Geekbrains

Система онлайн-бронирования для малого бизнеса на примере салона красоты

Программа: «Программист Python»

Юнусов Элим Вахаевич

Астрахань

2024

Содержание

Введение

Тема проекта

Проект представляет собой систему онлайн бронирования для малого бизнеса на примере салона красоты. Система позволяет клиентам записываться на услуги через интернет, а администраторам и менеджерам управлять расписанием и бронированиями. Основные функции включают регистрацию и авторизацию пользователей, создание и управление услугами, бронирование времени у мастеров, а также оставление отзывов.

Обоснование темы проекта

В современном мире цифровизация и автоматизация бизнес-процессов становятся все более актуальными. Малый бизнес, в частности салоны красоты, нуждаются в эффективных инструментах для управления клиентами и расписанием. Онлайн система бронирования позволяет сократить время на организацию встреч, улучшить клиентский сервис и повысить общую эффективность работы салона.

Цель проекта

Целью проекта является разработка и внедрение системы онлайн бронирования для салона красоты, которая позволит автоматизировать процесс записи клиентов, улучшить управление расписанием и повысить удовлетворенность клиентов.

Какую проблему решает проект

Проект решает проблему неэффективного управления расписанием и бронированиями в салонах красоты. Система позволяет автоматизировать процесс записи клиентов, что снижает вероятность ошибок, улучшает клиентский сервис и освобождает время для сотрудников салона.

Инструменты

Django: фреймворк для разработки веб-приложений на Python.

SQLite: база данных для хранения данных.

HTML/CSS: для создания пользовательских интерфейсов.

JavaScript: для добавления интерактивности на веб-страницах.

Git: система контроля версий для управления исходным кодом.

Технологии

Django ORM: для работы с базой данных.

Django Forms: для создания и обработки форм.

Django Templates: для рендеринга HTML-страниц.

AJAX: для асинхронного взаимодействия с сервером.

Состав команды

Проект выполнялся автором данной работы самостоятельно.

Теоретическая часть

Обзор существующих систем бронирования

В современном мире существует множество систем онлайн бронирования, которые помогают малому бизнесу, в том числе салонам красоты, эффективно управлять своими услугами и клиентами. Рассмотрим несколько популярных решений и проанализируем их преимущества и недостатки.

1. Fresha (бывший Shedul)

Преимущества:

- Бесплатное использование: Fresha предлагает бесплатный план, который включает в себя большинство необходимых функций для управления бронированиями.
- Интуитивно понятный интерфейс: платформа имеет простой и удобный интерфейс, что облегчает работу как для администраторов, так и для клиентов.
- Интеграция с социальными сетями: возможность интеграции с Facebook и Instagram для привлечения новых клиентов.

Недостатки:

- Ограниченные возможности кастомизации: бесплатный план не позволяет вносить значительные изменения в интерфейс и функционал системы.

2. Booksy

Преимущества:

- Мобильное приложение: Booksy предлагает удобное мобильное приложение для клиентов, что упрощает процесс бронирования.

- Маркетинговые инструменты: платформа предоставляет инструменты для маркетинга и продвижения услуг, такие как рассылка сообщений и уведомлений.

Недостатки:

- Стоимость: Booksy предлагает платные планы, которые могут быть дорогими для малого бизнеса.
- Сложность настройки: некоторые пользователи отмечают, что настройка системы может быть сложной и требует времени.

3. Mindbody

Преимущества:

- Широкий функционал: Mindbody предлагает множество функций, включая управление расписанием, оплату, маркетинг и аналитику.
- Интеграция с фитнес-устройствами: платформа интегрируется с различными фитнес-устройствами и приложениями, что делает её популярной среди фитнес-центров и студий йоги.

Недостатки:

- Высокая стоимость: Mindbody является одним из самых дорогих решений на рынке, что делает его недоступным для многих малых бизнесов.
- Сложность использования: из-за большого количества функций система может быть сложной в использовании для новых пользователей.

4. SimplyBook.me

Преимущества:

- Гибкость настройки: SimplyBook.me позволяет настраивать систему под конкретные нужды бизнеса, включая создание собственных страниц бронирования.
- Многоязычная поддержка: платформа поддерживает множество языков, что делает её удобной для международных пользователей.

Недостатки:

- Ограниченный бесплатный план: Бесплатный план имеет ограничения по количеству бронирований и доступным функциям.
- Интерфейс: некоторые пользователи отмечают, что интерфейс системы может быть устаревшим и неинтуитивным.

Анализируя существующие системы бронирования, можно выделить несколько общих недостатков, которые могут быть решены в рамках разработки собственной системы:

- Стоимость: многие популярные системы бронирования предлагают платные планы, которые могут быть дорогими для малого бизнеса. Разработка собственной системы позволяет избежать ежемесячных платежей и контролировать расходы на поддержку и развитие.
- Ограниченные возможности кастомизации: большинство готовых решений не позволяют вносить значительные изменения в интерфейс и функционал системы. Собственная разработка предоставляет полную свободу в настройке и адаптации системы под конкретные нужды бизнеса.
- Сложность использования: некоторые системы имеют сложный интерфейс и требуют времени на обучение. Разработка собственной системы позволяет создать интуитивно понятный интерфейс, ориентированный на конкретных пользователей.
- Ограничения бесплатных планов: бесплатные планы часто имеют ограничения по количеству бронирований и доступным функциям. Собственная система не будет иметь таких ограничений и сможет масштабироваться по мере роста бизнеса.

Таким образом, разработка собственной системы онлайн бронирования для салона красоты позволяет учесть все особенности и потребности бизнеса, обеспечивая гибкость, экономичность и удобство использования.

Требования к системе онлайн-бронирования

Требования для малого бизнеса могут быть разделены на функциональные и нефункциональные требования. Функциональные требования описывают конкретные функции и возможности, которые должна предоставлять система, в то время как нефункциональные требования касаются аспектов производительности, безопасности, надежности и других качественных характеристик системы.

Функциональные требования

1. Управление бронированиями: система должна предоставлять возможность создавать, изменять, просматривать и отменять бронирования.
2. Управление ресурсами: система должна позволять владельцам бизнеса управлять доступностью ресурсов.
3. Управление клиентами: система должна предоставлять инструменты для управления информацией о клиентах, включая историю бронирований, предпочтения и отзывы.
4. Аутентификация и авторизация: система должна обеспечивать безопасный вход в систему для клиентов и администраторов, а также различные уровни доступа в зависимости от роли пользователя.
5. Интеграция с другими системами: возможность интеграции с другими системами и сервисами, например, с системами управления отношениями с клиентами (CRM), платежными системами и т.д.
6. API для внешних интеграций: предоставление REST API для взаимодействия с другими системами и разработкой мобильных приложений.

Нефункциональные требования

1. Производительность: система должна обеспечивать быструю обработку запросов и высокую скорость ответа, особенно при высокой нагрузке.

2. Надежность: система должна быть устойчивой к сбоям и обеспечивать высокий уровень доступности.
3. Безопасность: система должна обеспечивать защиту данных пользователей и соблюдение требований к безопасности, включая шифрование данных и защиту от атак.
4. Масштабируемость: система должна быть способна легко масштабироваться для поддержки увеличения объема бронирований и количества пользователей.
5. Удобство использования: интерфейс системы должен быть интуитивно понятным и удобным для пользователей различных возрастных групп и уровня компетенции.
6. Поддержка мобильных устройств: система должна обеспечивать корректную работу на различных мобильных устройствах и платформах.

Эти требования служат основой для разработки системы онлайн-бронирования, позволяя создать решение, которое будет отвечать потребностям как малого бизнеса, так и его клиентов.

Технологический стек

Выбор Python и Django как основных технологий для разработки системы онлайн-бронирования для малого бизнеса обусловлен рядом преимуществ, которые эти технологии предлагают.

Python является одним из самых популярных языков программирования, который известен своей простотой и читаемостью кода. Это делает его идеальным выбором для разработки сложных систем, таких как система онлайн-бронирования, где требуется быстрое и эффективное решение.

- Простота и читаемость: Python позволяет разработчикам писать код, который легко читать и поддерживать, что ускоряет процесс разработки и упрощает его для новых членов команды.
- Богатая экосистема: Python имеет огромное количество библиотек и фреймворков, которые могут быть использованы для решения различных задач, включая разработку веб-приложений, работы с базами данных, обработку данных и многое другое.
- Кросс-платформенность: Python поддерживает разработку кросс-платформенных приложений, что позволяет легко адаптировать систему онлайн-бронирования под различные операционные системы.

Django — это высокоуровневый Python веб-фреймворк, который позволяет быстро разрабатывать безопасные и масштабируемые веб-приложения. Django предлагает множество встроенных функций, которые упрощают разработку, включая ORM (Object-Relational Mapping) для работы с базами данных, систему аутентификации и авторизации, административный интерфейс и многое другое.

- Быстрая разработка: благодаря "batteries-included" подходу Django, разработчики могут сосредоточиться на логике приложения, не тратя время на разработку общих функций.

- Масштабируемость: Django разработан с учетом масштабируемости, что позволяет легко адаптировать систему онлайн-бронирования под растущие потребности бизнеса.
- Безопасность: Django предлагает множество встроенных механизмов безопасности, которые помогают предотвратить распространенные веб-уязвимости.

Выбор Python и Django как основных технологий для разработки системы онлайн-бронирования обусловлен их простотой, мощностью и гибкостью. Эти технологии позволят создать удобное, эффективное и масштабируемое решение, которое будет отвечать потребностям малого бизнеса.

Разработка системы

Архитектура системы

Модели данных

Архитектура системы онлайн-бронирования для малого бизнеса должна быть гибкой и масштабируемой, чтобы соответствовать потребностям как малых, так и средних предприятий. В основе системы лежат модели данных, которые отражают основные сущности и их взаимоотношения.

Пользователи

Модель пользователей включает следующие поля:

- id: уникальный идентификатор пользователя.
- username: имя пользователя.
- password: хэшированный пароль пользователя.
- first_name: имя пользователя.
- last_name: фамилия пользователя.
- phone: номер телефона пользователя.
- email: электронная почта пользователя.
- birth_date: дата рождения пользователя.

Администраторы

Модель администраторов включает следующие поля:

- id: уникальный идентификатор администратора.
- user: ссылка на пользователя.
- phone: номер телефона администратора.
- birth_date: дата рождения администратора.

Менеджеры

Модель менеджеров включает следующие поля:

- id: уникальный идентификатор менеджера.
- user: ссылка на пользователя.
- phone: номер телефона менеджера.
- birth_date: дата рождения менеджера.

Поставщики услуг

Модель поставщиков услуг включает следующие поля:

- id: уникальный идентификатор поставщика услуг.
- user: ссылка на пользователя.
- specialization: специализация поставщика услуг.
- phone: номер телефона поставщика услуг.
- birth_date: дата рождения поставщика услуг.

Услуги

Модель услуг включает следующие поля:

- id: уникальный идентификатор услуги.
- name: название услуги.
- description: описание услуги.
- price: цена услуги.
- duration: продолжительность услуги.

Бронирования

Модель бронирований включает следующие поля:

- id: уникальный идентификатор бронирования.
- customer: ссылка на пользователя, который совершил бронирование.
- service: ссылка на услугу.
- service_provider: ссылка на поставщика услуги.
- appointment_datetime: дата и время бронирования.
- created_at: дата и время создания бронирования.

Отзывы

Модель отзывов включает следующие поля:

- id: уникальный идентификатор отзыва.

- booking: ссылка на бронирование.
- rating: оценка услуги.
- comment: комментарий к отзыву.
- created_at: дата и время создания отзыва.

Взаимоотношения между моделями

- Пользователи могут бронировать услуги, создавая связи между моделями "Пользователи" и "Бронирования".
- Услуги предоставляются поставщиками услуг, которые также являются пользователями. Это отражается через связь между моделями "Пользователи" и "Услуги".
- Отзывы оставляются пользователями по услугам, что отражается через связь между моделями "Пользователи" и "Отзывы".

Архитектура программы

Архитектура программы построена на основе MVT (Model-View-Template) паттерна, который является основой для разработки веб-приложений на Django. В этом случае:

- Модели (Models): определяют структуру данных и взаимоотношения между ними.
- Представления (Views): обрабатывают запросы от клиентов, взаимодействуют с моделями для получения данных и передают их в шаблоны для отображения.
- Шаблоны (Templates): определяют, как данные представляются пользователю.

Такая архитектура позволяет создать гибкую и масштабируемую систему, которая может легко адаптироваться к изменениям в бизнесе и потребностям пользователей.

Виды пользователей

В системе онлайн-бронирования для малого бизнеса различаются несколько типов пользователей, каждый из которых имеет свои уникальные потребности и функции в рамках системы.

Администраторы

Администраторы системы отвечают за управление всеми аспектами системы. Они имеют полный доступ ко всем функциям и данным, включая:

- Управление информацией о услугах и ресурсах.
- Управление профилями клиентов и поставщиков услуг.
- Управление настройками системы.
- Поддержка пользователей.

Менеджеры

Менеджеры системы имеют доступ к управлению бронированиями и взаимодействию с клиентами и поставщиками услуг. Их функции включают:

- Управление бронированиями, включая подтверждение, изменение и отмену бронирований.
- Взаимодействие с клиентами для решения вопросов и предоставления информации.
- Управление расписанием поставщиков услуг.
- Просмотр отчетов и статистики по бронированиям и услугам.
- Поддержка клиентов и поставщиков услуг в случае возникновения проблем.

