість.

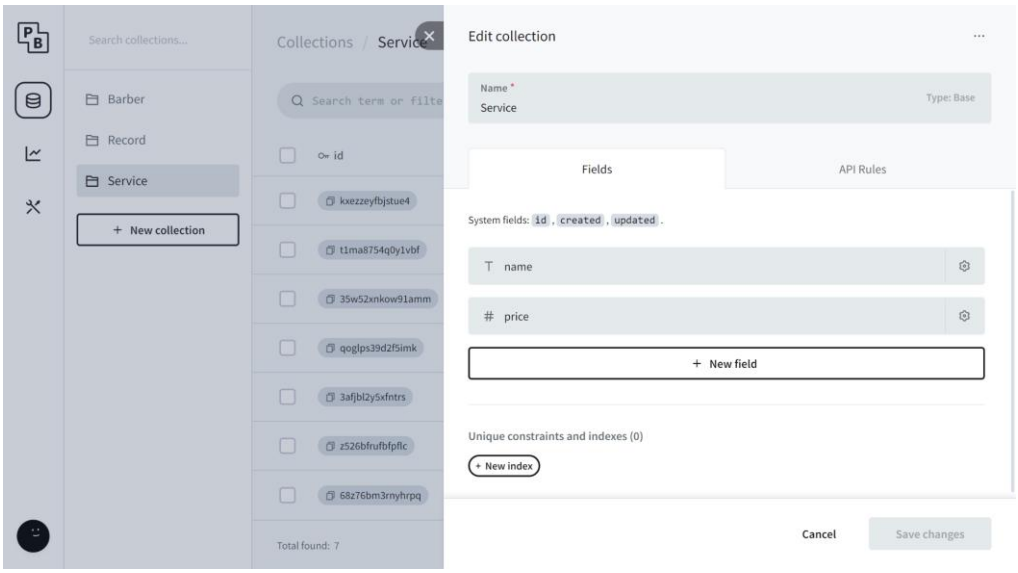


Рисунок 3.2 – Таблиця послуг

І, звісно, таблиця що містить записи клієнтів (рис. 3.3). Містить поля: час запису, зв’язок з послугою, зв’язок з працівником, ім’я клієнта, електронна пошта клієнта.

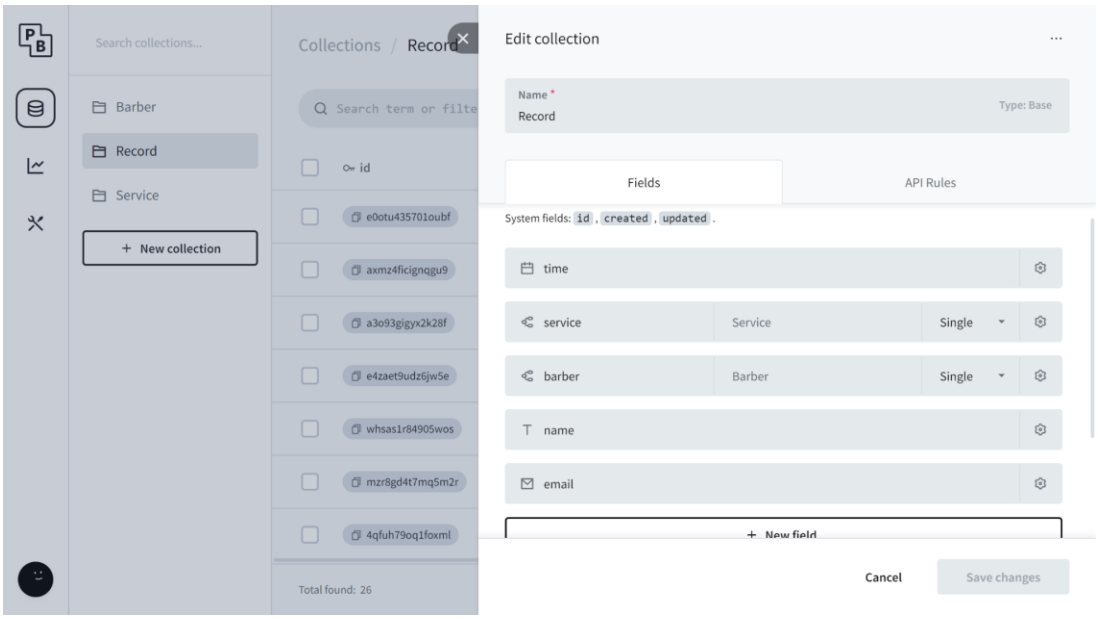


Рисунок 3.3 – Таблиця записів

3.2 Серверна взаємодія

Оскільки некоректно доступатися до бази дани з клієнтського застосунку – робота з бд відбувається на стороні сервера в апі обробниках та серверній логіці сторінок (рис. 3.4).

```
import pb from '$lib/pocketbase.js';

/** @type {import('./$types').LayoutServerLoad} */
export const load = async ({ cookies }) => {
    const service = JSON.parse(cookies.get('service') || '{}');
    const datetime = cookies.get('datetime');
    const barbers = await pb.collection('Barber').getFullList({
        filter: service.id ? `service ?~ "${service.id}" && level =
"${decodeURIComponent(escape(service.level))}"` : '',
    });
    return { barbers };
};
```

Рисунок 3.4 – Отримання даних з бд для відображення

Оскільки sveltekit підтримує directory based маршрутизацію, кожна директорія в шляху /routes – обробляється як сторінка. Вона повинна містити +page.svelte, що відповідає, власне, за розмітку, а також, в силу того що підтримується ssr – функцію завантаження вмісту на стороні сервера. Це описується в вищенаведеному +page.server.svelte. З допомогою клієнта pocketbase робиться запит з параметрами фільтрації для отримання даних. Вони будуть доступні на сторінці в експортованій змінній data (рис. 3.5)

```svelte
<script>
    import { Center, Paper } from '@svelteuidev/core';
    import { goto } from '$app/navigation';

    /** @type {import('./$types').LayoutServerLoad} */
    export let data;
</script>
```

Рисунок 3.5 – Отримання серверних даних на сторінці

Ідентично описуються і апі обробники. Вони розміщуються в відповідній директорії в файлі +server.js (рис. 3.6).

```js
import pb from '$lib/pocketbase.js';

/** @type {import('./$types').RequestHandler} */
export async function GET({ url }) {
    const record = url.searchParams.get('record');
    const res = await pb.collection('Record').delete(record);
    if (res) {
        return new Response('Запис скасовано');
    }
    return new Response('Щось пішло не так');
}
```

Рисунок 3.6 – Апі обробник

Docker налаштовано на розгортання pocketbase в одному контейнері з застосунком, але без доступу до мережі. Це ізолює базу даних від можливого доступу ззовні. Але це і означає що нема змоги отримувати дані на стороні клієнта. Якщо це вирішується з одного боку ssr, то з іншого є медіаресурси, що передаються як url посилання на ресурс, що також надається pocketbase. Зрозуміло, що використати їх на стороні клієнта безпосередньо ми їх не можемо. Для цього був написаний проксі для отримання зображення з бд і пересилання на зовнішню адресу (рис. 3.7).

```
import { PB } from '$env/static/private';

/** @type {import('./$types').RequestHandler} */
export async function GET({ url }) {
    const collection = url.searchParams.get('collection');
    const id = url.searchParams.get('id');
    const file = url.searchParams.get('file');
    try {
        const attachment = await fetch(`${PB}/api/files/${collection}/${id}/${file}?thumb=150x150`);

        return new Response(await attachment.arrayBuffer(), {
            headers: {
                'Content-Type': 'image/jpeg'
            }
        });
    } catch (error) {
        console.log(error);
    }
}
```