Поставщики услуг

Поставщики услуг (мастера) являются ключевыми пользователями системы, предоставляющими услуги клиентам. Их функции включают:

- Управление своим профилем и расписанием.

- Просмотр и управление своими бронированиями.
- Взаимодействие с клиентами для уточнения деталей услуг.
- Оценка и управление отзывами клиентов.
- Обновление информации о предоставляемых услугах и их стоимости.

Клиенты

Клиенты системы являются конечными пользователями, которые бронируют услуги. Их функции включают:

- Регистрация и управление своим профилем.
- Просмотр доступных услуг и поставщиков услуг.
- Создание, изменение и отмена бронирований.
- Оставление отзывов и оценок о полученных услугах.
- Просмотр истории своих бронирований и взаимодействий с системой.

Функциональные возможности системы

Регистрация и аутентификация

Система предоставляет функционал для регистрации новых пользователей и аутентификации существующих. Это включает:

- Регистрацию новых пользователей с указанием личных данных и контактной информации.
- Вход в систему с использованием имени пользователя и пароля.

Управление профилями

Каждый тип пользователя имеет возможность управлять своим профилем, включая:

- Обновление личной информации и контактных данных.
- Изменение пароля.
- Деактивация профиля.

Бронирование услуг

Система предоставляет клиентам возможность бронирования услуг, включая:

- Просмотр доступных услуг и поставщиков услуг.
- Выбор даты и времени для бронирования.

Управление бронированиями

Администраторы и менеджеры могут управлять бронированиями, включая:

- Подтверждение, изменение и отмену бронирований.
- Просмотр и анализ статистики бронирований.
- Управление расписанием поставщиков услуг.

Оставление отзывов

Клиенты могут оставлять отзывы о полученных услугах, включая:

- Оценку качества услуги.
- Написание комментариев и предложений.
- Просмотр отзывов других клиентов.

Система онлайн-бронирования для малого бизнеса предоставляет широкий спектр функциональных возможностей для различных типов пользователей. Она обеспечивает удобство и эффективность управления бронированиями, улучшает взаимодействие между клиентами и поставщиками услуг, а также предоставляет инструменты для анализа и улучшения качества предоставляемых услуг.

Бизнес-логика

Система онлайн бронирования для салона красоты предназначена для автоматизации и упрощения процесса записи клиентов на услуги. Основные бизнес-процессы и правила бронирования включают в себя следующие аспекты:

1. Регистрация и авторизация пользователей

- Регистрация: пользователи могут зарегистрироваться в системе, заполнив форму регистрации, где указывают свои личные данные, такие как имя, фамилия, адрес электронной почты, номер телефона и дату рождения. После успешной регистрации создается профиль клиента.
- Авторизация: зарегистрированные пользователи могут войти в систему, используя свои учетные данные (имя пользователя и пароль). Администраторы и менеджеры также проходят авторизацию, но имеют расширенные права доступа.

2. Управление пользователями

- Администраторы: имеют доступ к управлению всеми аспектами системы, включая создание и редактирование профилей пользователей, управление услугами и расписанием мастеров.
- Менеджеры: могут управлять бронированиями, просматривать и редактировать записи клиентов, а также управлять расписанием мастеров.
- Мастера: имеют доступ к своему расписанию и могут просматривать записи клиентов на свои услуги.
- Клиенты: могут просматривать доступные услуги, бронировать их, а также оставлять отзывы о предоставленных услугах.

3. Управление услугами

- Создание и редактирование услуг: администраторы могут добавлять новые услуги, указывая их название, описание, продолжительность и стоимость. Также они могут редактировать существующие услуги.

- Назначение услуг мастерам: услуги могут быть назначены конкретным мастерам, что позволяет клиентам выбирать мастера при бронировании услуги.

4. Процесс бронирования

- Выбор услуги и мастера: клиенты выбирают услугу из списка доступных, а затем выбирают мастера, который предоставляет эту услугу.
- Выбор даты и времени: клиенты выбирают удобную дату и время для бронирования. Система проверяет доступность выбранного времени у выбранного мастера.
- Подтверждение бронирования: после выбора услуги, мастера и времени, клиент подтверждает бронирование. Система сохраняет запись в базе данных и уведомляет клиента о успешном бронировании.

5. Управление бронированиями

- Просмотр и редактирование бронирований: клиенты могут просматривать свои текущие и прошедшие бронирования, а также редактировать или отменять их при необходимости.
- Управление бронированиями администраторами и менеджерами: Администраторы и менеджеры могут просматривать все бронирования, редактировать их, а также удалять при необходимости.

6. Оставление отзывов

- Оставление отзывов клиентами: после завершения услуги клиенты могут оставить отзыв о предоставленной услуге, указав рейтинг и комментарий. Отзывы помогают улучшать качество обслуживания и предоставляют обратную связь мастерам и администрации салона.

7. Валидация и проверка доступности

- Проверка доступности времени: при бронировании система проверяет, доступно ли выбранное время у выбранного мастера, чтобы избежать конфликтов в расписании.
- Валидация данных: Все введенные данные проходят валидацию для обеспечения корректности и полноты информации.

Реализация на Django

Этапы разработки системы онлайн-бронирования для малого бизнеса на примере салона красоты

Этап 1. Инициализация проекта и настройка окружения

Инструменты:

- Django
- SQLite
- Git

Описание: на первом этапе был создан новый проект на Django. Это включало установку Django и создание структуры проекта с помощью команды django-admin startproject. База данных была настроена на использование SQLite, что упрощает начальную настройку и разработку. Git использовался для контроля версий, что позволило отслеживать изменения в коде и работать с ветками для различных функций.

Решаемые проблемы:

- Создание базовой структуры проекта.
- Настройка базы данных для хранения данных.
- Введение системы контроля версий для управления исходным кодом.

Этап 2. Разработка моделей данных

Инструменты:

- Django ORM

Описание: на этом этапе были разработаны модели данных, которые описывают основные сущности системы: пользователи, администраторы, менеджеры, поставщики услуг, услуги, бронирования и отзывы. Django ORM позволил легко определить модели и их взаимоотношения, а также автоматически сгенерировать таблицы в базе данных.

Решаемые проблемы:

- Определение структуры данных и их взаимоотношений.
- Автоматическое создание таблиц в базе данных.

Этап 3: Создание представлений и маршрутизация

Инструменты:

- Django Views
- Django URLconf

Описание: были созданы представления (views) для обработки запросов от клиентов и взаимодействия с моделями данных. Также была настроена маршрутизация (URLconf) для определения URL-адресов, которые соответствуют различным представлениям. Это позволило организовать логику приложения и обеспечить доступ к различным функциям системы.

Решаемые проблемы:

- Обработка запросов от клиентов.
- Организация логики приложения.
- Настройка маршрутизации для доступа к различным функциям.

Этап 4: Разработка шаблонов и пользовательского интерфейса

Инструменты:

- Django Templates
- HTML/CSS
- JavaScript

Описание: были созданы HTML-шаблоны для отображения данных пользователям. Django Templates использовались для рендеринга страниц с данными, полученными из представлений. HTML и CSS помогли создать привлекательный и удобный интерфейс, а JavaScript добавил интерактивности.

Решаемые проблемы:

- Создание удобного и привлекательного пользовательского интерфейса.
- Рендеринг данных на страницах.

- Добавление интерактивности для улучшения пользовательского опыта.

Этап 5: Реализация аутентификации и авторизации

Инструменты:

- Django Authentication System

Описание: были реализованы механизмы регистрации, входа и выхода пользователей с использованием встроенной системы аутентификации Django. Также были настроены уровни доступа для различных типов пользователей (администраторы, менеджеры, поставщики услуг, клиенты).

- Решаемые проблемы:
- Обеспечение безопасного входа и выхода пользователей.
- Управление доступом к различным функциям системы в зависимости от роли пользователя.

Этап 6: Тестирование

Инструменты:

- Ручное тестирование

Описание: были написаны команды для тестирования отдельных компонентов системы. Проводилось ручное тестирование для проверки корректности работы пользовательского интерфейса и функциональности приложения. Это позволило выявить и исправить ошибки до развертывания системы.

Решаемые проблемы:

- Проверка корректности работы системы.
- Обеспечение надежности и стабильности приложения.
- Выявление и исправление ошибок до развертывания.

Этап 7: Развертывание и поддержка

Инструменты:

- PythonAnywhere

Описание: проект был развернут на платформе PythonAnywhere, что упростило процесс развертывания и обеспечило стабильное окружение для работы приложения. PythonAnywhere предоставляет удобные инструменты для развертывания Django-приложений и управления ими.

Решаемые проблемы:

- Упрощение процесса развертывания.
- Обеспечение стабильного окружения.
- Быстрое и надежное развертывание системы.

Каждый из этих этапов был важен для создания функциональной и надежной системы онлайн-бронирования, которая отвечает потребностям малого бизнеса и его клиентов.

Тестирование

Тестирование является неотъемлемой частью процесса разработки программного обеспечения. Оно позволяет выявить и исправить ошибки, а также убедиться в том, что система работает корректно и соответствует требованиям. В данной главе описаны методология тестирования, проведенные тесты и их результаты.

Методология тестирования.

Для тестирования системы онлайн-бронирования были использованы следующие подходы:

1. Ручное тестирование:
    - Проверка функциональности пользовательского интерфейса.
    - Проверка корректности выполнения основных операций (регистрация, вход, бронирование, изменение и удаление бронирований).
    - Проверка работы системы аутентификации и авторизации.
2. Командное тестирование:
    - Написание и выполнение команд для тестирования отдельных компонентов системы.
    - Проверка взаимодействия между различными компонентами системы.

Проведенные тесты

1. Тестирование функциональности пользовательского интерфейса

Цель: убедиться, что пользовательский интерфейс работает корректно и предоставляет пользователю все необходимые функции.

Процедура:

- Проверка отображения главной страницы.
- Проверка формы регистрации и входа.
- Проверка страницы профиля пользователя.
- Проверка страницы бронирования услуг.

- Проверка страницы управления бронированиями для администратора.

Результаты:

- Все страницы отображаются корректно.
- Формы регистрации и входа работают без ошибок.
- Страницы профиля пользователя и бронирования услуг предоставляют все необходимые функции.
- Страница управления бронированиями для администратора позволяет корректно управлять бронированиями.

2. Тестирование основных операций

Цель: убедиться, что основные операции системы выполняются корректно.

Процедура:

- Регистрация нового пользователя.
- Вход в систему.
- Создание нового бронирования.
- Изменение существующего бронирования.
- Удаление бронирования.

Результаты:

- Регистрация и вход в систему работают корректно.
- Создание, изменение и удаление бронирований выполняются без ошибок.

3. Тестирование системы аутентификации и авторизации

Цель: убедиться, что система аутентификации и авторизации работает корректно и предоставляет доступ к функциям в зависимости от роли пользователя.

Процедура:

- Проверка доступа к функциям для зарегистрированного пользователя.
- Проверка доступа к функциям для администратора.

- Проверка доступа к функциям для менеджера и поставщика услуг.

Результаты:

- Зарегистрированные пользователи имеют доступ только к своим функциям.
- Администраторы имеют доступ ко всем функциям системы.
- Менеджеры и поставщики услуг имеют доступ к функциям, соответствующим их ролям.

4. Командное тестирование

Цель: проверить корректность работы отдельных компонентов системы и их взаимодействие.

Процедура:

- Написание команд для тестирования моделей данных.
- Написание команд для тестирования представлений.
- Написание команд для тестирования маршрутизации.

Результаты:

- Модели данных работают корректно, данные сохраняются и извлекаются без ошибок.
- Представления корректно обрабатывают запросы и возвращают правильные ответы.
- Маршрутизация работает корректно, все URL-адреса соответствуют своим представлениям.

Проведенное тестирование показало, что система онлайн-бронирования работает корректно и соответствует требованиям. Все основные функции системы были проверены и работают без ошибок. Система аутентификации и авторизации обеспечивает правильный доступ к функциям в зависимости от роли пользователя. Командное тестирование подтвердило корректность работы отдельных компонентов системы и их взаимодействие.

Тестирование позволило выявить и исправить несколько незначительных ошибок, что повысило надежность и стабильность системы. В дальнейшем планируется продолжить тестирование и улучшение системы на основе отзывов пользователей и новых требований.

Оценка результатов и предложения по улучшению

В данной главе представлены результаты работы по разработке системы онлайн-бронирования для малого бизнеса на примере салона красоты. Оценены достигнутые результаты, а также предложены направления для дальнейшего улучшения системы.

Результаты работы

В ходе выполнения дипломной работы была разработана и внедрена система онлайн-бронирования, которая включает следующие основные компоненты:

1) Регистрация и аутентификация пользователей:
   - Реализована система регистрации новых пользователей.
   - Внедрена система аутентификации, обеспечивающая безопасный вход и выход из системы.
2) Управление профилями пользователей:
   - Созданы профили для различных типов пользователей: администраторов, менеджеров, поставщиков услуг и клиентов.
   - Реализована возможность редактирования профилей.
3) Управление услугами и бронированиями:
   - Разработан интерфейс для добавления, редактирования и удаления услуг.
   - Внедрена система бронирования, позволяющая клиентам выбирать услуги и назначать время посещения.
   - Администраторы и менеджеры могут управлять бронированиями, изменять и удалять их.
4) Отзывы и рейтинги:

- Реализована возможность оставлять отзывы и оценки для услуг и поставщиков услуг.
- Отзывы отображаются на страницах услуг и профилях поставщиков услуг.