Рисунок 3.7 – Проксі для медіафайлів

Окремо слід описати створення запису. Після валідації на стороні клієнта – сформований об’єкт приходить на сервер (рис. 3.8). Створюється запис, якщо є помилка – вона повертається на сайт. в іншому випадку – користувачу, на вказану адресу електронної пошти надсилається лист з деталями запису: часом, видом послуги, згенерованим посиланням для додання в google calendar, та посиланням для скасування запису.

```
import { Resend } from 'resend';
import pb from '$lib/pocketbase.js';

const resend = new Resend('re_FUqFckcZ_6WbAuGBenVPkq7QJeAPU6gBT');

function generateTimeRange(startDate) {
    const startDateTime = new Date(startDate);

    if (isNaN(startDateTime.getTime())) {
        console.error('Invalid date format');
        return null;
    }

    const endDateTime = new Date(startDateTime.getTime() + 60 * 60 * 1000);

    const formattedStartDate = startDateTime.toISOString().slice(0, -1);
    const formattedEndDate = endDateTime.toISOString().slice(0, -1);

    return `${formattedStartDate}/${formattedEndDate}`;
}


/** @type {import('./$types').RequestHandler} */export async function GET({ cookies }) {
    const barber = JSON.parse(cookies.get('barber') || '{}');
    const service = JSON.parse(cookies.get('service') || '{}');
    const datetime = cookies.get('datetime') || '';
    const user = JSON.parse(cookies.get('user') || '{}');

    const record = await pb.collection('Record').create({
        'time': datetime,
        'service': service.id,
        'barber': barber.id,
        'name': user.name,
        'email': user.email
    });

    if (!record.id) {
        return new Response(JSON.stringify(record));
    }

    const { data, error } = await resend.emails.send({
        from: 'RazorBeard <manager@razorbeard.site>',
        to: [user.email],
        subject: 'Record',
        html: `
        <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<meta http-equiv="Content-Type" content="text/html charset=UTF-8" />
weight:400;color:#484848;padding:17px 0 0"><h1>Вітаємо ${decodeURIComponent(escape(user.name))}!</h1>
<br/>Ваш запис на ${decodeURIComponent(escape(service.name))} в RazorBeard</h1>
          <table style="padding:27px 0 27px" align="center" border="0" cellPadding="0" cellSpacing="0"
role="presentation" width="100%">
            <tbody>
              <tr>
                <td><a href="https://calendar.google.com/calendar/u/0/r/eventedit?
text=${decodeURIComponent(escape(service.name))}+в+RazorBeard&details=${decodeURIComponent(escape(serv
ice.name))}+в+RazorBeard&${generateTimeRange(datetime)}&ctz=Ukraine/Lviv" target="_blank"
style="background-color:#d25e5e;border-radius:3px;font-weight:600;color:#fff;font-size:15px;text-
decoration:none;text-align:center;display:inline-block;p-x:23px;p-y:11px;line-height:100%;max-
width:100%;padding:11px 23px"></a></td>
              </tr>
            </tbody>
          </table>
          <p style="font-size:15px;line-height:1.4;margin:0 0 15px;color:#3c4149">Чекаємо Вас:</p><p
style="font-family:monospace;font-weight:700;padding:1px 4px;background-color:#dfe1e4;letter-
spacing:-0.3px;font-size:21px;border-radius:4px;color:#3c4149;text-align:
center">${datetime.split(':00.')[0]}</p>
          <a href="https://barbershop.razorbeard.site/api/cancel?record=${record.id}">Скасувати
запис</a>
          <hr style="width:100%;border:none;border-top:1px solid #eaeaea;border-
color:#dfe1e4;margin:42px 0 26px" /><a target="_blank" style="color:#b4becc;text-decoration:none;font-
size:14px" href="https://linear.app">RazorBeard</a>
        </td>
      </tr>
    </table>
  </body>

</html>
        `
    });

    if (error) {
        return new Response(JSON.stringify(error));
    }

    return new Response(JSON.stringify({ data, record }));
}
```

Рисунок 3.8 – Оформлення запису

3.3 Розробка клієнтського застосунку

Клієнтський застосунок розроблявся з використанням бібліотеки Svelte. Графічний інтерфейс його складається з декількох сторінок навігація між якими описана структурою директорій (рис. 3.9).

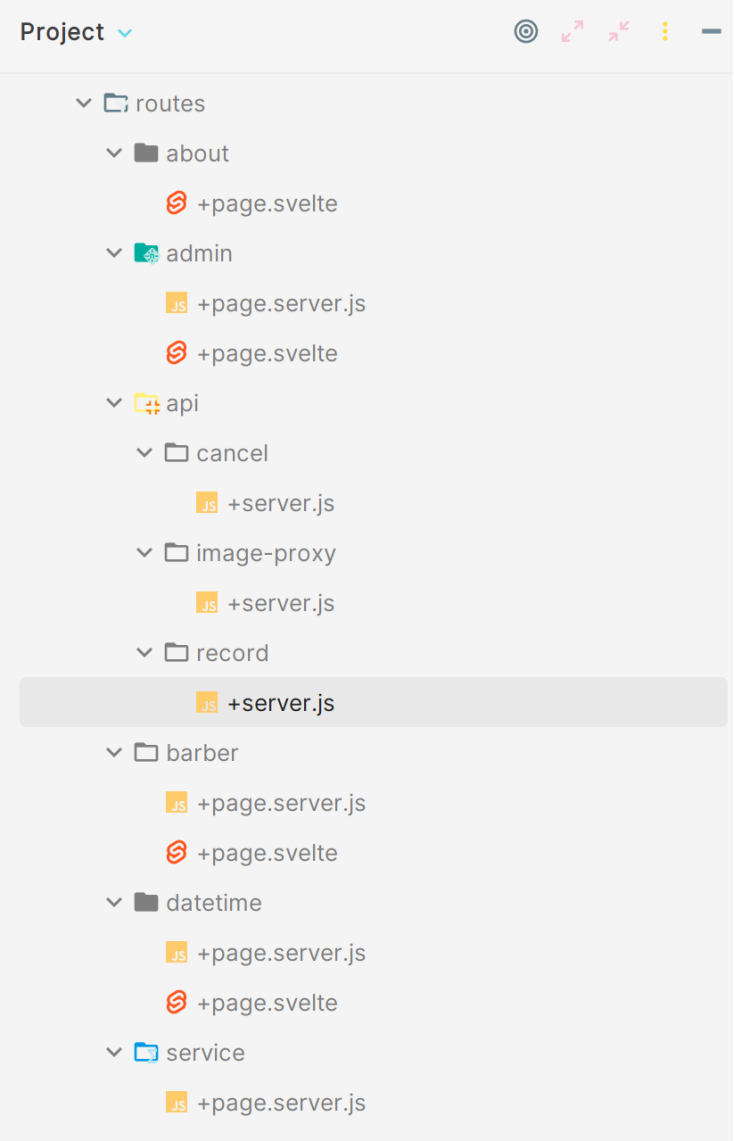
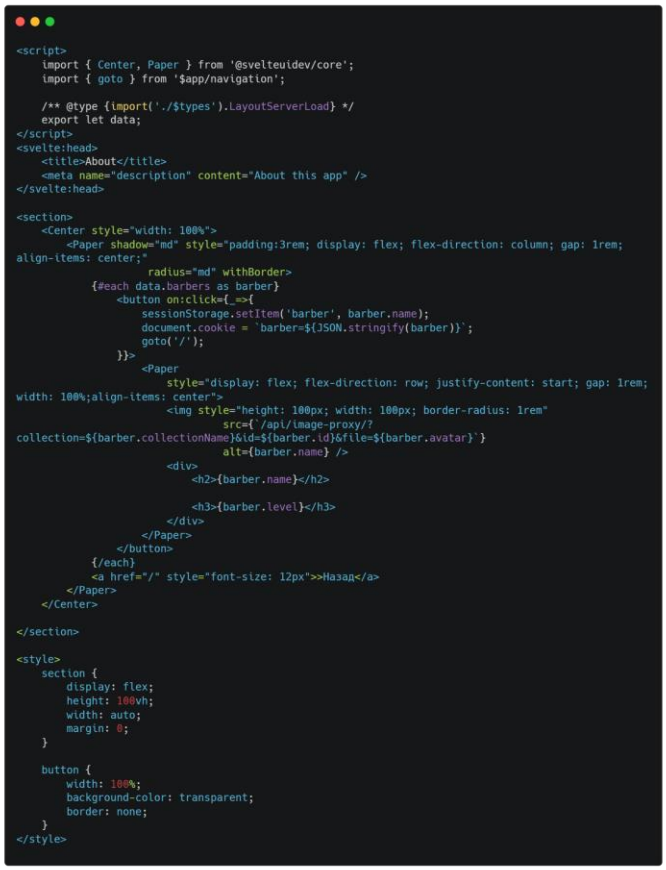


Рисунок 3.9 – Структура навігації застосунку

Структура сторінки svelte схожа на звичайну html розмітку та скрипти та відмінність полягає в вбудованій реактивності, що дозволяє мутацією значення змінних – миттєво змінювати шаблон, описаний html, інтерполяцією, та власними директивами svelte (рис. 3.10).



```
<script>
    import { Center, Paper } from '@svelteuidev/core';
    import { goto } from '$app/navigation';

    /** @type {import('./$types').LayoutServerLoad} */
    export let data;
</script>
<svelte:head>
    <title>About</title>
    <meta name="description" content="About this app" />
</svelte:head>

<section>
    <Center style="width: 100%">
        <Paper shadow="md" style="padding:3rem; display: flex; flex-direction: column; gap: 1rem;
align-items: center;"
                    radius="md" withBorder>
            {#each data.barbers as barber}
                <button on:click={_=>{
                    sessionStorage.setItem('barber', barber.name);
                    document.cookie = `barber=${JSON.stringify(barber)}`;
                    goto('/');
                }}>
                    <Paper
                        style="display: flex; flex-direction: row; justify-content: start; gap: 1rem;
width: 100%;align-items: center">
                        <img style="height: 100px; width: 100px; border-radius: 1rem"
                                src={`/api/image-proxy/?
collection=${barber.collectionName}&id=${barber.id}&file=${barber.avatar}`}
                                alt={barber.name} />
                        <div>
                            <h2>{barber.name}</h2>

                            <h3>{barber.level}</h3>
                        </div>
                    </Paper>
                </button>
            {/each}
            <a href="/" style="font-size: 12px">>Назад</a>
        </Paper>
    </Center>

</section>

<style>
    section {
        display: flex;
        height: 100vh;
        width: auto;
        margin: 0;
    }

    button {
        width: 100%;
        background-color: transparent;
        border: none;
    }
</style>
```

Рисунок 3.10 – Розмітка svelte сторінки

Svelte використовує компонентний підхід до розробки веб-додатків. Компоненти - це самостійні блоки коду, які відповідають за певну частину інтерфейсу користувача. Вони можуть бути повторно використані в різних місцях додатку. Тут використано такі з бібліотеки SvelteUI.

Компоненти Svelte створюються за допомогою директив. Директиви - це спеціальні маркери, які додають певну поведінку до компонента. Наприклад, директива <script> дозволяє додати код JavaScript до компонента, а директива <style> дозволяє додати стилі до компонента.

Svelte використовує реактивну анімацію для створення інтерактивних елементів інтерфейсу користувача. Реактивність означає, що компоненти автоматично оновлюються, коли змінюється їхнє стан.

Стан компонента може бути змінений за допомогою директив <input>, <select> та інших. Коли стан компонента змінюється, Svelte автоматично оновлює компонент, щоб відображати зміни.

Svelte компресує код, що генерується, під час компіляції. Це робить додатки Svelte меншими за розміром і швидшими завантаженням.

Компресія Svelte відбувається за допомогою алгоритму, який називається Rollup. Rollup об'єднує всі файли, які використовуються в додатку, в один файл. Це дозволяє зменшити кількість запитів HTTP, які потрібно зробити для завантаження додатку.

Svelte розроблений з урахуванням ефективності. Він не використовує DOM-функції, такі як appendChild() та removeChild(). Замість цього Svelte використовує власні механізми, які є більш ефективними.

Використовується техніка, яка називається "virtual DOM". Virtual DOM - це віртуальна модель документа, яка використовується для відображення інтерфейсу користувача. Svelte оновлює virtual DOM, коли стан компонента змінюється. Потім Svelte порівнює virtual DOM із фактичним DOM і вносить необхідні зміни. Це дозволяє Svelte оновлювати інтерфейс користувача тільки тоді, коли це необхідно.

3.4 Аналіз контрольного прикладу

Далі подано опис роботи фінальної версії застосунку (рис. 3.11).

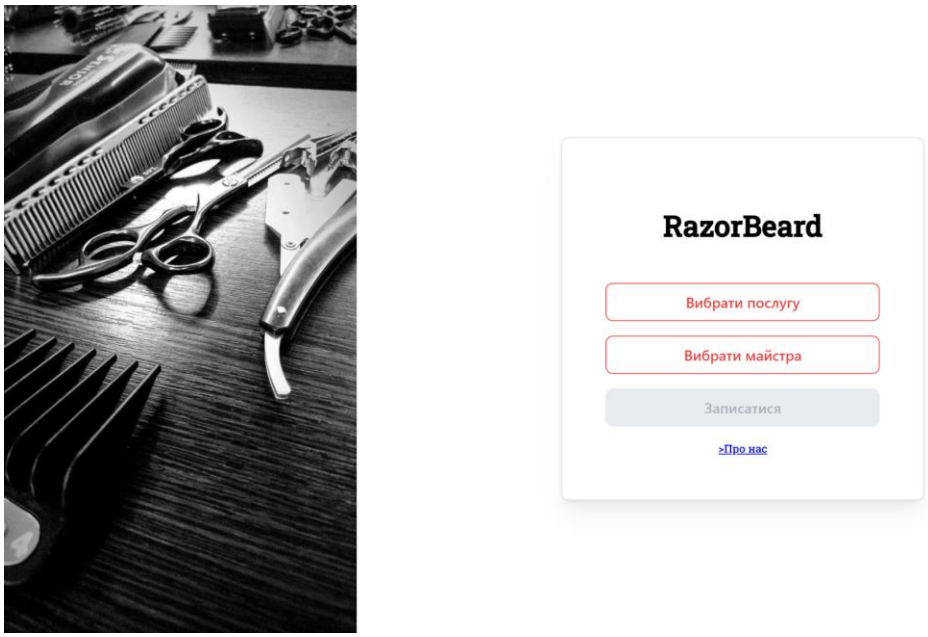


Рисунок 3.11 – Головна сторінка

Після переходу на сторінку послуг – можна обрати послугу від різних майстрів (рис. 3.12)

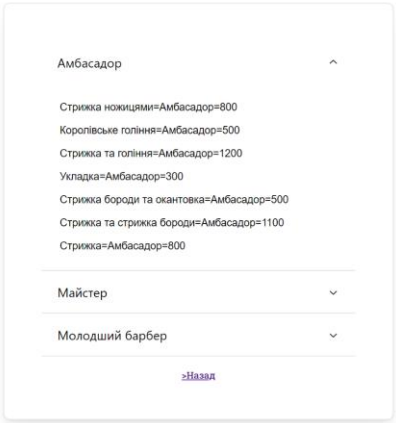


Рисунок 3.12 – Сторінка обрання послуги

Та в разі якщо спочатку був обраний барбер (рис. 3.13),

Рисунок 3.13 – Сторінка вибору майстра

то на сторінці послуг – відображатимуться тільки послуги для обраного майстра (рис. 3.14)

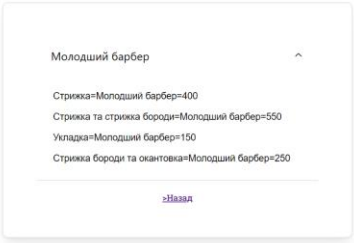


Рисунок 3.14 – Сторінка послуг після обрання майстра

Кожен вибір можна скасувати на головній сторінці (рис. 3.15).

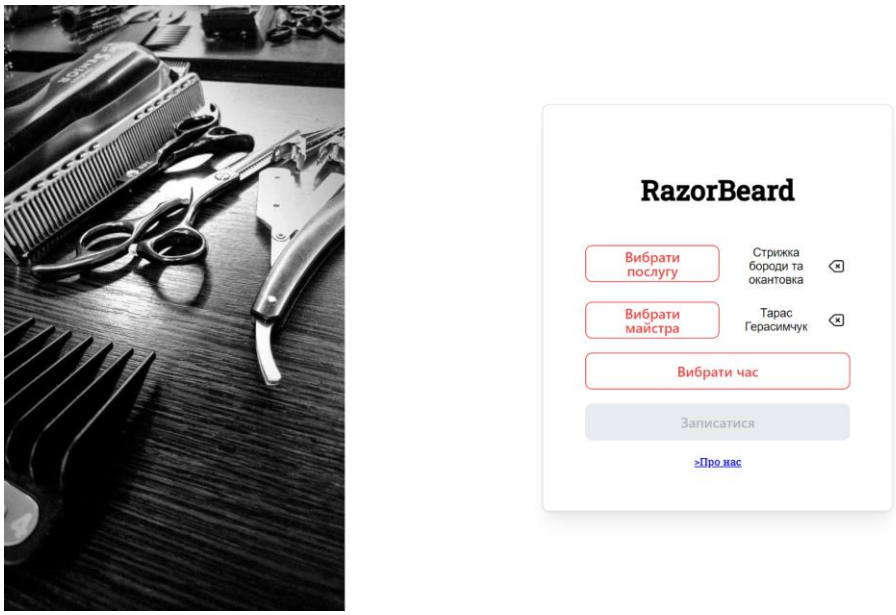


Рисунок 3.15 – Можливість скасування вибору

Після обрання обох пунктів – відкривається вибір часу (рис. 3.16).