5) Интерфейс пользователя:
   - Создан удобный и интуитивно понятный интерфейс для взаимодействия с системой.
   - Использованы современные технологии HTML, CSS и JavaScript для улучшения пользовательского опыта.

Оценка достигнутых результатов

Разработанная система онлайн-бронирования успешно решает поставленные задачи и предоставляет следующие преимущества:

1) Удобство для клиентов:
   - Клиенты могут легко находить и бронировать услуги через веб-интерфейс.
   - Возможность оставлять отзывы и оценки помогает клиентам делать осознанный выбор.
2) Эффективность управления для бизнеса:
   - Администраторы и менеджеры могут эффективно управлять услугами и бронированиями.
   - Система позволяет оптимизировать расписание и улучшить обслуживание клиентов.
3) Безопасность и надежность:
   - Реализована безопасная система аутентификации и авторизации.
   - Проведенное тестирование подтвердило надежность и стабильность работы системы.

Предложения по улучшению

Несмотря на достигнутые результаты, система может быть улучшена в следующих направлениях:

1) Интеграция с внешними сервисами:
   - Интеграция с платежными системами для онлайн-оплаты услуг.
   - Интеграция с системами уведомлений (SMS, email) для напоминаний о бронированиях.
2) Расширение функциональности:
   - Введение системы лояльности для постоянных клиентов.
   - Разработка мобильного приложения для удобства использования на смартфонах.
3) Оптимизация производительности:
   - Оптимизация запросов к базе данных для повышения скорости работы системы.
   - Внедрение кэширования для уменьшения нагрузки на сервер.
4) Улучшение пользовательского интерфейса:
   - Проведение UX-исследований для выявления и устранения проблем в интерфейсе.
   - Внедрение адаптивного дизайна для улучшения отображения на различных устройствах.
5) Аналитика и отчеты:
   - Разработка системы аналитики для отслеживания ключевых показателей бизнеса.
   - Внедрение отчетов для анализа эффективности работы и принятия управленческих решений.

Разработанная система онлайн-бронирования для малого бизнеса на примере салона красоты успешно решает поставленные задачи и предоставляет удобные инструменты для клиентов и администраторов. Проведенное тестирование подтвердило надежность и стабильность работы системы. В дальнейшем планируется продолжить развитие системы, внедряя новые функции и улучшая существующие, что позволит сделать её еще более полезной и эффективной для пользователей.

Заключение

В ходе выполнения данного проекта была разработана система онлайн-бронирования для малого бизнеса на примере салона красоты. Основной целью проекта было создание удобного и эффективного инструмента для автоматизации процесса записи клиентов на услуги, управления расписанием и улучшения взаимодействия между клиентами и администрацией салона.

Достигнутые результаты

Разработанная система включает в себя множество функциональных возможностей, которые обеспечивают удобство и эффективность работы как для клиентов, так и для сотрудников салона. Клиенты получили возможность легко и быстро бронировать услуги через веб-интерфейс, управлять своими бронированиями и оставлять отзывы о предоставленных услугах. Администраторы и менеджеры, в свою очередь, могут эффективно управлять расписанием, бронированиями и профилями пользователей, что значительно упрощает их работу и повышает качество обслуживания.

Одним из ключевых аспектов системы является безопасность. Реализованная система аутентификации и авторизации обеспечивает надежную защиту данных пользователей и предотвращает несанкционированный доступ к системе. Проведенное тестирование подтвердило надежность и стабильность работы системы.

Вклад в цифровизацию малого бизнеса

Проект продемонстрировал, как современные технологии могут быть использованы для улучшения бизнес-процессов в малом бизнесе. Внедрение системы онлайн-бронирования позволяет салонам красоты сократить время на организацию встреч, уменьшить вероятность ошибок при записи клиентов и повысить общую эффективность работы. Это особенно актуально в условиях

растущей конкуренции и необходимости предоставления высококачественного сервиса.

Перспективы развития

Несмотря на достигнутые результаты, система имеет потенциал для дальнейшего развития и улучшения. Внедрение интеграции с внешними сервисами, такими как платежные системы и системы уведомлений, позволит сделать процесс бронирования еще более удобным и автоматизированным. Разработка мобильного приложения обеспечит доступ к системе с любых устройств, что повысит её привлекательность для пользователей.

Оптимизация производительности системы, включая улучшение запросов к базе данных и внедрение кэширования, позволит повысить скорость работы и уменьшить нагрузку на сервер. Проведение UX-исследований и внедрение адаптивного дизайна помогут улучшить пользовательский интерфейс и сделать его более интуитивно понятным и удобным для всех категорий пользователей.

Влияние на бизнес-процессы

Внедрение системы онлайн-бронирования оказывает положительное влияние на бизнес-процессы салона красоты. Автоматизация записи клиентов позволяет сократить время на административные задачи, что освобождает ресурсы для более важной работы, такой как улучшение качества предоставляемых услуг и взаимодействие с клиентами. Система также предоставляет инструменты для анализа и улучшения бизнес-процессов, что способствует повышению эффективности работы и росту бизнеса.

Заключительные мысли

Проект по разработке системы онлайн-бронирования для салона красоты показал, что использование современных технологий может значительно улучшить управление бизнесом и повысить удовлетворенность клиентов.

Разработанная система является надежным и эффективным инструментом, который отвечает потребностям малого бизнеса и его клиентов. В дальнейшем планируется продолжить развитие системы, внедряя новые функции и улучшая существующие, что позволит сделать её еще более полезной и эффективной для пользователей.

Таким образом, данный проект является важным шагом на пути к цифровизации и автоматизации бизнес-процессов в малом бизнесе. Он демонстрирует, как современные технологии могут быть использованы для решения реальных бизнес-задач и улучшения качества обслуживания клиентов. Внедрение системы онлайн-бронирования позволяет салонам красоты не только повысить свою эффективность, но и предоставить клиентам удобный и современный сервис, что является ключевым фактором успеха в условиях растущей конкуренции.

Список использованной литературы

1) Книги и учебные пособия:
   - Адитья Бхаргава. "Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих". СПб.: Питер, 2017.
   - Марк Лутц. "Изучаем Python". 5-е издание. СПб.: Питер, 2017.
   - Эрик Мэтиз. "Изучаем Python. Программирование игр, визуализация данных, веб-приложения". 2-е издание. СПб.: Питер, 2019.
   - Уэс МакКинни. "Python и анализ данных". 2-е издание. СПб.: Питер, 2018.
2) Документация и официальные ресурсы:
   - Django Software Foundation. "Django Documentation". https://docs.djangoproject.com/
   - Python Software Foundation. "Python Documentation". https://docs.python.org/
   - SQLite Documentation. "SQLite Documentation". https://www.sqlite.org/docs.html
3) Статьи и онлайн-ресурсы:
   - Real Python. "Django Tutorials". https://realpython.com/tutorials/django/
   - MDN Web Docs. "HTML, CSS, and JavaScript Documentation". https://developer.mozilla.org/
   - Stack Overflow. "Questions and Answers on Django". https://stackoverflow.com/questions/tagged/django

Приложения

Код программы: Полный исходный код разработанной системы.