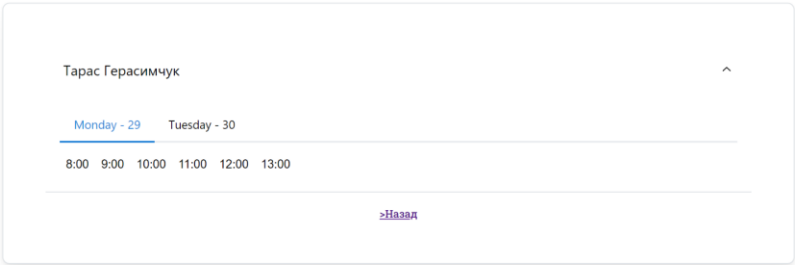


Рисунок 3.16 – Вибір дати й часу

Після введення імені й адреси електронної пошти – можна створити запис (рис. 3.17).

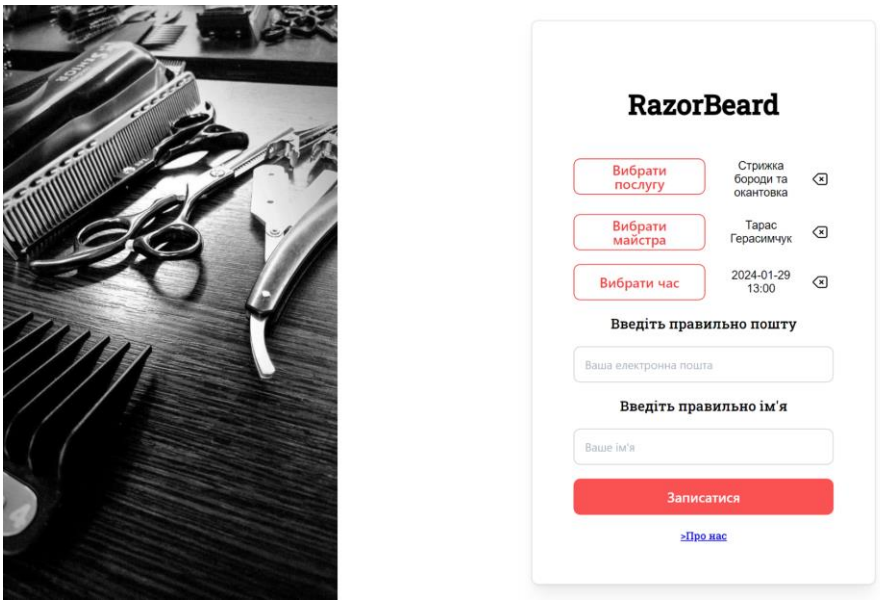


Рисунок 3.17 – Можливість створення запису

Дані про запис прийдуть на вказану скриньку (рис. 3.18)

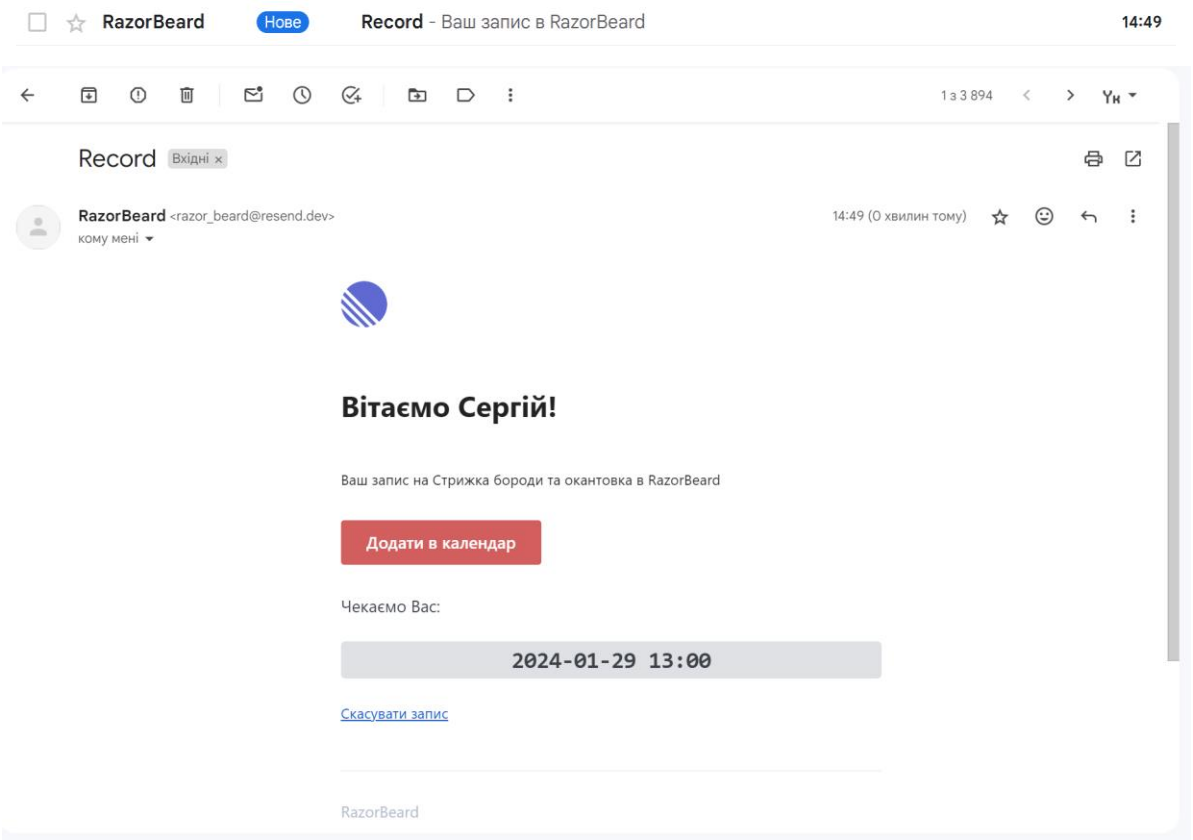


Рисунок 3.18 – Лист з інформацією

В ньому ж є посилання для скасування запису.

3.6 Характеристики апаратного та програмного забезпечення

IntelliJ IDEA - це інтегроване середовище розробки (IDE) для розробки програмного забезпечення, розроблене компанією JetBrains. Воно підтримує широкий спектр мов програмування, включаючи Java, Kotlin, Python, JavaScript, PHP, C/C++, Go, Ruby та інші.

IntelliJ IDEA має широкий спектр функцій, які полегшують і прискорюють розробку програмного забезпечення. До них відносяться:

- Автодоповнення коду - IntelliJ IDEA пропонує автодоповнення коду, яке допомагає розробникам швидко знайти потрібний код.
- Інтелектуальна навігація - IntelliJ IDEA пропонує інтелектуальну навігацію, яка дозволяє розробникам швидко переміщатися по коду.
- Зневадження - IntelliJ IDEA пропонує потужні засоби зневадження, які допомагають розробникам знаходити і виправляти помилки в коді.
- Тести - IntelliJ IDEA пропонує засоби для написання, запуску та відлагодження тестів.
- Розробка веб-додатків - IntelliJ IDEA пропонує засоби для розробки веб-додатків, включаючи підтримку JavaScript, HTML, CSS та інших технологій.

IntelliJ IDEA доступна в декількох випусках:

- Community Edition - безкоштовна версія, яка підтримує Java, Kotlin, Python та JavaScript.
- Ultimate Edition - платна версія, яка підтримує всі мови програмування, а також додаткові функції, такі як управління проектами, рефакторинг коду та інше.

IntelliJ IDEA є одним з найпопулярніших IDE для розробки програмного забезпечення. Вона використовується багатьма компаніями та розробниками в усьому світі.

Ось деякі з основних переваг IntelliJ IDEA:

- продуктивність: IntelliJ IDEA пропонує широкий спектр функцій, які допомагають розробникам швидко і ефективно писати код.

- інтуїтивність: IntelliJ IDEA має простий і інтуїтивно зрозумілий інтерфейс.
- адаптивна: IntelliJ IDEA адаптується до індивідуальних потреб розробників.

IntelliJ IDEA [11] - це потужний і ефективний інструмент для розробки програмного забезпечення. Вона пропонує широкий спектр функцій, які допомагають розробникам швидко і ефективно писати код.

Docker Desktop - це безкоштовна програма для розробників, яка дозволяє створювати, запускати та обмінюватися контейнеризованими додатками та мікросервісами. Вона доступна для macOS, Linux та Windows.

Docker Desktop працює з вашим вибором інструментів розробки та мов і надає доступ до величезної бібліотеки сертифікованих образів та шаблонів у Docker Hub. Це дозволяє розробницьким командам розширити свою середовище для швидкого автоматичного створення, безперервної інтеграції та співпраці за допомогою надійного сховища.

Основні можливості Docker Desktop включають:

- Створення контейнерів: Docker Desktop дозволяє створювати контейнери з вашими додатками та середовищем. Контейнери - це зручний спосіб упаковки ваших додатків, включаючи всі їхні залежності, в ізольоване середовище. Це дозволяє запускати свої додатки на будь-якому комп'ютері з Docker, незалежно від його конфігурації.
- Запуск контейнерів: Docker Desktop дозволяє запускати контейнери на вашому локальному комп'ютері. Це дозволяє тестувати та запускати свої додатки локально, перш ніж розгортати їх у виробництво.
- Обмін контейнерами: Docker Desktop дозволяє ділитися контейнерами з іншими розробниками. Це дозволяє співпрацювати над проектами та легко розгортати додатки в production.

Docker Desktop [10] - це потужний інструмент, який може допомогти розробникам швидко та легко створювати, запускати та обмінюватися контейнеризованими додатками та мікросервісами.

Ось кілька прикладів того, як можна використовувати Docker Desktop:

- Можна використовувати Docker Desktop для створення веб-додатків, які працюють на будь-якому комп'ютері з Docker.
- Можна використовувати Docker Desktop для створення серверних додатків, таких як веб-сервери, бази даних та API.
- Можна використовувати Docker Desktop для створення мікросервісів, які можуть легко взаємодіяти один з одним.

Docker Desktop - це хороший вибір для розробників, які хочуть:

- Створювати контейнеризовані додатки та мікросервіси
- Запускати контейнеризовані додатки локально
- Обмінюватися контейнеризованими додатками з іншими розробниками

ВИСНОВКИ

Метою виконання дипломного проекту була розробка та реалізація crm системи для організації бізнес процесів барбершопу засобами sveltekit, бекенду pocketbase та компонентів графічного інтерфейсу що надає пакет svelte-ui.

Розроблений програмний продукт був протестований за допомогою мануального тестування і перевірений у відповідності до поставлених у технічному завданні вимог.

В результаті можна сказати що застосунок повністю відповідає поставленому завданню і показує себе як надійне рішення для бізнесу в сфері надання послуг.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Volkmann M. Svelte and Sapper in Action, Manning Publications, 2020. – 456 ст.
2. Libby A. Practical Svelte. – Birmingham: Packt Publishing, 2022. – 325 ст.
3. Hildenbrand D. SvelteKit Up and Running, O'Reilly Media, 2023. – 142 ст.
4. Li Hau T. Real-World Svelte, Packt Publishing, 2023. – 180 ст.
5. Lim G. Beginning Svelte Development, Independently published, 2022. – 146 ст.
6. Molinaro A. SQL Cookbook, O'Reilly Media, 2020. – 572 ст.
7. Silva R. MySQL Crash Course, Packt Publishing, 2023. – 646 ст.
8. Faroult S. The Art of SQL, O'Reilly Media, 2006. – 367 ст.
9. SQLite. [Електронний ресурс] – URL: https://www.sqlite.org/index.html (дата звернення: 15.04.2024). 10. Docker. [Електронний ресурс] – URL: https://www.docker.com/ (дата звернення: 15.04.2024).
11. JetBrains IntelliJ IDEA. [Електронний ресурс] – URL: https://www.jetbrains.com/idea/ (дата звернення: 15.04.2024).
12. SvelteKit. [Електронний ресурс] – URL: https://kit.svelte.dev/ (дата звернення: 15.04.2024).
13. PocketBase. [Електронний ресурс] – URL: https://pocketbase.io/ (дата звернення: 15.04.2024).
14. Resend. [Електронний ресурс] – URL: https://resend.com/ (дата звернення: 15.04.2024).
15. Svelte. [Електронний ресурс] – URL: https://svelte.dev/ (дата звернення: 15.04.2024).