Ссылка на github с кодом проекта: https://github.com/Sent1mus/GB_Diplom
Ссылка на запущенный сервер с проектом: https://sent1mus.pythonanywhere.com/

```
#full_path_to_project\Project\requirements.txt
asgiref==3.8.1
chardet==5.2.0
Django==5.0.4
sqlparse==0.5.0
tzdata==2024.1

# templates\profile\user_booking_update.html
{% extends 'base.html' %}

{% block content %}
<div class="container user-booking-update">
    <h1>Update Booking</h1>
    <form method="post" id="bookingForm" class="form-inline">
        {% csrf_token %}
        {{ form.as_p }}<br>
        <button type="submit" class="form-button">Save Changes</button>
    </form>
    {% if error_message %}
    <p style="color: red; text-align: center;">{{ error_message }}</p>
    {% endif %}
</div>
{% endblock %}

# static\css\style.css
/* Основные стили для всего тела документа */
body {
    margin: 0; /* Убираем отступы */
    font-family: 'Fira Sans', sans-serif; /* Устанавливаем шрифт для текста
*/
    background-color: #333; /* Цвет фона темно-серый */
    color: #ccc; /* Цвет текста светло-серый */
}

/* Стили для нижнего колонтитула */
footer {
    color: gold; /* Цвет текста золотой */
    text-align: center; /* Выравнивание текста по центру */
    position: absolute; /* Абсолютное позиционирование */
    bottom: 0; /* Прижимаем к нижней части */
    width: 100%; /* Ширина на всю доступную область */
}
```

```css
/* Стили для основного содержимого */
main {
    padding: 200px; /* Внутренний отступ */
}

/* Стили для контейнера навигации */
.nav-container {
    height: 100px; /* Высота контейнера */
    width: 100%; /* Ширина на всю доступную область */
    position: fixed; /* Фиксированное позиционирование */
    top: 0; /* Прижимаем к верхней части */
    left: 0; /* Прижимаем к левой части */
    display: flex; /* Используем flexbox для выравнивания элементов */
    align-items: center; /* Центрирование элементов по вертикали */
    justify-content: center; /* Центрирование элементов по горизонтали */
    background-color: #111111; /* Цвет фона темно-черный */
    z-index: 1000; /* Устанавливаем z-index для наложения элементов */
    box-sizing: border-box; /* Размеры включают padding и border */
    font-size: 24px; /* Размер шрифта */
    text-align: center; /* Выравнивание текста по центру */
}

/* Стили для изображений в ссылках навигации */
.nav-link img {
    width: 100%; /* Ширина изображения на всю доступную область */
    height: 100px; /* Высота изображения фиксированная */
    object-fit: contain; /* Сохраняем пропорции изображения */
}

/* Стили для списка в навигационном контейнере */
.nav-container ul {
    display: flex; /* Используем flexbox для выравнивания элементов */
    align-items: center; /* Центрирование элементов по вертикали */
    justify-content: center; /* Центрирование элементов по горизонтали */
    list-style-type: none; /* Убираем маркеры списка */
    margin: 0; /* Убираем внешние отступы */
    padding: 0; /* Убираем внутренние отступы */
}

/* Стили для ссылок, кнопок и форм в навигационном контейнере */
.nav-container .nav-link, .nav-container .nav-button, .nav-container form {
    text-decoration: none; /* Убираем подчеркивание */
    color: gold; /* Цвет текста золотой */
    margin-left: 20px; /* Отступ слева */
    background-color: transparent; /* Прозрачный фон */
    border: none; /* Без рамки */
    cursor: pointer; /* Курсор в виде указателя */
    font-size: 24px; /* Размер шрифта */
    display: inline-flex; /* Вывод элементов в одну строку с возможностью
выравнивания */
    align-items: center; /* Центрирование элементов по вертикали */
}

/* Стили для ссылок и форм при наведении */
.nav-container .nav-link:hover, .nav-container .nav-button:hover, .nav-
container form:hover {
    color: white; /* Цвет текста белый при наведении */
    background-color: transparent; /* Фон остается прозрачным */
}

/* Центрирование кнопок в формах */
```

```
.nav-container form {
    display: flex; /* Используем flexbox для выравнивания элементов */
    align-items: center; /* Центрирование элементов по вертикали */
    justify-content: center; /* Центрирование элементов по горизонтали */
    margin: 0 20px; /* Горизонтальные отступы */
    padding: 5px 10px; /* Внутренние отступы */
    width: auto; /* Ширина по содержимому */
}

/* Стили для списка навигации */
.navlist {
    display: flex; /* Используем flexbox для выравнивания элементов */
    list-style: none; /* Убираем маркеры списка */
    padding: 0; /* Убираем внутренние отступы */
    margin: 0; /* Убираем внешние отступы */
    width: 100%; /* Ширина на всю доступную область */
    justify-content: space-around; /* Равномерное распределение элементов с
промежутками */
}

/* Стили для элементов списка в навигации */
.navlist li {
    flex-grow: 1; /* Элементы растягиваются для заполнения доступного
пространства */
    text-align: center; /* Выравнивание текста по центру */
    display: flex; /* Используем flexbox для выравнивания элементов */
    justify-content: center; /* Центрирование элементов по горизонтали */
    align-items: center; /* Центрирование элементов по вертикали */
}

/* Стили для изображений в навигации */
.nav-img {
    width: 100%; /* Ширина изображения на всю доступную область */
    height: auto; /* Высота автоматическая, сохраняется пропорция */
    max-height: 100px; /* Максимальная высота изображения */
}

/* Стили для содержимого заголовка */
.header-content {
    display: flex; /* Используем flexbox для выравнивания элементов */
    flex-direction: column; /* Элементы располагаются вертикально */
    align-items: center; /* Центрирование элементов по вертикали */
    justify-content: center; /* Центрирование элементов по горизонтали */
    text-align: center; /* Выравнивание текста по центру */
}

.services {
    display: flex;
    flex-direction: column;
    align-items: flex-end; /* Aligns children to the bottom */
}

/* Основные стили для таблицы услуг */
.services table {
    width: 80%; /* Ширина таблицы */
    margin: 20px auto; /* Центрирование таблицы с вертикальными отступами */
    border-collapse: collapse; /* Стиль границы, границы ячеек объединены */
    box-shadow: 0 4px 8px rgba(0,0,0,0.1); /* Тень для таблицы */
    border: 1px solid gold; /* Граница золотого цвета */
}
```

```
/* Стили для ячеек и заголовков в таблице услуг */
.services td, .services th {
    position: relative;
    border: 1px solid gold; /* Граница золотого цвета */
    padding: 8px; /* Внутренние отступы в ячейках */
    text-align: left; /* Выравнивание текста по левому краю */
}

/* Стили для заголовков в блоке услуг */
.services h2 {
    color: gold; /* Цвет текста золотой */
    font-size: 24px; /* Размер шрифта */
}

/* Стили для параграфов в блоке услуг */
.services p {
    color: #ccc; /* Цвет текста серый */
    margin: 5px 0; /* Отступы сверху и снизу для разделения параграфов */
}

/* Стили для форм внутри таблицы услуг */
.services form {
    display: block; /* Форма занимает всю ширину ячейки */
}

/* Стили для элементов ввода, выбора и кнопок внутри форм */
.services input, .services select, .services button {
    width: 100%; /* Ширина элементов на всю доступную ширину */
    padding: 10px; /* Внутренний отступ для лучшего взаимодействия */
    box-sizing: border-box; /* Размеры включают padding и border */
}

/* Стили для кнопок в блоке услуг */
.services button {
    font-size: 16px;
    background-color: gold; /* Фоновый цвет кнопок золотой */
    color: black; /* Цвет текста черный */
    border: none; /* Без рамки для более чистого вида */
    cursor: pointer; /* Курсор в виде указателя */
    transition: background-color 0.3s; /* Плавный переход цвета при
наведении */
}

/* Стили для кнопок при наведении */
.services .link-as-button {
    font-size: 16px;
    position: absolute; /* Абсолютное позиционирование относительно
ближайшего позиционированного предка */
    bottom: 8px; /* Прикрепляет кнопку к нижней части */
    left: 8px; /* Выравнивает кнопку по левому краю */
    right: 8px; /* Выравнивает кнопку по правому краю */
    height: 35px; /* Высота кнопки */
    width: auto; /* Ширина в зависимости от содержимого */
    background-color: gold; /* Цвет фона */
    color: black; /* Цвет текста */
    text-align: center; /* Центрирование текста внутри кнопки */
    line-height: 35px; /* Высота строки текста */
    text-decoration: none; /* Убирает подчеркивание у ссылок */
    border: none; /* Без рамки */
    cursor: pointer; /* Курсор в виде указателя при наведении */
    transition: background-color 0.3s; /* Плавный переход цвета фона */
```

```
}

.services .link-as-button:hover, button:hover {
    background-color: #d4af37; /* Темно-золотой цвет при наведении */
}

/* Стили для сообщений об ошибках */
.services p[style*="color: red;"] {
    font-weight: bold; /* Жирный шрифт для сообщений */
    color: #ff0000; /* Цвет текста ярко-красный */
}

/* Стили для центрированной формы с таблицей на странице регистрации */
.signup-container {
    display: flex; /* Использование flexbox для выравнивания */
    flex-direction: column; /* Элементы расположены вертикально */
    justify-content: center; /* Центрирование элементов по вертикали */
    align-items: center; /* Центрирование элементов по горизонтали */
    width: 100%; /* Ширина контейнера на всю доступную ширину */
}

/* Стили для заголовков в контейнере регистрации */
.signup-container h2 {
    margin-bottom: 20px; /* Отступ снизу для разделения заголовка и формы */
}

/* Стили для таблицы в форме регистрации */
.signup-container table {
    border-collapse: collapse; /* Стиль границы, границы ячеек объединены */
    margin: auto; /* Центрирование таблицы */
    width: 100%; /* Ширина таблицы на всю доступную ширину */
}

/* Стили для ячеек в таблице регистрации */
.signup-container td {
    padding: 8px; /* Внутренние отступы в ячейках */
    text-align: left; /* Выравнивание текста по левому краю */
}

/* Стили для первой ячейки в строках таблицы регистрации */
.signup-container td:first-child {
    text-align: right; /* Выравнивание текста по правому краю */
    padding-right: 20px; /* Отступ справа */
}

/* Стили для кнопок регистрации */
.signup-container .register-button {
    width: 100%; /* Ширина кнопки на всю доступную ширину */
    padding: 10px 0; /* Вертикальный внутренний отступ для удобства нажатия
*/
}

/* Стили для контейнера страницы входа */
.login-container {
    display: flex; /* Использование flexbox для выравнивания */
    flex-direction: column; /* Элементы расположены вертикально */
    justify-content: center; /* Центрирование элементов по вертикали */
    align-items: center; /* Центрирование элементов по горизонтали */
}

/* Стили для заголовков на странице входа */
```

```
.login-container h2 {
    margin-bottom: 20px; /* Отступ снизу для разделения заголовка и формы */
}

/* Стили для таблицы на странице входа */
.login-container table {
    width: 300px; /* Фиксированная ширина таблицы */
    border-collapse: collapse; /* Стиль границы, границы ячеек объединены */
}

/* Стили для ячеек на странице входа */
.login-container td {
    padding: 8px; /* Внутренние отступы в ячейках */
}

/* Стили для кнопок входа */
.login-container .login-button {
    width: 100%; /* Ширина кнопки на всю доступную ширину */
    padding: 10px 0; /* Вертикальный внутренний отступ для удобства нажатия
*/
}

/* Основные стили для контейнера профиля пользователя */
.user-profile-container {
    display: flex; /* Использование flexbox для выравнивания */
    flex-direction: column; /* Элементы расположены вертикально */
    align-items: center; /* Центрирование элементов по горизонтали */
    justify-content: center; /* Центрирование элементов по вертикали */
    width: 100%; /* Ширина контейнера на всю доступную ширину */
    padding: 20px; /* Внутренний отступ */
    color: #ccc; /* Цвет текста светло-серый */
    background-color: #333; /* Цвет фона темно-серый */
}

/* Стили для каждого раздела в профиле пользователя */
.profile-section {
    width: 50%; /* Ширина раздела 50% от ширины контейнера */
    margin: 20px 0; /* Вертикальные отступы для разделения разделов */
    padding: 20px; /* Внутренний отступ */
    background-color: #161618; /* Цвет фона темно-серый */
    box-shadow: 0 4px 8px rgba(0,0,0,0.1); /* Тень для элемента */
    display: flex; /* Использование flexbox для выравнивания */
    flex-direction: column; /* Элементы расположены вертикально */
    align-items: center; /* Центрирование элементов по горизонтали */
    justify-content: center; /* Центрирование элементов по вертикали */
}

/* Стили для строк в разделах профиля */
.profile-row {
    display: flex; /* Использование flexbox для выравнивания */
    justify-content: space-between; /* Распределение элементов с
промежутками */
    width: 100%; /* Ширина строки на всю доступную ширину */
    margin-bottom: 10px; /* Отступ снизу для разделения строк */
}

/* Стили для колонок в строках профиля */
.profile-column {
    display: flex; /* Использование flexbox для выравнивания */
    flex-direction: column; /* Элементы расположены вертикально */
    justify-content: center; /* Центрирование элементов по вертикали */
```

```
    width: 50%; /* Ширина колонки 50% от ширины строки */
}

/* Стили для ввода данных, кнопок редактирования и сохранения в профиле */
.profile-input, .edit-button, .save-button {
    display: none; /* Элементы изначально скрыты, отображаются через
JavaScript */
}

/* Секция профиля: кнопки и поля ввода */
.profile-section button, .profile-input, input[type="email"],
input[type="text"], input[type="password"] {
    border: 0px;
    width: auto; /* Автоматическая ширина в зависимости от содержимого */
    padding: 10px; /* Отступы вокруг текста */
    background-color: gold; /* Золотой фон */
    color: #333; /* Цвет текста темно-серый */
}

/* Изменение фона кнопок при наведении */
.profile-section button:hover {
    background-color: #d4af37; /* Темно-золотой фон при наведении */
}

/* Стилизация кнопок и полей ввода */
.edit-button, .save-button, .profile-button, input[type="email"],
input[type="text"], input[type="password"] {
    background-color: #fff; /* Белый фон */
    color: #333; /* Цвет текста темно-серый */
    border: none; /* Без границ */
    cursor: pointer; /* Курсор в виде указателя */
    transition: background-color 0.3s; /* Плавное изменение фона */
}

.profile-section button{
    font-size: 16px;
}
/* Изменение фона при наведении на кнопки и поля ввода */
.edit-button:hover, .save-button:hover, .profile-button:hover,
input[type="email"]:hover, input[type="text"]:hover,
input[type="password"]:hover {
    background-color: #d4af37; /* Темно-золотой фон при наведении */
}

/* Расположение кнопок в строке */
.profile-button-row {
    display: flex; /* Элементы в строке */
    justify-content: center; /* Центрирование по горизонтали */
    width: 100%; /* Ширина 100% */
    margin: 10px 0; /* Отступ сверху и снизу */
}

/* Стилизация ссылок как кнопок */
.profile-button-row .link-as-button, .profile-button-row .delete-profile-
button {
    display: inline-block; /* Элементы в линию */
    font-size: 16px;
    width: 50%; /* Ширина 50% от контейнера */
    background-color: #007bff; /* Синий фон */
    color: white; /* Белый текст */
    text-align: center; /* Текст по центру */
```

```
    padding: 10px 0px; /* Отступы вокруг текста */
    text-decoration: none; /* Без подчеркивания текста */
    transition: background-color 0.3s; /* Плавное изменение фона */
}

/* Изменение фона ссылки при наведении */
.profile-button-row .link-as-button:hover {
    background-color: #0056b3; /* Темно-синий фон при наведении */
}

/* Красный фон для кнопки удаления профиля */
.profile-button-row .delete-profile-button {
    background-color: #ff0000; /* Красный фон */
}

/* Темно-красный фон при наведении на кнопку удаления */
.profile-button-row .delete-profile-button:hover {
    background-color: #8b0000; /* Темно-красный фон при наведении */
}

/* Базовый блок для истории заказов */
.order-history {
    width: 50%; /* Ширина 50% */
    margin: auto; /* Центрирование */
}

/* Заголовок в блоке истории заказов */
.order-history__header {
    text-align: center; /* Текст по центру */
}

/* Контейнер для таблицы в блоке истории заказов */
.order-history__table-container {
    margin: 20px; /* Отступы вокруг контейнера */