ДОДАТКИ

```
import PocketBase from 'pocketbase';
import { PB } from '$env/static/private';
export default new PocketBase(PB || 'http://localhost:8090')

<script>
 import { Center, Paper } from '@svelteuidev/core';
</script>
<svelte:head>
 <title>About</title>
 <meta name="description" content="About this app" />
</svelte:head>

<section>
 <Center style="width: 100%">
  <Paper shadow="md"
      style="padding:3rem; max-width: 80vw;display: flex; flex-direction: column;
gap: 1rem; align-items: center;"
      radius="md" withBorder>
   <p>
    <b>RazorBeard</b> - це барбершоп, де мистецтво стрижки та догляду за бородою
поєднуються з атмосферою чоловічої
    елегантності. У цьому закладі присутній справжній професіоналізм, що дозволяє
клієнтам отримати відмінний сервіс
    та бездоганний вигляд.
   </p>
   <p>
    Зручно розташований в центрі міста, "RazorBeard" вітає своїх клієнтів
    сучасним та стильним інтер'єром. Кожен візит - це спеціальний момент, де кожен
клієнт може розслабитися і
    насолоджуватися неперевершеним досвідом стрижки від висококваліфікованих
фахівців.
   </p>
   <p>
    Експерти "RazorBeard" відзначаються не лише вмінням вирізняти та створювати
найкращі стрижки, але й здатністю
    зробити процес комфортним та приємним для кожного клієнта. Вони розуміють
індивідуальні особливості кожного,
    надаючи персоналізований підхід до стрижки та догляду за бородою.
   </p>
   <p>
```

```
    Незалежно від того, чи ви шукаєте класичний стрижку, модне оформлення бороди чи стильну укладку, "RazorBeard"
    гарантує вам високий стандарт обслуговування та залишить вас враженими власною стильністю та свіжістю.
   </p>
   <a href="/" style="font-size: 12px">>Назад</a>
  </Paper>
 </Center>

</section>

<style>
   section {
       display: flex;
       height: 100vh;
       width: auto;
       margin: 0;
   }
</style>

import { redirect } from '@sveltejs/kit';
import pb from '$lib/pocketbase.js';

/** @type {import('./$types').LayoutServerLoad} */
export const load = async ({ url }) => {
 if (url.searchParams.get('token') !== 'auth') {
  return redirect(302, '/');
 }
 const records = await pb.collection('Record').getFullList({
  filter: '', expand: 'barber,service'
 });
 const barbers = await pb.collection('Barber').getFullList({
  filter: '', expand: 'service'
 });
 const services = await pb.collection('Service').getFullList({
  filter: '', expand: ''
 });
 return { records, barbers, services };
};

<script>
 import { Tabs, Center, Paper } from '@svelteuidev/core';
 import { SvelteUIProvider } from '@svelteuidev/core';
```

```
 /** @type {import('./$types').LayoutServerLoad} */
 export let data;

 function filterFutureObjects(objectList) {
  const currentTime = new Date();
  return objectList.filter(function(object) {
   const objectTime = new Date(object.time);
   return objectTime > currentTime;
  });
 }
</script>
<svelte:head>
 <title>About</title>
 <meta name="description" content="About this app" />
</svelte:head>

<SvelteUIProvider>
 <section>
  <Center style="width: 100%">
   <Paper shadow="md"
       style="padding:3rem;flex-direction: column; gap: 1rem; align-items: center;"
       radius="md" withBorder>
    <Tabs>
     <Tabs.Tab label='Записи'>
      <div style=" display: grid;grid-template-columns: auto auto; gap:1rem;">
       {#each filterFutureObjects(data.records) as record}
        <Paper
         style="display: flex; flex-direction: row; justify-content: start; gap:
1rem; width: 100%;align-items: center">
         <div>
          <h2>{record.expand.barber.name}</h2>
          <h3>{record.expand.barber.level}</h3>
         </div>
         <div style="display: flex;flex-direction: column">
          <i>{record.expand.service.name}</i>
          <i>{record.expand.barber.time}</i>
          <i>{record.expand.barber.email}</i>
         </div>
        </Paper>
       {/each}
      </div>
     </Tabs.Tab>
     <Tabs.Tab label='Барбери'>
      <div style=" display: grid;grid-template-columns: auto auto; gap: 1rem;">
```

```
       {#each data.barbers as barber}
        <Paper
         style="display: flex; flex-direction: row; justify-content: start; gap:
1rem; width: 100%;align-items: center">
         <img style="height: 100px; width: 100px; border-radius: 1rem"
            src={`/api/image-
proxy/?collection=${barber.collectionName}&id=${barber.id}&file=${barber.avatar}`}
            alt={barber.name} />
         <div>
          <h2>{barber.name}</h2>

          <h3>{barber.level}</h3>
         </div>
         <div style="display: flex;flex-direction: column">
          <i>{barber.price}</i>
          <i>{barber.time}</i>
          <i>{barber.email}</i>
         </div>
        </Paper>
       {/each}
      </div>
     </Tabs.Tab>
     <Tabs.Tab label='Сервіси'>
      <div style=" display: grid;grid-template-columns: auto auto; gap: 1rem;">
       {#each data.services as service}
        <Paper
         style="display: flex; flex-direction: row; justify-content: start; gap:
1rem; width: 100%;align-items: center">
         <div style="display: flex;flex-direction: column">
          <i>{service.name}</i>
          <i>{service.price}</i>
         </div>
        </Paper>
       {/each}
      </div>
     </Tabs.Tab>
    </Tabs>
   </Paper>
  </Center>

 </section>
</SvelteUIProvider>

<style>
```

```
   section {
       display: flex;
       padding: 1rem;
       width: auto;
       margin: 0;
   }
</style>

import pb from '$lib/pocketbase.js';

/** @type {import('./$types').RequestHandler} */export async function GET({ url }) {
 const record = url.searchParams.get('record');
 const res = await pb.collection('Record').delete(record);
 if (res) {
  return new Response('Запис скасовано');
 }
 return new Response('Щось пішло не так');
}

import { PB } from '$env/static/private';

/** @type {import('./$types').RequestHandler} */export async function GET({ url }) {
 const collection = url.searchParams.get('collection');
 const id = url.searchParams.get('id');
 const file = url.searchParams.get('file');
 try {
  const attachment = await
fetch(`${PB}/api/files/${collection}/${id}/${file}?thumb=150x150`);

  return new Response(await attachment.arrayBuffer(), {
   headers: {
    'Content-Type': 'image/jpeg'
   }
  });
 } catch (error) {
  console.log(error);
 }
}

import { Resend } from 'resend';
import pb from '$lib/pocketbase.js';

const resend = new Resend();
```

```
function generateTimeRange(startDate) {
 const startDateTime = new Date(startDate);

 if (isNaN(startDateTime.getTime())) {
  console.error('Invalid date format');
  return null;
 }

 const endDateTime = new Date(startDateTime.getTime() + 60 * 60 * 1000);

 const formattedStartDate = startDateTime.toISOString().slice(0, -1);
 const formattedEndDate = endDateTime.toISOString().slice(0, -1);

 return `${formattedStartDate}/${formattedEndDate}`;
}


/** @type {import('./$types').RequestHandler} */export async function GET({ cookies
}) {
 const barber = JSON.parse(cookies.get('barber') || '{}');
 const service = JSON.parse(cookies.get('service') || '{}');
 const datetime = cookies.get('datetime') || '';
 const user = JSON.parse(cookies.get('user') || '{}');

 const record = await pb.collection('Record').create({
  'time': datetime,
  'service': service.id,
  'barber': barber.id,
  'name': user.name,
  'email': user.email
 });

 if (!record.id) {
  return new Response(JSON.stringify(record));
 }

 const { data, error } = await resend.emails.send({
  from: 'RazorBeard <manager@razorbeard.site>',
  to: [user.email],
  subject: 'Record',
  html: `
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<meta http-equiv="Content-Type" content="text/html charset=UTF-8" />
```

```
<html lang="en">

 <div id="__react-email-preview" style="display:none;overflow:hidden;line-
height:1px;opacity:0;max-height:0;max-width:0">Ваш запис в RazorBeard<div>
                                        </div>
 </div>

 <body style="background-color:#ffffff;font-family:-apple-
system,BlinkMacSystemFont,&quot;Segoe UI&quot;,Roboto,Oxygen-
Sans,Ubuntu,Cantarell,&quot;Helvetica Neue&quot;,sans-serif">
   <table align="center" role="presentation" cellSpacing="0" cellPadding="0"
border="0" width="100%" style="max-width:37.5em;margin:0 auto;padding:20px 0
48px;width:560px">
     <tr style="width:100%">
       <td><img alt="Linear" src="https://react-email-demo-ijnnx5hul-
resend.vercel.app/static/linear-logo.png" width="42" height="42"
style="display:block;outline:none;border:none;text-decoration:none;border-
radius:21px;width:42px;height:42px" />
         <h1 style="font-size:24px;letter-spacing:-0.5px;line-height:1.3;font-
weight:400;color:#484848;padding:17px 0 0"><h1>Вітаємо
${decodeURIComponent(escape(user.name))}!</h1><br/>Ваш запис на
${decodeURIComponent(escape(service.name))} в RazorBeard</h1>
         <table style="padding:27px 0 27px" align="center" border="0" cellPadding="0"
cellSpacing="0" role="presentation" width="100%">
           <tbody>
             <tr>
               <td><a
href="https://calendar.google.com/calendar/u/0/r/eventedit?text=${decodeURIComponent(
escape(service.name))}+в+RazorBeard&details=${decodeURIComponent(escape(service.name)
)}+в+RazorBeard&${generateTimeRange(datetime)}&ctz=Ukraine/Lviv" target="_blank"
style="background-color:#d25e5e;border-radius:3px;font-weight:600;color:#fff;font-