    background-color: transparent; /* Прозрачный фон */
}

/* Стилизация таблицы в блоке истории заказов */
.order-history__table {
    width: 80%; /* Ширина 80% */
    margin-left: auto; /* Отступ слева автоматический */
    margin-right: auto; /* Отступ справа автоматический */
    border-collapse: collapse; /* Убрать промежутки между ячейками */
}

/* Заголовки столбцов в таблице */
.order-history__column-header {
    padding: 10px; /* Отступы вокруг текста */
    text-align: center; /* Текст по центру */
}

/* Строки и ячейки в таблице */
.order-history__row {}
.order-history__cell {
    height: 30px; /* Высота ячейки 30px */
    padding: 3px; /* Отступы вокруг текста */
    border: 1px solid #ddd; /* Граница светло-серого цвета */
}

/* Контейнер для ссылок действий */
.order-history__actions-container {
```

```
    display: flex; /* Элементы в строке */
    width: 100%; /* Ширина 100% */
}

/* Стилизация ссылок в ячейках таблицы */
.order-history__link {
    flex: 1; /* Заполнить доступное пространство */
    text-align: center; /* Текст по центру */
    padding: 8px 12px; /* Отступы вокруг текста */
    color: white; /* Белый текст */
    text-decoration: none; /* Без подчеркивания текста */
    display: inline-block; /* Элементы в линию */
}

/* Стилизация ссылки "Оставить отзыв" */
.order-history__link--review {
    position: relative; /* Относительное позиционирование для правильного
выравнивания внутри ячейки таблицы */
    display: inline-block; /* Отображение в виде блока для правильного
размера */
    padding: 8px 12px; /* Отступы вокруг текста */
    background-color: gold; /* Золотой цвет фона */
    color: black; /* Черный цвет текста */
    text-align: center; /* Выравнивание текста по центру */
    text-decoration: none; /* Без подчеркивания */
    border: none; /* Без рамки */
    cursor: pointer; /* Курсор в виде указателя при наведении */
    transition: background-color 0.3s; /* Плавный переход цвета фона */
}

.order-history__link--review:hover {
    background-color: #d4af37; /* Темно-золотой цвет при наведении */
}

/* Модификатор для ссылки редактирования */
.order-history__link--edit {
    background-color: #007bff; /* Синий фон */
}
.order-history__link--edit:hover{
    background-color: #0056b3; /* Темно-синий фон при наведении */
}

/* Модификатор для ссылки удаления */
.order-history__link--delete {
    background-color: #ff0000; /* Красный фон */
}
.order-history__link--delete:hover{
    background-color: #8b0000; /* Темно-красный фон при наведении */
}

/* Блок обновления бронирования пользователя */
.user-booking-update {
    width: 50%; /* Ширина 50% */
    margin: 0 auto; /* Центрирование */
    padding: 20px; /* Отступы вокруг блока */
    background-color: transparent; /* Прозрачный фон */
}

/* Заголовок в блоке обновления бронирования */
.user-booking-update h1 {
    color: #ccc; /* Цвет текста светло-серый */
```

```
    text-align: center; /* Текст по центру */
}

/* Форма в блоке обновления бронирования */
.user-booking-update .form-inline {
    display: flex; /* Элементы в строке */
    flex-direction: row; /* Элементы ориентированы горизонтально */
    justify-content: center; /* Центрирование по горизонтали */
    align-items: center; /* Центрирование по вертикали */
    gap: 10px; /* Расстояние между элементами */
}

/* Выпадающие списки в форме */
.user-booking-update form select {
    width: 150px; /* Фиксированная ширина */
    padding: 10px; /* Отступы для удобства клика */
    text-overflow: ellipsis; /* Текст не выходит за границы */
}

/* Кнопка в форме */
.user-booking-update .form-button {
    font-size: 16px;
    display: block;
    width: 200px; /* Фиксированная ширина */
    height: calc(2 * 40px); /* Высота в два раза больше стандартной высоты
поля */
    background-color: #ffdd57; /* Желтый фон */
    color: black; /* Черный текст для контраста */
}

/* Стили для контейнера отзывов */
.reviews-container {
    width: 50%; /* Ширина контейнера */
    margin: 20px auto; /* Центрирование контейнера с вертикальными отступами
*/
    text-align: center; /* Выравнивание текста по центру */
    background-color: #333; /* Темно-серый фон */
    color: #ccc; /* Светло-серый цвет текста */
    padding: 20px; /* Внутренние отступы */
}

/* Стили для таблицы в контейнере отзывов */
.reviews-container table {
    width: 100%; /* Ширина таблицы на всю доступную ширину контейнера */
    margin: auto; /* Автоматическое центрирование таблицы */
    border-collapse: collapse; /* Убираем промежутки между ячейками */
    border: 2px solid gold; /* Золотая граница вокруг таблицы */
}

/* Стили для заголовков ячеек в таблице отзывов */
.reviews-container th {
    text-align: center; /* Выравнивание текста заголовков по центру */
    border-bottom: 1px solid gold; /* Золотая граница снизу заголовков */
    padding: 10px; /* Внутренние отступы для заголовков */
    background-color: #161618; /* Темно-серый фон для заголовков */
    color: gold; /* Золотой цвет текста для заголовков */
}

/* Стили для ячеек в таблице отзывов */
.reviews-container td {
    padding: 8px; /* Внутренние отступы в ячейках */
```

```
    text-align: center; /* Выравнивание текста в ячейках по центру */
    border: 1px solid gold; /* Золотая граница вокруг каждой ячейки */
    background-color: #222; /* Темно-серый фон для ячеек */
}

/* Стили для заголовка в контейнере отзывов */
.reviews-container h1 {
    margin-bottom: 20px; /* Отступ снизу для разделения заголовка и таблицы
*/
    color: gold; /* Золотой цвет текста для заголовка */
}

/* Стили для сообщения, когда отзывов нет */
.reviews-container h3 {
    color: #ff0000; /* Ярко-красный цвет текста для сообщения об отсутствии
отзывов */
}

/* Стили для select элемента (rating) */
.reviews-container select{
    width: 100%; /* Ширина на всю доступную ширину контейнера */
    height: 40px; /* Высота элемента */
    font-size: 16px; /* Размер шрифта */
    color: black; /* Цвет текста */
    background-color: #fff; /* Белый фон */
    border: 1px solid #ccc; /* Серая граница */
    padding: 5px 10px; /* Внутренние отступы */
}

/* Стили для textarea (comment) */
.reviews-container textarea {
    width: 100%; /* Ширина на всю доступную ширину контейнера */
    height: 160px; /* Высота в четыре раза больше, чем у select */
    font-size: 16px; /* Размер шрифта */
    color: black; /* Цвет текста */
    background-color: #fff; /* Белый фон */
    border: 1px solid #ccc; /* Серая граница */
}

/* Стили для кнопки "Submit Review" */
.reviews-container button {
    width: 100%; /* Ширина на всю доступную ширину контейнера */
    height: 40px; /* Высота, равная select */
    font-size: 16px; /* Размер шрифта, равный select */
    color: black; /* Цвет текста */
    background-color: gold; /* Золотой фон */
    border: none; /* Без рамки */
    cursor: pointer; /* Курсор в виде указателя при наведении */
    transition: background-color 0.3s; /* Плавный переход цвета фона */
}

.reviews-container button:hover {
    background-color: #d4af37; /* Темно-золотой цвет при наведении */
}

/* Стили для меток в форме отзывов */
.label-rating, .label-comment {
    font-size: 16px; /* Увеличенный размер шрифта для меток */
}

# templates\profile\all_bookings.html
```

```
{% extends 'base.html' %}

{% block content %}
<div class="all-bookings-container">
    <h1>All Bookings</h1>
    <table>
        <thead>
            <tr>
                <th>Service</th>
                <th>Customer</th>
                <th>Service Provider</th>
                <th>Appointment Date and Time</th>
                <th>Actions</th>
            </tr>
        </thead>
        <tbody>
            {% for booking in bookings %}
            <tr>
                <td>{{ booking.service.name }}</td>
                <td>{{ booking.customer.user.username }}</td>
                <td>{{ booking.service_provider.user.username }}</td>
                <td>{{ booking.appointment_datetime|date:"Y-m-d H:i" }}</td>
                <td>
                    <a href="{% url 'admin_booking_update' booking.id %}"
class="btn btn-primary">Edit</a>
                    <a href="{% url 'admin_booking_delete' booking.id %}"
class="btn btn-danger">Delete</a>
                </td>
            </tr>
            {% empty %}
            <tr>
                <td colspan="5">No bookings found.</td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
</div>
{% endblock %}


# views\auth.py
from django.contrib.auth import authenticate, login, logout
from django.shortcuts import render, redirect
from ..forms import RegisterForm


def signup(request):
    # Handle user signup
    if request.method == 'POST':
        form = RegisterForm(request.POST)
        if form.is_valid():
            user = form.save()  # Save the new user
            login(request, user)  # Log in the new user
            return redirect('/profile/')  # Redirect to profile page after
signup
        else:
            return render(request, 'signup.html', {'form': form})  # Re-
render the signup form with errors
    else:
        form = RegisterForm()
        return render(request, 'signup.html', {'form': form})  # Display the
```

```
empty signup form


def user_login(request):
    # Handle user login
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']
        user = authenticate(request, username=username, password=password)
# Authenticate user
        if user is not None:
            login(request, user)  # Log in the user
            return redirect('main')  # Redirect to main page after login
        else:
            return render(request, 'login.html', {'error': 'Invalid username
or password'})  # Show error if login fails
    else:
        return render(request, 'login.html')  # Display the login form


def user_logout(request):
    logout(request)  # Log out the user
    return redirect('main')  # Redirect to main page after logout


# views\__init__.py
import datetime

from django.shortcuts import render

from ..models import *
from ..views import *


def main_page(request):
    # Render the main page template
    return render(request, 'main.html')


def about_us(request):
    # Render the about us page template
    return render(request, 'about_us.html')


def get_customers():
    # Retrieve all customer records from the database
    return Customer.objects.all()


def get_administrators():
    # Retrieve all administrator records from the database
    return Administrator.objects.all()


def get_service_providers():
    # Retrieve all service provider records from the database
    return ServiceProvider.objects.all()


def get_services():
    # Retrieve all service records from the database
```

```
    return Service.objects.all()


def get_reviews():
    # Retrieve all review records from the database
    return Review.objects.all()


def get_bookings():
    # Retrieve all booking records from the database
    return Booking.objects.all()


class BookingValidator:
    @staticmethod
    def is_slot_available(service_provider_id, appointment_datetime):
        # Calculate the end time of the appointment
        end_time = appointment_datetime + datetime.timedelta(hours=1)
        # Check for any overlapping bookings
        overlapping_bookings = Booking.objects.filter(
            service_provider_id=service_provider_id,
            appointment_datetime__lt=end_time,
            appointment_datetime__gte=appointment_datetime
        ).exists()
        # Return True if no overlapping bookings are found
        return not overlapping_bookings


def temp_list_all_db(request):
    # Prepare context with all database entities for rendering
    context = {
        'customers': get_customers(),
        'administrators': get_administrators(),
        'service_providers': get_service_providers(),
        'services': get_services(),
        'reviews': get_reviews(),
        'bookings': get_bookings()
    }
    # Render the template with all database entities listed
    return render(request, 'temp_list_all_db.html', context)


# management\commands\__init__.py


# management\commands\run_all_fill_scripts.py
from django.core.management.base import BaseCommand
from django.core.management import call_command


class Command(BaseCommand):
    help = 'Run migrations and all fill scripts for customers,
administrators, and service providers'

    def handle(self, *args, **options):
        # Run makemigrations and migrate to ensure the database is up to
date
        self.stdout.write(self.style.SUCCESS('Running makemigrations...'))
        call_command('makemigrations')
        call_command('makemigrations', 'booking')
        self.stdout.write(self.style.SUCCESS('Running migrate...'))
```

```
        call_command('migrate')
        self.stdout.write(self.style.SUCCESS('Database migration
completed.'))

        # Call fill_customers command
        self.stdout.write(self.style.SUCCESS('Starting to fill
customers...'))
        call_command('fill_customers')
        self.stdout.write(self.style.SUCCESS('Finished filling customers.'))

        # Call fill_administrators command
        self.stdout.write(self.style.SUCCESS('Starting to fill
administrators...'))
        call_command('fill_administrators')
        self.stdout.write(self.style.SUCCESS('Finished filling
administrators.'))

        # Call fill_managers command
        self.stdout.write(self.style.SUCCESS('Starting to fill
managers...'))
        call_command('fill_managers')
        self.stdout.write(self.style.SUCCESS('Finished filling managers.'))

        # Call fill_service_providers command
        self.stdout.write(self.style.SUCCESS('Starting to fill service
providers...'))
        call_command('fill_service_providers')
        self.stdout.write(self.style.SUCCESS('Finished filling service
providers.'))

        # Call fill_services command
        self.stdout.write(self.style.SUCCESS('Starting to fill
services...'))
        call_command('fill_services')
        self.stdout.write(self.style.SUCCESS('Finished filling services.'))

        self.stdout.write(self.style.SUCCESS('Starting to fill reviews...'))
        call_command('fill_reviews')
        self.stdout.write(self.style.SUCCESS('Finished filling reviews.'))

        self.stdout.write(self.style.SUCCESS('Starting to fill
bookings...'))
        call_command('fill_bookings')
        self.stdout.write(self.style.SUCCESS('Finished filling bookings.'))

        self.stdout.write(self.style.SUCCESS('Starting to fill
manytomany...'))
        call_command('fill_manytomany')
        self.stdout.write(self.style.SUCCESS('Finished filling
manytomany.'))


# templates\profile\user_profile.html
{% extends 'base.html' %}

{% load static %}

{% block content %}
<div class="user-profile-container">
    <h1>Welcome to Your Profile</h1>
    <div class="profile-section">
```

```
        <div class="profile-row">
            <div class="profile-column">
                <span>Your Email: </span>
                <span id="email-value">{{ user.email }}</span>
            </div>
            <div class="profile-column">
                <button onclick="toggleEdit('email')">Edit</button>
                <input type="email" id="email-input" value="{{ user.email
}}" style="display: inline;">
                <button onclick="save('email')"
style="display:none;">Save</button>
            </div>
        </div>
        <div class="profile-row">
            <div class="profile-column">
                <span>Your Phone: </span>
                <span id="phone-value">{{ user.customer.phone }}</span>
            </div>
            <div class="profile-column">
                <button onclick="toggleEdit('phone')">Edit</button>
                <input type="text" id="phone-input" value="{{
user.customer.phone }}" style="display:none;">
                <button onclick="save('phone')"
style="display:none;">Save</button>
            </div>
        </div>
        <div class="profile-row">
            <div class="profile-column">
                <span>Date of Birth: </span>
                <span>{{ user.customer.birth_date|date:"j F Y" }}</span>
            </div>
            <div class="profile-column">
                <button onclick="togglePasswordChange()">Change
Password</button>
                <div id="password-change" style="display:none;">
                    <input type="password" id="old-password"
placeholder="Old Password">
                    <input type="password" id="new-password"
placeholder="New Password">
                    <input type="password" id="confirm-password"
placeholder="Confirm Password">
                    <button onclick="changePassword()">Save
Password</button>
                </div>
            </div>
        </div>
        <div class="profile-button-row">
            <a href="{% url 'list_booking' %}" class="link-as-button
booking-button">View Booking History</a>
        </div>
        <div class="profile-button-row">
            <button onclick="deactivateProfile()" class="link-as-button
delete-profile-button">Delete Profile</button>
        </div>
    </div>
</div>
<!-- Hidden input to store URLs for JavaScript -->
<div id="data-container"
     data-update-profile-url="{% url 'update_profile' %}"
     data-change-password-url="{% url 'change_password' %}"
     data-deactivate-profile-url="{% url 'deactivate_profile' %}">
```

```
</div>

<script src="../static/js/profile/user_profile.js"></script>
{% endblock %}


# templates\profile\admin_booking_update.html
{% extends 'base.html' %}

{% block content %}
<div class="container">
    <h1>Edit Booking</h1>
    {% if messages %}
        <div class="alert alert-danger">
            {% for message in messages %}
                <p style="color:red">{{ message }}</p>
            {% endfor %}
        </div>
    {% endif %}
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        {{ provider_form.as_p }}
        {{ service_form.as_p }}
        <button type="submit" class="btn btn-primary">Save Changes</button>
    </form>
</div>
{% endblock %}


# views\service.py
from django.shortcuts import render, redirect
from django.contrib.auth.decorators import login_required
from . import BookingValidator
from ..forms import ServiceProviderForm, BookingDateTimeForm
from ..models import Service, Booking


@login_required  # Ensures that only authenticated users can access this
view
def services(request):
    services = Service.objects.all()  # Retrieve all service objects from
the database
    forms_list = []  # Initialize an empty list to store forms for each
service

    # Iterate over each service to create and manage forms
    for service in services:
        # Create forms with a unique prefix based on service ID
        provider_form = ServiceProviderForm(prefix=f"service_{service.id}")
        date_time_form = BookingDateTimeForm(prefix=f"service_{service.id}")
        error_message = None  # Variable to store error messages
        current_data = None  # Variable to store current form data if needed

        # Check if the form is submitted and corresponds to the current
service
        if request.method == 'POST' and request.POST.get('service_id') ==
str(service.id):
            # Populate forms with POST data
            provider_form = ServiceProviderForm(request.POST,
prefix=f"service_{service.id}")
```

```
            date_time_form = BookingDateTimeForm(request.POST,
prefix=f"service_{service.id}")

            # Validate forms
            if provider_form.is_valid() and date_time_form.is_valid():
                # Extract cleaned data from forms
                service_provider =
provider_form.cleaned_data['service_provider']
                appointment_datetime =
date_time_form.cleaned_data['appointment_datetime']

                # Check if the selected time slot is available
                if BookingValidator.is_slot_available(service_provider.id,
appointment_datetime):
                    # Create and save a new booking
                    booking = Booking(
                        customer=request.user.customer,
                        service=service,
                        service_provider=service_provider,
                        appointment_datetime=appointment_datetime
                    )
                    booking.save()
                    return redirect('/profile/list_booking')  # Redirect
user after booking
                else:
                    # Set error message if the time slot is not available
                    error_message = "Sorry, but this time has already been
booked."
                    # Store current form data to repopulate the form
                    current_data = {
                        'service_provider':
provider_form.cleaned_data['service_provider'],
                        'appointment_datetime':
date_time_form.cleaned_data['appointment_datetime']
                    }

        # Append the service and its forms to the list
        forms_list.append((service, provider_form, date_time_form,
error_message, current_data))

    # Render the page with the list of services and their forms
    return render(request, 'services.html', {
        'forms_list': forms_list,
    })


# templates\profile\user_booking_delete.html
{% extends 'base.html' %}

{% block content %}
<div class="container">
    <h1>Delete Booking</h1>
    <p>Are you sure you want to delete this booking?</p>
    <form method="post">
        {% csrf_token %}
        <button type="submit">Delete</button>
    </form>
</div>
{% endblock %}

# management\commands\clear_db.py
```

```
import os
from django.core.management.base import BaseCommand
from django.conf import settings
from django.db import connection
from django.db.migrations.recorder import MigrationRecorder

class Command(BaseCommand):
    help = 'Clears all migration files, resets the migration history, and
deletes the database.'

    def handle(self, *args, **options):
        # Удаление файлов миграций
        for app in settings.INSTALLED_APPS:
            try:
                app_path = os.path.join(settings.BASE_DIR, app.split('.')[-
1])
                migrations_path = os.path.join(app_path, 'migrations')
                if os.path.exists(migrations_path):
                    for file in os.listdir(migrations_path):
                        file_path = os.path.join(migrations_path, file)
                        if file != '__init__.py' and file.endswith('.py')
and os.path.isfile(file_path):
                            os.remove(file_path)
                            self.stdout.write(self.style.SUCCESS(f'Deleted
{file} from {migrations_path}'))
                else:
                    self.stdout.write(self.style.WARNING(f'No migrations
directory found at {migrations_path}'))
            except Exception as e:
                self.stdout.write(self.style.ERROR(f'Error processing {app}:
{str(e)}'))

        # Очистка истории миграций в базе данных
        MigrationRecorder.Migration.objects.all().delete()
        self.stdout.write(self.style.SUCCESS('Cleared all migration history
from the database.'))



# admin.py
from django.contrib import admin
from .models import Administrator, Manager, ServiceProvider, Customer,
Service, Booking, Review

# Register your models here.
admin.site.register(Administrator)
admin.site.register(Manager)
admin.site.register(Customer)
admin.site.register(Service)
admin.site.register(Booking)
admin.site.register(Review)


class ServiceProviderAdmin(admin.ModelAdmin):
    list_display = ('user', 'specialization', 'phone')
    search_fields = ('user__username', 'specialization')


admin.site.register(ServiceProvider, ServiceProviderAdmin)
```

```
# management\commands\fill_services.py
from datetime import timedelta
from django.core.management.base import BaseCommand
from booking.models import Service

class Command(BaseCommand):
    help = 'Ensures specific services are in the Service database'

    def handle(self, *args, **options):
        # Define a list of services with their details
        services_data = [
            {
                'name': 'Haircut',
                'description': 'Basic haircut',
                'duration': timedelta(hours=0, minutes=30),
                'price': 1000.00
            },
            {
                'name': 'Manicure',
                'description': 'Nail shaping and cuticle care',
                'duration': timedelta(hours=0, minutes=45),
                'price': 2000.00
            },
            {
                'name': 'Facial',
                'description': 'Skin cleansing and facial treatment',
                'duration': timedelta(hours=1, minutes=00),
                'price': 3000.00
            }
        ]

        # Iterate over the services data
        for service_data in services_data:
            # Use get_or_create to avoid creating duplicate entries
            service, created = Service.objects.get_or_create(
                name=service_data['name'],
                defaults={
                    'description': service_data['description'],
                    'duration': service_data['duration'],
                    'price': service_data['price']
                }
            )

            # Check if the service was created or fetched
            if created:
                message = f'Successfully created service {service.name}'
            else:
                message = f'Service {service.name} already exists'

            # Print success message
            self.stdout.write(self.style.SUCCESS(message))


# forms.py
import calendar
from django import forms
from django.contrib.auth.forms import UserCreationForm, PasswordChangeForm
from .models import *


# Form for user registration
```

```
class RegisterForm(UserCreationForm):
    # Define form fields with placeholders
    first_name = forms.CharField(max_length=30, required=True,

widget=forms.TextInput(attrs={'placeholder': 'First Name'}))
    last_name = forms.CharField(max_length=30, required=True,
                                widget=forms.TextInput(attrs={'placeholder':
'Last Name'}))
    email = forms.EmailField(max_length=75, required=True,
widget=forms.EmailInput(attrs={'placeholder': 'Email'}))
    phone = forms.CharField(max_length=20, required=True,
widget=forms.TextInput(attrs={'placeholder': 'Phone Number'}))
    birth_date = forms.DateField(required=True, input_formats=['%d.%m.%Y'],
                                 widget=forms.DateInput(format='%d.%m.%Y',
attrs={'placeholder': 'DD.MM.YYYY'}))

    class Meta:
        model = User
        fields = ('first_name', 'last_name', 'username', 'email', 'phone',
'password1', 'password2')

    def __init__(self, *args, **kwargs):
        super(RegisterForm, self).__init__(*args, **kwargs)
        # Update placeholders for username and password fields
        self.fields['username'].widget.attrs.update({'placeholder':
'Username'})
        self.fields['password1'].widget.attrs.update({'placeholder':
'Password'})
        self.fields['password2'].widget.attrs.update({'placeholder':
'Confirm Password'})

    def save(self, commit=True):
        user = super().save(commit=False)
        user.first_name = self.cleaned_data['first_name']
        user.last_name = self.cleaned_data['last_name']
        user.email = self.cleaned_data['email']
        if commit:
            user.save()
            # Create a customer profile linked to the user
            Customer.objects.create(user=user,
phone=self.cleaned_data['phone'],

birth_date=self.cleaned_data['birth_date'])
        return user


# Form for editing user profile
class EditProfileForm(forms.ModelForm):
    email = forms.EmailField(required=True)

    class Meta:
        model = Customer
        fields = ['phone', 'email']

    def __init__(self, *args, **kwargs):
        super(EditProfileForm, self).__init__(*args, **kwargs)
        # Initialize email field with user's current email
        self.fields['email'].initial = self.instance.user.email

    def save(self, commit=True):
        instance = super().save(commit=False)
```

```
        instance.user.email = self.cleaned_data['email']
        if commit:
            instance.user.save()
            instance.save()
        return instance


# Custom form for changing password
class CustomPasswordChangeForm(PasswordChangeForm):
    class Meta:
        model = User
        fields = ('old_password', 'new_password1', 'new_password2')


# Custom field for displaying model choice fields
class CustomModelChoiceField(forms.ModelChoiceField):
    def label_from_instance(self, obj):
        return f"{obj.user.first_name} {obj.user.last_name} -
{obj.specialization}"


# Form for selecting a service provider
class ServiceProviderForm(forms.Form):
    service_provider =
CustomModelChoiceField(queryset=ServiceProvider.objects.all(),

widget=forms.Select(attrs={'id': 'service_provider'}),
                                                label="Select master")


# Form for selecting a service
class ServiceForm(forms.Form):
    service = forms.ModelChoiceField(queryset=Service.objects.all(),
label="Select Service")


# Form for booking date and time
class BookingDateTimeForm(forms.Form):
    current_year = timezone.now().year
    current_month = timezone.now().month
    current_day = timezone.now().day
    current_hour = timezone.now().hour

    # Adjust month choices based on current month
    if current_month == 12:
        month_choices = [(12, 'December'), (1, 'January')]
    else:
        month_choices = [(current_month,
calendar.month_name[current_month]),
                         (current_month + 1,
calendar.month_name[current_month + 1])]

    # Define fields for month, day, and hour selection
    month = forms.ChoiceField(choices=month_choices, initial=current_month)
    day = forms.ChoiceField(choices=[(i, i) for i in range(1,
calendar.monthrange(current_year, current_month)[1] + 1)],
                            initial=current_day)
    hour = forms.ChoiceField(choices=[(i, i) for i in range(9, 21)],
                             initial=current_hour if 9 <= current_hour <= 20
else 9)
```

```
    def clean(self):
        cleaned_data = super().clean()
        year = self.current_year  # Use the current year
        month = cleaned_data.get('month')
        day = cleaned_data.get('day')
        hour = cleaned_data.get('hour')
        minute = 0  # Always set minutes to 00

        if month and day and hour:
            try:
                # Validate and create datetime object
                cleaned_data['appointment_datetime'] =
timezone.datetime(year, int(month), int(day), int(hour), minute,

tzinfo=timezone.get_current_timezone())
            except ValueError as e:
                raise forms.ValidationError("Invalid date or time:
{}".format(e))
        else:
            raise forms.ValidationError("Please fill in all date and time
fields.")

        return cleaned_data


# Form for submitting reviews
class ReviewForm(forms.ModelForm):
    class Meta:
        model = Review
        fields = ['rating', 'comment']


# management\commands\fill_customers.py
from django.core.management.base import BaseCommand
from django.contrib.auth.models import User
from booking.models import Customer

class Command(BaseCommand):
    help = 'Fills the Customer database with dummy data'

    def handle(self, *args, **options):
        # Number of customers to create
        num_customers = 5

        for i in range(num_customers):
            # Create a user for each customer
            username = f'user{i}'
            first_name = f'First{i}'
            last_name = f'Last{i}'
            email = f'user{i}@example.com'
            password = 'password'
            phone = f'7927333222{i}'
            birth_date = f'200{i}-01-0{i+1}'

            user = User.objects.create_user(
                username=username,
                password=password,
                email=email,
                first_name=first_name,
                last_name=last_name
            )
```

```
            # Create a customer linked to the user
            Customer.objects.create(
                user=user,
                birth_date=birth_date,
                phone=phone
            )

            self.stdout.write(self.style.SUCCESS(f'Successfully created
customer {username}'))


# templates\login.html
{% extends 'base.html' %}

{% block content %}
<div class="login-container">
    <h2>Login</h2><br>
    <form method="post" action="{% url 'login' %}">
        {% csrf_token %}
        <table>
            <tr>
                <td>Username:</td>
                <td><input type="text" name="username" id="username"
required></td>
            </tr>
            <tr>
                <td>Password:</td>
                <td><input type="password" name="password" id="password"