size:15px;text-decoration:none;text-align:center;display:inline-block;p-x:23px;p-
y:11px;line-height:100%;max-width:100%;padding:11px 23px"><span><!--[if mso]><i
style="letter-spacing: 23px;mso-font-width:-100%;mso-text-raise:16.5"
hidden> </i><![endif]--></span><span style="background-color:#d25e5e;border-
radius:3px;font-weight:600;color:#fff;font-size:15px;text-decoration:none;text-
align:center;display:inline-block;p-x:23px;p-y:11px;max-width:100%;line-
height:120%;text-transform:none;mso-padding-alt:0px;mso-text-raise:8.25px">Додати в
календар</span><span><!--[if mso]><i style="letter-spacing: 23px;mso-font-width:-
100%" hidden> </i><![endif]--></span></a></td>
             </tr>
           </tbody>
         </table>
```

```
        <p style="font-size:15px;line-height:1.4;margin:0 0
15px;color:#3c4149">Чекаємо Вас:</p><p style="font-family:monospace;font-
weight:700;padding:1px 4px;background-color:#dfe1e4;letter-spacing:-0.3px;font-
size:21px;border-radius:4px;color:#3c4149;text-align:
center">${datetime.split(':00.')[0]}</p>
        <a
href="https://barbershop.razorbeard.site/api/cancel?record=${record.id}">Скасувати
запис</a>
        <hr style="width:100%;border:none;border-top:1px solid #eaeaea;border-
color:#dfe1e4;margin:42px 0 26px" /><a target="_blank" style="color:#b4becc;text-
decoration:none;font-size:14px" href="https://linear.app">RazorBeard</a>
       </td>
     </tr>
   </table>
 </body>

</html>
  `
 });

 if (error) {
  return new Response(JSON.stringify(error));
 }

 return new Response(JSON.stringify({ data, record }));
}

import pb from '$lib/pocketbase.js';

/** @type {import('./$types').LayoutServerLoad} */
export const load = async ({ cookies }) => {
 const service = JSON.parse(cookies.get('service') || '{}');
 const datetime = cookies.get('datetime');
 const barbers = await pb.collection('Barber').getFullList({
  filter: service.id ? `service ?~ "${service.id}" && level =
"${decodeURIComponent(escape(service.level))}"` : '',
 });
 return { barbers };
};

<script>
 import { Center, Paper } from '@svelteuidev/core';
 import { goto } from '$app/navigation';
```

```
 /** @type {import('./$types').LayoutServerLoad} */
 export let data;
</script>
<svelte:head>
 <title>About</title>
 <meta name="description" content="About this app" />
</svelte:head>

<section>
 <Center style="width: 100%">
  <Paper shadow="md" style="padding:3rem; display: flex; flex-direction: column; gap:
1rem; align-items: center;"
      radius="md" withBorder>
   {#each data.barbers as barber}
    <button on:click={_=>{
     sessionStorage.setItem('barber', barber.name);
     document.cookie = `barber=${JSON.stringify(barber)}`;
     goto('/');
    }}>
     <Paper
      style="display: flex; flex-direction: row; justify-content: start; gap: 1rem;
width: 100%;align-items: center">
      <img style="height: 100px; width: 100px; border-radius: 1rem"
         src={`/api/image-
proxy/?collection=${barber.collectionName}&id=${barber.id}&file=${barber.avatar}`}
         alt={barber.name} />
      <div>
       <h2>{barber.name}</h2>

       <h3>{barber.level}</h3>
      </div>
     </Paper>
    </button>
   {/each}
   <a href="/" style="font-size: 12px">>Назад</a>
  </Paper>
 </Center>

</section>

<style>
   section {
       display: flex;
       height: 100vh;
```

```
        width: auto;
        margin: 0;
    }

    button {
        width: 100%;
        background-color: transparent;
        border: none;
    }
</style>

import pb from '$lib/pocketbase.js';

/** @type {import('./$types').LayoutServerLoad} */
export const load = async ({ cookies }) => {
 const barber = JSON.parse(cookies.get('barber') || '{}');
 const service = JSON.parse(cookies.get('service') || '{}');
 const barbers = service.id ? await pb.collection('Barber').getFullList({
  filter: service.id ? `service ?~ "${service.id}" && level =
"${decodeURIComponent(escape(service.level))}"` + (barber.id ? `&& id =
"${barber.id}"` : '') : '',
 }) : await pb.collection('Barber').getFullList({
  filter: barber.id ? `id = "${barber.id}"` : ''
 });
 const records = await pb.collection('Record').getFullList({
  filter: barber.id ? `barber = "${barber.id}"` : '', expand: 'service,barber'
 });
 console.log(barbers, records);
 return { barbers, records };
};

<script>
 import { Center, Paper } from '@svelteuidev/core';
 import { Tabs } from '@svelteuidev/core';
 import { goto } from '$app/navigation';
 import { Accordion } from '@svelteuidev/core';

 /** @type {import('./$types').LayoutServerLoad} */
 export let data;

 function formatDate(inputDate) {
  const date = new Date(inputDate);

  const year = date.getFullYear();
```

```
 const month = String(date.getMonth() + 1).padStart(2, '0');
 const day = String(date.getDate()).padStart(2, '0');
 const hour = String(date.getHours()).padStart(2, '0');
 const minute = String(date.getMinutes()).padStart(2, '0');
 const second = String(date.getSeconds()).padStart(2, '0');
 const millisecond = String(date.getMilliseconds()).padStart(3, '0');

 return `${year}-${month}-${day} ${hour}:${minute}:${second}.${millisecond}Z`;
}

function areDatesEqual(date1, date2) {
 const objectDate1 = new Date(date1);
 const objectDate2 = new Date(date2);

 if (isNaN(objectDate1.getTime()) || isNaN(objectDate2.getTime())) {
  return false;
 }

 return objectDate1.getDate() === objectDate2.getDate() &&
  objectDate1.getMonth() === objectDate2.getMonth() &&
  objectDate1.getFullYear() === objectDate2.getFullYear();
}

function filterRecords(barber, date) {
 let times = [];
 let records = data.records.filter(i => i.barber === barber.id);
 records = records.filter(i => areDatesEqual(i.time, date));
 console.log(records);
 for (let i = Number(barber.time.split(':')[0]); i < Number(barber.time.split('-
')[1].split(':')[0]); i += 1) {
  let f = true;
  for (let r of records) {
   if (`${i<10 ? '0'+i : i}:00` === r.time.split(' ')[1].split(':00.')[0]) {
    f = false
    break;
   }
  }
  if (f) times.push(`${i}:00`);
 }
 return times;
}

function week() {
 const today = new Date();
```

```
  let day = today.getDay();
  let diff = (2 - day + 7) % 7;
  let list = [];

  for (let i = 0; i <= diff; i++) {
   let current = new Date(today);
   current.setDate(today.getDate() + i);
   list.push({
    day: current.toLocaleDateString('en-US', { weekday: 'long' }),
    date: current.toLocaleDateString('en-US')
   });
  }

  return list;
 }

</script>
<svelte:head>
 <title>About</title>
 <meta name="description" content="About this app" />
</svelte:head>

<section>
 <Center style="width: 100%">
  <Paper shadow="md" style="padding:3rem; display: flex; flex-direction: column; gap:
1rem; align-items: center;"
      radius="md" withBorder>
   <Accordion style="width: 800px">
    {#each data.barbers as barber}
     <Accordion.Item value={barber.name}>
      <div slot="control">{barber.name}</div>
      <Tabs>
       {#each week() as day}
        <Tabs.Tab style="display: flex; flex-direction: row"
             label={`${day.day} - ${day.date.split('/')[1].split('/')[0]}`}>
         {#each filterRecords(barber, day.date) as time}
          <button on:click={_=>{
           const datetime = formatDate(`${day.date} ${time}`);
           sessionStorage.setItem('datetime', datetime);
           document.cookie = `datetime=${datetime}`;
           goto('/');
          }}>
           {time}
          </button>
```

```
          {/each}
         </Tabs.Tab>
        {/each}
       </Tabs>
      </Accordion.Item>
     {/each}
    </Accordion>
    <a href="/" style="font-size: 12px">>Назад</a>
   </Paper>
 </Center>


</section>

<style>
    section {
        display: flex;
        height: 100vh;
        width: auto;
        margin: 0;
    }

    button {
        background-color: transparent;
        border: none;
        transition: all 200ms;
        border-radius: 1rem;
        padding: 7px;
    }

    button:hover {
        background-color: lightgrey;
    }
</style>

import pb from '$lib/pocketbase.js';

/** @type {import('./$types').LayoutServerLoad} */
export const load = async ({ cookies }) => {
 const barber = JSON.parse(cookies.get('barber') || '{}');
 const services = await pb.collection('Barber').getFullList({
  filter: barber.id ? `id = "${barber.id}"` : '', expand: 'service'
 });
 return { services };
```

```
};

<script>
 import { Center, Paper } from '@svelteuidev/core';
 import { goto } from '$app/navigation';
 import { Accordion } from '@svelteuidev/core';

 /** @type {import('./$types').LayoutServerLoad} */
 export let data;
</script>
<svelte:head>
 <title>About</title>
 <meta name="description" content="About this app" />
</svelte:head>

<section>
 <Center style="width: 100%">
  <Paper shadow="md" style="padding:3rem; display: flex; flex-direction: column; gap:
1rem; align-items: center;"
      radius="md" withBorder>
   <Accordion style="width: 400px">
    {#each data.services as barber}
     <Accordion.Item value={barber.level}>
      <div slot="control">{barber.level}</div>
      {#each barber.expand.service as service}
       <button on:click={_=>{
     sessionStorage.setItem('service', service.name);
     document.cookie = `service=${JSON.stringify({level: barber.level,
...service})}`;
     goto('/');
    }}>
        {service.name}={barber.level}={service.price * barber.price}
       </button>
       <br />
      {/each}
     </Accordion.Item>
    {/each}
   </Accordion>