required></td>
            </tr>
            <tr>
                <td colspan="2"><button type="submit" class="login-
button">Login</button></td>
            </tr>
        </table>
        {% if error %}
            <p style="color: red;">{{ error }}</p>
        {% endif %}
    </form>
</div>
{% endblock %}


# views\booking.py
from django.http import JsonResponse
from django.utils import timezone
from django.contrib import messages
from django.contrib.auth.decorators import login_required, user_passes_test
from django.shortcuts import render, redirect, get_object_or_404
from . import BookingValidator
from ..forms import BookingDateTimeForm, ServiceProviderForm, ServiceForm
from ..models import Booking, Customer


@login_required
@user_passes_test(lambda u: u.is_staff)
def all_bookings(request):
    bookings = Booking.objects.all()
    return render(request, 'profile/all_bookings.html', {'bookings':
bookings})
```

```
@login_required
@user_passes_test(lambda u: u.is_staff)
def admin_booking_update(request, pk):
    booking = get_object_or_404(Booking, pk=pk)
    if request.method == 'POST':
        form = BookingDateTimeForm(request.POST)
        provider_form = ServiceProviderForm(request.POST)
        service_form = ServiceForm(request.POST)
        if form.is_valid() and provider_form.is_valid() and
service_form.is_valid():
            new_appointment_datetime =
form.cleaned_data['appointment_datetime']
            new_service_provider =
provider_form.cleaned_data['service_provider']
            new_service = service_form.cleaned_data['service']

            # Check if the selected time slot is available
            if BookingValidator.is_slot_available(new_service_provider.id,
new_appointment_datetime):
                booking.appointment_datetime = new_appointment_datetime
                booking.service_provider = new_service_provider
                booking.service = new_service
                booking.save()
                return redirect('all_bookings')
            else:
                messages.error(request, "This time slot is not available.
Please choose another time.")
    else:
        form = BookingDateTimeForm(initial={
            'month': booking.appointment_datetime.month,
            'day': booking.appointment_datetime.day,
            'hour': booking.appointment_datetime.hour,
        })
        provider_form = ServiceProviderForm(initial={'service_provider':
booking.service_provider})
        service_form = ServiceForm(initial={'service': booking.service})
    return render(request, 'profile/admin_booking_update.html', {
        'form': form,
        'provider_form': provider_form,
        'service_form': service_form
    })


@login_required
@user_passes_test(lambda u: u.is_staff)
def admin_booking_delete(request, pk):
    booking = get_object_or_404(Booking, pk=pk)
    if request.method == 'POST':
        booking.delete()
        return redirect('all_bookings')
    return render(request, 'profile/admin_booking_delete.html', {'booking':
booking})


@login_required
def list_booking(request):
    user = request.user
    now = timezone.now()
    try:
```

```
        # Attempt to retrieve the customer profile associated with the
current user
        customer_profile = Customer.objects.get(user=user)
        # Retrieve all bookings associated with the customer profile
        bookings = customer_profile.booking_set.all()
    except Customer.DoesNotExist:
        # If no customer profile exists, return an empty list of bookings
        bookings = []
    # Render the booking list page with the bookings and current time
    return render(request, 'profile/user_booking.html', {'bookings':
bookings, 'now': now})


@login_required
def user_booking_update(request, pk):
    # Retrieve the booking object or return a 404 error if not found
    booking = get_object_or_404(Booking, pk=pk)
    error_message = None

    if request.method == 'POST':
        form = BookingDateTimeForm(request.POST)
        if form.is_valid():
            # Extract the new appointment datetime from the form data
            new_appointment_datetime =
form.cleaned_data['appointment_datetime']
            # Check if the new appointment time slot is available
            if
BookingValidator.is_slot_available(booking.service_provider_id,
new_appointment_datetime):
                # Update the booking with the new appointment datetime and
save it
                booking.appointment_datetime = new_appointment_datetime
                booking.save()
                # Redirect to the booking list view after successful update
                return redirect('list_booking')
            else:
                # Set error message if the time slot is not available
                error_message = "This time slot is not available. Please
choose another time."
        # Render the update form again with the error message if form is not
valid or time slot not available
        return render(request, 'profile/user_booking_update.html', {'form':
form, 'error_message': error_message})
    else:
        # Pre-fill the form with the existing booking details
        form = BookingDateTimeForm(initial={
            'month': booking.appointment_datetime.month,
            'day': booking.appointment_datetime.day,
            'hour': booking.appointment_datetime.hour,
        })
    # Render the update form
    return render(request, 'profile/user_booking_update.html', {'form':
form, 'error_message': error_message})


# Ensure the user is authenticated before allowing access to the view
@login_required
def user_booking_delete(request, pk):
    if request.method == 'POST':
        # Retrieve the booking object or return a 404 error if not found
        booking = get_object_or_404(Booking, pk=pk)
```

```
        # Delete the booking
        booking.delete()
        # Return a success response
        return JsonResponse({'status': 'success'}, status=200)
    # Return an error response if not accessed via POST
    return JsonResponse({'status': 'error'}, status=400)


# management\commands\fill_reviews.py
from django.core.management.base import BaseCommand
from booking.models import Booking, Review, Service, Customer,
ServiceProvider
from django.utils import timezone


class Command(BaseCommand):
    help = 'Fills the Review database with structured data'

    def handle(self, *args, **options):
        customers = Customer.objects.all()
        services = list(Service.objects.all())
        service_providers = list(ServiceProvider.objects.all())

        if len(services) < 3 or len(service_providers) < 3:
            self.stdout.write(self.style.ERROR('Not enough services or
service providers to assign 3 each.'))
            return

        for customer in customers:
            for i in range(3):  # Create 3 bookings and reviews per customer
                service = services[i % len(services)]
                service_provider = service_providers[i %
len(service_providers)]

                # Create a booking
                booking = Booking.objects.create(
                    customer=customer,
                    service=service,
                    service_provider=service_provider,
                    appointment_datetime=timezone.now()  # Set the
appointment time to now for simplicity
                )

                # Create a review for the booking
                Review.objects.create(
                    booking=booking,
                    rating=i % 5 + 1,  # Simple pattern for rating
                    comment=f'Review {i + 1} by {customer.user.username} for
{service.name}'
                )

                self.stdout.write(self.style.SUCCESS(
                    f'Successfully created review {i + 1} by
{customer.user.username} for {service.name}'))


# static\js\profile\user_booking.js
function confirmDelete(e, element) {
e.preventDefault();
var bookingId = element.getAttribute('data-id');
if (confirm('Are you sure you want to delete this booking?')) {
```

```
    fetch(`/profile/booking/delete/${bookingId}/`, {
        method: 'POST',
        headers: {
            'X-CSRFToken': getCookie('csrftoken')
        }
    })
    .then(response => {
        if (response.ok) {
            return response.json();
        }
        throw new Error('Something went wrong');
    })
    .then(data => {
        alert('Booking deleted successfully');
        window.location.reload();
    })
    .catch(error => console.error('Error:', error));
}
}

function getCookie(name) {
    let cookieValue = null;
    if (document.cookie && document.cookie !== '') {
        const cookies = document.cookie.split(';');
        for (let i = 0; i < cookies.length; i++) {
            const cookie = cookies[i].trim();
            if (cookie.substring(0, name.length + 1) === (name + '=')) {
                cookieValue =
decodeURIComponent(cookie.substring(name.length + 1));
                break;
            }
        }
    }
    return cookieValue;
}

# templates\main.html
{% extends 'base.html' %}

{% block title %}
Welcome to Beauty Saloon
{% endblock %}

{% block content %}
<div class="header-content">
    <h1>Welcome to Beauty Saloon</h1>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
eiusmod tempor incididunt ut labore et dolore
        magna aliqua. Vitae suscipit tellus mauris a. Libero nunc consequat
interdum varius sit amet mattis. Lectus sit
        amet est placerat in egestas. Interdum consectetur libero id
faucibus nisl. Nulla malesuada pellentesque elit
        eget gravida cum sociis natoque. Ultricies mi eget mauris pharetra
et ultrices neque ornare aenean. Aliquet nibh
        praesent tristique magna sit amet purus gravida. Pellentesque id
nibh tortor id aliquet lectus proin nibh.
        Suspendisse ultrices gravida dictum fusce ut. Sed lectus vestibulum
mattis ullamcorper velit sed ullamcorper
        morbi tincidunt. Feugiat pretium nibh ipsum consequat nisl vel.</p>
</div>
{% endblock %}
```

```
# urls.py
from django.urls import path
from .views.auth import signup, user_login, user_logout
from .views.profile import manager_profile,user_profile, update_profile,
change_user_password, deactivate_user_profile
from .views.booking import all_bookings, list_booking, user_booking_update,
user_booking_delete, admin_booking_update, \
    admin_booking_delete
from .views.review import add_review, service_reviews
from .views.service import services
from .views import main_page, about_us, temp_list_all_db

urlpatterns = [
    path('', main_page, name='main'),
    path('about-us/', about_us, name='about_us'),
    path('signup/', signup, name='signup'),
    path('login/', user_login, name='login'),
    path('logout/', user_logout, name='logout'),

    path('services/', services, name='services'),

    path('manager/profile/', manager_profile, name='manager_profile'),
    path('manager/all_bookings/', all_bookings, name='all_bookings'),
    path('manager/booking/update/<int:pk>/', admin_booking_update,
name='admin_booking_update'),
    path('manager/booking/delete/<int:pk>/', admin_booking_delete,
name='admin_booking_delete'),

    path('profile/', user_profile, name='user_profile'),
    path('profile/update_profile/', update_profile, name='update_profile'),
    path('profile/change_password/', change_user_password,
name='change_password'),
    path('profile/deactivate_profile/', deactivate_user_profile,
name='deactivate_profile'),

    path('profile/list_booking', list_booking, name='list_booking'),
    path('profile/booking/update/<int:pk>/', user_booking_update,
name='user_booking_update'),
    path('profile/booking/delete/<int:pk>/', user_booking_delete,
name='user_booking_delete'),

    path('review/add/<int:booking_id>/', add_review, name='add_review'),
    path('reviews/<int:service_id>/', service_reviews,
name='service_reviews'),

    path('list/', temp_list_all_db, name='list'),
]



# management\commands\fill_managers.py
from django.core.management.base import BaseCommand
from django.contrib.auth.models import User
from booking.models import Manager


class Command(BaseCommand):
    help = 'Fills the Manager database with dummy data'
```

```
    def handle(self, *args, **options):
        # Number of managers to create
        num_managers = 5  # You can change this number as needed

        for i in range(num_managers):
            # Create a user for each manager
            username = f'manager{i}'
            first_name = f'ManagerFirstName'
            last_name = f'ManagerLastName'
            email = f'manager{i}@example.com'
            password = 'password'  # You might want to generate a more
secure password
            phone = f'7927333333{i}'
            birth_date = f'198{i}-01-0{i + 1}'

            user = User.objects.create_user(
                username=username,
                password=password,
                email=email,
                first_name=first_name,
                last_name=last_name
            )
            user.is_staff = True
            user.save()

            # Create a manager linked to the user
            Manager.objects.create(
                user=user,
                birth_date=birth_date,
                phone=phone
            )

            self.stdout.write(self.style.SUCCESS(f'Successfully created
manager {username}'))


# templates\profile\user_review.html
{% extends 'base.html' %}

{% block content %}
<div class="reviews-container">
    <div class="review-form">
        <h1>Leave review</h1>
        <form method="post" action="{% url 'add_review' booking_id %}">
            {% csrf_token %}
            <span class="label-rating">Rating:</span> <br><br>{{ form.rating
}}<br><br><br>
            <span class="label-comment">Write down your
comment:</span><br><br>{{ form.comment }}<br><br><br>
            <button type="submit" class="form-control">Submit
Review</button>
        </form>
    </div>
</div>
{% endblock %}


# templates\signup.html
{% extends 'base.html' %}

{% block title %}Signup - Beauty Saloon{% endblock %}
```

```
{% block content %}
<div class="signup-container">
    <h2>Registration</h2><br>
    <form method="post">
        {% csrf_token %}
        <table>
            <tr>
                <td>First Name:</td>
                <td>{{ form.first_name }}</td>
            </tr>
            <tr>
                <td>Last Name:</td>
                <td>{{ form.last_name }}</td>
            </tr>
            <tr>
                <td>Username:</td>
                <td>{{ form.username }}</td>
            </tr>
            <tr>
                <td>Email:</td>
                <td>{{ form.email }}</td>
            </tr>
            <tr>
                <td>Phone Number:</td>
                <td>{{ form.phone }}</td>
            </tr>
            <tr>
                <td>Birth Date:</td>
                <td>{{ form.birth_date }}</td>
            </tr>
            <tr>
                <td>Password:</td>
                <td>{{ form.password1 }}</td>
            </tr>
            <tr>
                <td>Confirm Password:</td>
                <td>{{ form.password2 }}</td>
            </tr>
        </table>
        {% if form.errors %}
            <div class="form-errors" style="color: red;">
                Please correct the errors below.
            </div>
        {% endif %}
            <button type="submit" class="register-button">Register</button>
        </div>
    </form>
</div>
<script>
document.addEventListener('DOMContentLoaded', function() {
    var birthDateInput = document.getElementById('id_birth_date');

    birthDateInput.addEventListener('keyup', function() {
        var value = this.value.replace(/[^0-9]/g, ''); // Удаляем все
нечисловые символы
        if (value.length >= 2 && value.length <= 3) {
            this.value = value.slice(0, 2) + '.' + value.slice(2);
        } else if (value.length > 4) {
            this.value = value.slice(0, 2) + '.' + value.slice(2, 4) + '.' +
value.slice(4);
```

```
        }
    });
});

</script>
{% endblock %}

# templates\reviews.html
{% extends 'base.html' %}

{% block content %}
<div class="reviews-container">
    <h1>Reviews for {{ service.name }}</h1>
    <table>
        {% for review in reviews %}
        <tr>
            <th>Rating</th>
            <th><strong>Comment by: {{ review.booking.customer.user.username
}}</strong><br></th>
        </tr>