   <a href="/" style="font-size: 12px">>Назад</a>
  </Paper>
 </Center>

</section>
```

```
<style>
   section {
       display: flex;
       height: 100vh;
       width: auto;
       margin: 0;
   }

   button {
       background-color: transparent;
       border: none;
       transition: all 200ms;
       border-radius: 1rem;
       padding: 7px;
   }

   button:hover {
       background-color: lightgrey;
   }
</style>

<script>
 import preview from '$lib/preview.jpg';
 import { Alert, Button, Center, Paper } from '@svelteuidev/core';
 import { goto } from '$app/navigation';
 import { browser } from '$app/environment';
 import { Input } from '@svelteuidev/core';

 let alertMessage = false;

 let barber = '';
 let service = '';
 let datetime = '';
 let email = '';
 let name = '';
 if (browser) {
  barber = sessionStorage.getItem('barber');
  service = sessionStorage.getItem('service');
  datetime = sessionStorage.getItem('datetime');
 }

 function isValidEmail(email) {
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return emailRegex.test(email);
```

```
 }

 function deleteAllCookies() {
  const allCookies = document.cookie.split(';');

  for (let i = 0; i < allCookies.length; i++) {
   const cookieName = allCookies[i].split('=')[0];
   document.cookie = cookieName + '=; expires=Thu, 01 Jan 1970 00:00:00 GMT; path=/';
  }
 }
</script>

<svelte:head>
 <title>RazorBeard</title>
 <meta name="description" content="RazorBeard Barbershop" />
</svelte:head>

<section>
 <img src={preview} alt="Welcome" class="welcome" />

 <Center style="width: 100%">
  <Paper shadow="md"
      style="padding:3rem; display: flex; flex-direction: column; gap: 1rem; width:
400px; align-items: center"
      radius="md" withBorder>
   <h1>RazorBeard</h1>
   <div style="display: flex; flex-direction: row; gap: 1rem; width: 100%">
    <Button style="width: 100%" on:click={_=>{goto('/service')}} variant="outline"
color="red" radius="md" size="md"
        ripple>
     Вибрати послугу
    </Button>
    {#if !!service}
     <button on:click={_=>{
    document.cookie = `service=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;`;
    sessionStorage.removeItem('service');
    service = null;
   }}
         style="width: 100%; display: flex; cursor: pointer; flex-direction: row;
justify-content: space-evenly; align-items: center; gap: 1rem">
      {service}
      <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24" stroke-
width="1.5"
         stroke="currentColor" style="min-width: 20px;width: 20px;height: 20px">
```

```
       <path stroke-linecap="round" stroke-linejoin="round"
          d="M12 9.75 14.25 12m0 0 2.25 2.25M14.25 12l2.25-2.25M14.25 12 12 14.25m-
2.58 4.92-6.374-6.375a1.125 1.125 0 0 1 0-1.59L9.42 4.83c.21-.211.497-.33.795-
.33H19.5a2.25 2.25 0 0 1 2.25 2.25v10.5a2.25 2.25 0 0 1-2.25 2.25h-9.284c-.298 0-
.585-.119-.795-.33Z" />
      </svg>
     </button>
    {/if}
   </div>
   <div style="display: flex; flex-direction: row; gap: 1rem; width: 100%">
    <Button style="width: 100%" on:click={_=>{goto('/barber')}} variant="outline"
color="red" radius="md" size="md"
        ripple>
     Вибрати майстра
    </Button>
    {#if !!barber}
     <button on:click={_=>{
    document.cookie = `barber=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;`;
    sessionStorage.removeItem('barber');
    barber = null;
   }}
         style="width: 100%; display: flex; cursor: pointer; flex-direction: row;
justify-content: space-evenly; align-items: center; gap: 1rem">
      {barber}
      <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24" stroke-
width="1.5"
         stroke="currentColor" style="min-width: 20px;width: 20px;height: 20px">
       <path stroke-linecap="round" stroke-linejoin="round"
          d="M12 9.75 14.25 12m0 0 2.25 2.25M14.25 12l2.25-2.25M14.25 12 12 14.25m-
2.58 4.92-6.374-6.375a1.125 1.125 0 0 1 0-1.59L9.42 4.83c.21-.211.497-.33.795-
.33H19.5a2.25 2.25 0 0 1 2.25 2.25v10.5a2.25 2.25 0 0 1-2.25 2.25h-9.284c-.298 0-
.585-.119-.795-.33Z" />
      </svg>
     </button>
    {/if}
   </div>
   {#if !(!barber || !service)}
    <div style="display: flex; flex-direction: row; gap: 1rem; width: 100%">
     <Button style="width: 100%" on:click={_=>{goto('/datetime')}} variant="outline"
color="red" radius="md"
         size="md"
         ripple>
      Вибрати час
     </Button>
```

```
     {#if !!datetime}
      <button on:click={_=>{
    document.cookie = `datetime=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;`;
    sessionStorage.removeItem('datetime');
    datetime = null;
   }}
          style="width: 100%; display: flex; cursor: pointer; flex-direction: row;
justify-content: space-evenly; align-items: center; gap: 1rem">
       {datetime.split(':00.')[0].replace(' ', '\n')}
       <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24"
stroke-width="1.5"
          stroke="currentColor" style="min-width: 20px;width: 20px;height: 20px">
        <path stroke-linecap="round" stroke-linejoin="round"
           d="M12 9.75 14.25 12m0 0 2.25 2.25M14.25 12l2.25-2.25M14.25 12 12 14.25m-
2.58 4.92-6.374-6.375a1.125 1.125 0 0 1 0-1.59L9.42 4.83c.21-.211.497-.33.795-
.33H19.5a2.25 2.25 0 0 1 2.25 2.25v10.5a2.25 2.25 0 0 1-2.25 2.25h-9.284c-.298 0-
.585-.119-.795-.33Z" />
       </svg>
      </button>
     {/if}
    </div>
   {/if}
   {#if !(!barber || !service || !datetime)}
    {#if !isValidEmail(email)}
     Введіть правильно пошту
    {/if}
    <Input style="width:100%"
        placeholder="Ваша електронна пошта"
        radius="md"
        size="md"
        bind:value={email}
    />
    {#if name.length < 3}
     Введіть правильно ім'я
    {/if}
    <Input style="width:100%"
        placeholder="Ваше ім'я"
        radius="md"
        size="md"
        bind:value={name}
        required
    />
   {/if}
   <Button style="width: 100%" on:click={async _=>{
```

```
    document.cookie = `user=${JSON.stringify({email,name})}`;
    const res = await(await fetch('/api/record')).json();
    if(res.data && res.record){
     deleteAllCookies();
     sessionStorage.clear();
     barber = '';
     service = '';
     datetime = '';
     email = '';
     name = '';
    }
    alertMessage = true;
    await new Promise(r => setTimeout(r, 5000));
    alertMessage = false;
   }} variant="filled" color="red" radius="md" size="md" disabled={!barber ||
!service || !datetime}
       ripple>
    Записатися
   </Button>
   <a href="/about" style="font-size: 12px">>Про нас</a>
  </Paper>
 </Center>

</section>
{#if alertMessage}
<Alert style="position: absolute;top:1rem;left:50%;transform: translateX(-50%)"
title="Це успіх!" color="red">
Запис створено! Перегляньте електронну пошту для деталей.
</Alert>
 {/if}
<style>
   section {
       display: flex;
       height: 100vh;
       width: auto;
       margin: 0;
   }

   img {
       width: auto;
       height: 100vh;
   }

   h1 {
```

```
        font-size: 2rem;
    }

    button {
        width: 100%;
        background-color: transparent;
        border: none;
    }
</style>

<!doctype html>
<html lang="en">
 <head>
  <meta charset="utf-8" />
  <link rel="icon" href="%sveltekit.assets%/favicon.png" />
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link
href="https://fonts.googleapis.com/css2?family=Roboto+Slab:wght@500&display=swap"
rel="stylesheet">
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  %sveltekit.head%
 </head>
 <body data-sveltekit-preload-data="hover">
  <div>%sveltekit.body%</div>
 </body>
<style>
 html, body {
    margin: 0;
      font-family: 'Roboto Slab', serif;
 }
</style>
</html>

Dockerfile
.dockerignore
.git
.gitignore
.gitattributes
README.md
.npmrc
.prettierrc
.eslintrc.cjs
.graphqlrc
```

```
.editorconfig
.svelte-kit
.vscode
node_modules
build
package
**/.env
.idea

.DS_Store
node_modules
/build
/.svelte-kit
/package
.env
.env.*
!.env.example
.vercel
.output
vite.config.js.timestamp-*
vite.config.ts.timestamp-*
.svelte-kit

{
   "useTabs": true,
   "singleQuote": true,
   "trailingComma": "none",
   "printWidth": 100,
   "plugins": ["prettier-plugin-svelte"],
   "overrides": [{ "files": "*.svelte", "options": { "parser": "svelte" } }]
}

{
   "name": "razorbeard",
   "version": "0.0.1",
   "scripts": {
      "dev": "vite dev",
      "build": "vite build",
      "preview": "vite preview",
      "lint": "prettier --check .",
      "format": "prettier --write ."
   },
   "devDependencies": {
      "@fontsource/fira-mono": "^4.5.10",
```

```
      "@neoconfetti/svelte": "^1.0.0",
      "@sveltejs/adapter-auto": "^3.0.0",
      "@sveltejs/kit": "^2.0.0",
      "@sveltejs/vite-plugin-svelte": "^3.0.0",
      "prettier": "^3.1.1",
      "prettier-plugin-svelte": "^3.1.2",
      "svelte": "^4.2.7",
      "vite": "^5.0.3"
   },
   "type": "module",
   "dependencies": {
      "@sveltejs/adapter-node": "^3.0.1",
      "@svelteuidev/composables": "^0.15.4",
      "@svelteuidev/core": "^0.15.4",
      "@svelteuidev/dates": "^0.15.4",
      "@svelteuidev/motion": "^0.15.4",
      "@svelteuidev/preprocessors": "^0.15.4",
      "@svelteuidev/prism": "^0.15.4",
      "dayjs": "^1.11.10",
      "pocketbase": "^0.20.3",
      "resend": "^2.1.0"
   }
}

import adapter from '@sveltejs/adapter-node';

export default {
 kit: {
  adapter: adapter({
   // default options are shown
   out: 'build',
   precompress: false,
   envPrefix: ''
  })
 }
};
```