        <tr>
            <td>{{ review.rating }}</td>
            <td>

                Review: {{ review.comment }}<br><br>
                Review created at: <small>{{ review.created_at|date:"Y-m-d"
}}</small>
            </td>
        </tr>
        {% empty %}
            <h3>No reviews available.</h3>
        {% endfor %}
    </table>
</div>
{% endblock %}


# templates\profile\manager_profile.html
<!-- templates/profile/manager_profile.html -->
{% extends 'base.html' %}

{% block content %}
<div class="manager-profile-container">
    <h1>Manager Profile</h1>
    <div class="profile-section">
        <div class="profile-row">
            <div class="profile-column">
                <span>Email: </span>
                <span>{{ manager.user.email }}</span>
            </div>
        </div>
        <div class="profile-row">
            <div class="profile-column">
                <span>Phone: </span>
                <span>{{ manager.phone }}</span>
            </div>
        </div>
        <div class="profile-button-row">
            <a href="{% url 'all_bookings' %}" class="link-as-button">View
All Bookings</a>
```

```
        </div>
    </div>
</div>
{% endblock %}


# management\commands\fill_manytomany.py
from django.core.management.base import BaseCommand
from booking.models import ServiceProvider, Service
import random

class Command(BaseCommand):
    help = 'Populates the many-to-many relationship between Service and
ServiceProvider'

    def handle(self, *args, **options):
        # Fetch all service providers and services
        service_providers = list(ServiceProvider.objects.all())
        services = list(Service.objects.all())

        if not service_providers or not services:
            self.stdout.write(self.style.ERROR('No service providers or
services found. Please add some first.'))
            return

        # Optionally clear existing relationships
        for provider in service_providers:
            provider.service.clear()

        # Randomly assign services to each service provider
        for provider in service_providers:
            # Randomly choose a number of services to assign
            assigned_services = random.sample(services, k=random.randint(1,
len(services)))
            provider.service.add(*assigned_services)
            provider.save()  # Save the changes

            self.stdout.write(self.style.SUCCESS(f'Assigned
{len(assigned_services)} services to {provider.user.username}'))



# models.py
from django.db import models
from django.contrib.auth.models import User
from django.utils import timezone

"""
The module contains models for the basic user profile, administrator,
service provider, customer, service, booking and review in the service
management system.
"""


class BaseProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    phone = models.CharField(max_length=20)
    birth_date = models.DateField(null=True, blank=True)

    class Meta:
        abstract = True
```

```
    def __str__(self):
        return self.user.username


class Administrator(BaseProfile):
    pass


class Manager(BaseProfile):
    pass


class ServiceProvider(BaseProfile):
    specialization = models.CharField(max_length=100)
    service = models.ManyToManyField('Service')

    def __str__(self):
        return f"Поставщик услуг: {self.user.username}, Специализация:
{self.specialization}, Телефон: {self.phone}"


class Customer(BaseProfile):
    pass


class Service(models.Model):
    name = models.CharField(max_length=100)
    description = models.TextField()
    duration = models.DurationField()
    price = models.DecimalField(max_digits=6, decimal_places=2)

    def __str__(self):
        return self.name


class Booking(models.Model):
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
    service = models.ForeignKey(Service, on_delete=models.CASCADE)
    service_provider = models.ForeignKey(ServiceProvider,
on_delete=models.CASCADE)
    appointment_datetime = models.DateTimeField(
        default=timezone.now)  # TODO удалить default при релизе, нужен для
заполнения бд командой
    created_at = models.DateTimeField(default=timezone.now)

    def __str__(self):
        return f"{self.customer.user.username} booked {self.service.name}
with {self.service_provider.user.username} at {self.appointment_datetime}.
Booking created at {self.created_at}"


class Review(models.Model):
    booking = models.ForeignKey(Booking, on_delete=models.CASCADE)
    rating = models.IntegerField(default=1, choices=[(i, i) for i in
range(1, 6)])
    comment = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Review by {self.booking.customer.user.username} for
```

```
{self.booking.service.name} provided by
{self.booking.service_provider.user.username}"


# static\js\profile\user_profile.js
function getCSRFToken() {
    let cookies = document.cookie.split(';');
    for (let cookie of cookies) {
        let [name, value] = cookie.trim().split('=');
        if (name === 'csrftoken') {
            return decodeURIComponent(value);
        }
    }
    return null;
}

function toggleEdit(field) {
    let span = document.getElementById(field + '-value');
    let input = document.getElementById(field + '-input');
    let editButton =
document.querySelector(`button[onclick="toggleEdit('${field}')"]`);
    let saveButton =
document.querySelector(`button[onclick="save('${field}')"]`);

    if (input.style.display === 'none') {
        span.style.display = 'none';
        input.style.display = 'inline';
        editButton.style.display = 'none';
        saveButton.style.display = 'inline';
    } else {
        span.style.display = 'inline';
        input.style.display = 'none';
        editButton.style.display = 'inline';
        saveButton.style.display = 'none';
    }
}

function save(field) {
    let input = document.getElementById(field + '-input');
    let span = document.getElementById(field + '-value');
    let value = input.value.trim();
    let csrfToken = getCSRFToken();
    let updateProfileUrl = document.getElementById('data-
container').getAttribute('data-update-profile-url');

    fetch(updateProfileUrl, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'X-CSRFToken': csrfToken
        },
        body: JSON.stringify({field: field, value: value})
    }).then(response => response.json())
    .then(data => {
        if (data.success) {
            span.textContent = value;
            toggleEdit(field);
            alert('Данные успешно сохранены');
        } else {
            alert('Ошибка при сохранении данных');
        }
```

```
    }).catch(error => {
        console.error('Error:', error);
        alert('Произошла ошибка при обработке вашего запроса.');
    });
}

function togglePasswordChange() {
    let container = document.getElementById('password-change');
    container.style.display = container.style.display === 'none' ? 'block' :
'none';
}

function changePassword() {
    let oldPassword = document.getElementById('old-password').value.trim();
    let newPassword = document.getElementById('new-password').value.trim();
    let confirmPassword = document.getElementById('confirm-
password').value.trim();
    let csrfToken = getCSRFToken();
    let changePasswordUrl = document.getElementById('data-
container').getAttribute('data-change-password-url');

    if (!newPassword || !confirmPassword || !oldPassword) {
        alert('Пароль не может быть пустым');
        return;
    }

    if (newPassword !== confirmPassword) {
        alert('Пароли не совпадают');
        return;
    }

    fetch(changePasswordUrl, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'X-CSRFToken': csrfToken
        },
        body: JSON.stringify({old_password: oldPassword, new_password1:
newPassword, new_password2: confirmPassword})
    }).then(response => response.json())
    .then(data => {
        if (data.success) {
            alert('Пароль успешно изменен');
            togglePasswordChange();
        } else {
            if (data.errors && data.errors.new_password2) {
                alert('Ошибка при изменении пароля: ' +
data.errors.new_password2.join('; '));
            } else {
                alert('Ошибка при изменении пароля. Пожалуйста, проверьте
введённые данные.');
            }
        }
    }).catch(error => {
        console.error('Error:', error);
        alert('Произошла ошибка при обработке вашего запроса.');
    });
}

function deactivateProfile() {
    let csrfToken = getCSRFToken();
```

```
    let deactivateProfileUrl = document.getElementById('data-
container').getAttribute('data-deactivate-profile-url');
    if (window.confirm('Are you sure you want to delete your profile?')) {
        fetch(deactivateProfileUrl, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'X-CSRFToken': csrfToken
            }
        }).then(response => {
            if (response.ok) {
                window.location.href = "/logout/";
            } else {
                alert('Ошибка при удалении профиля');
            }
        }).catch(error => {
            console.error('Error:', error);
            alert('Произошла ошибка при обработке вашего запроса.');
        });
    }
}


# templates\profile\admin_booking_delete.html
{% extends 'base.html' %}

{% block content %}
<div class="container">
    <h1>Delete Booking</h1>
    <p>Are you sure you want to delete this booking?</p>
    <p>Service: {{ booking.service.name }}</p>
    <p>Customer: {{ booking.customer.user.username }}</p>
    <p>Service Provider: {{ booking.service_provider.user.username }}</p>
    <p>Appointment Date and Time: {{ booking.appointment_datetime|date:"Y-m-
d H:i" }}</p>
    <form method="post">
        {% csrf_token %}
        <button type="submit" class="btn btn-danger">Delete</button>
        <a href="{% url 'all_bookings' %}" class="btn btn-
secondary">Cancel</a>
    </form>
</div>
{% endblock %}


# templates\services.html
{% extends 'base.html' %}

{% block content %}
<div style="display: flex; justify-content: center;">
    <table class="services">
        {% for service, provider_form, date_time_form, error_message,
current_data in forms_list %}
        <tr>
            <td>
                <h2>{{ service.name }}</h2>
                <p>{{ service.description }}</p>
                <p>Duration: {{ service.duration }}</p>
                <p>Price: {{ service.price }}</p>
                <a href="{% url 'service_reviews' service.id %}"
class="link-as-button">See all reviews</a>
```

```
            </td>
            <td>
                <form method="post" action="">
                    {% csrf_token %}
                    <input type="hidden" name="service_id" value="{{
service.id }}">
                    {{ provider_form.as_p }}
                    {{ date_time_form.as_p }}
                    {% if error_message %}
                    <p style="color: red;">{{ error_message }}</p>
                    {% endif %}
                    <button type="submit">Book</button>
                </form>
            </td>
        </tr>
        {% endfor %}
    </table>
</div>
{% endblock %}


# views\profile.py
from django.shortcuts import render
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt
from django.contrib.auth.decorators import login_required
from django.contrib.auth import update_session_auth_hash
from ..models import Manager, ServiceProvider, Customer
from ..forms import CustomPasswordChangeForm
import json


# View for displaying manager profile
@login_required
def manager_profile(request):
    manager = Manager.objects.get(user=request.user)
    return render(request, 'profile/manager_profile.html', {'manager':
manager})


# View for displaying user profile
@login_required
def user_profile(request):
    customer = Customer.objects.get(user=request.user)
    password_form = CustomPasswordChangeForm(request.user)
    return render(request, 'profile/user_profile.html', {
        'password_form': password_form
    })


# View for updating user profile fields
@login_required
@csrf_exempt
def update_profile(request):
    if request.method == 'POST':
        data = json.loads(request.body)
        field = data.get('field')
        value = data.get('value')
        customer = Customer.objects.get(user=request.user)

        if field == 'email':
```

```
            customer.user.email = value
            customer.user.save()
        elif field == 'phone':
            customer.phone = value
            customer.save()

        return JsonResponse({'success': True})
    return JsonResponse({'success': False})


# View for changing user password
@login_required
@csrf_exempt
def change_user_password(request):
    if request.method == 'POST':
        data = json.loads(request.body)
        form = CustomPasswordChangeForm(user=request.user, data=data)
        if form.is_valid():
            user = form.save()
            update_session_auth_hash(request, user)  # Update session to
prevent logout
            return JsonResponse({'success': True})
        else:
            return JsonResponse({'success': False, 'errors': form.errors})


# View for deactivating a user profile
@login_required
@csrf_exempt
def deactivate_user_profile(request):
    if request.method == 'POST':
        customer = Customer.objects.get(user=request.user)
        customer.user.is_active = False
        customer.user.save()
        return JsonResponse({'success': True})
    return JsonResponse({'success': False})


# templates\temp_list_all_db.html
<!-- templates/customers_list.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>List of All Users</title>
</head>
<body>
<h1>List of All Users</h1>

<h2>Customers</h2>
<ul>
    {% for customer in customers %}
    <li>{{ customer.user.username }} - {{ customer.phone }}</li>
    {% empty %}
    <li>No customers found.</li>
    {% endfor %}
</ul>

<h2>Administrators</h2>
<ul>
    {% for admin in administrators %}
```

```
    <li>{{ admin.user.username }} - {{ admin.phone }}</li>
    {% empty %}
    <li>No administrators found.</li>
    {% endfor %}
</ul>

<h2>Service Providers</h2>
<ul>
    {% for provider in service_providers %}
    <li>{{ provider.user.username }} - {{ provider.specialization }} - {{
provider.phone }}</li>
    {% empty %}
    <li>No service providers found.</li>
    {% endfor %}
</ul>

<h2>Services</h2>
<ul>
    {% for service in services %}
    <li>{{ service.name }} - {{ service.description }} - Duration: {{
service.duration }} - Price: ${{ service.price }}</li>
    {% empty %}
    <li>No services found.</li>
    {% endfor %}
</ul>

<h2>Reviews</h2>
<ul>
    {% for review in reviews %}
    <li>Review by {{ review.customer.user.username }} for {{
review.service.name }} provided by {{ review.service_provider.user.username
}} - Rating: {{ review.rating }} - Comment: {{ review.comment }}</li>
    {% empty %}
    <li>No reviews found.</li>
    {% endfor %}
</ul>

<h2>Bookings</h2>
<ul>
    {% for booking in bookings %}
    <li>{{ booking.customer.user.username }} booked {{ booking.service.name
}} with {{ booking.service_provider.user.username }} at {{
booking.appointment_time }}. Booking create at {{ booking.created_at
}}</li>
    {% empty %}
    <li>No bookings found.</li>
    {% endfor %}
</ul>

</body>
</html>


# management\commands\fill_service_providers.py
from django.core.management.base import BaseCommand
from django.contrib.auth.models import User
from booking.models import ServiceProvider, Service
import random

class Command(BaseCommand):
    help = 'Fills the ServiceProvider database with dummy data and assigns
```

```
all services to them'

    def handle(self, *args, **options):
        # Number of service providers to create
        num_service_providers = 6  # You can change this number as needed

        # Fetch all services from the database
        all_services = list(Service.objects.all())

        for i in range(num_service_providers):
            # Create a user for each service provider
            username = f'master{i}'
            first_name = f'FirstName{i}'
            last_name = f'LastName{i}'
            email = f'master{i}@example.com'
            password = 'password'  # You might want to generate a more
secure password
            phone = f'7927333221{i}'
            specialization = f'Specialization{i}'
            birth_date = f'200{i}-01-0{i+1}'

            user = User.objects.create_user(
                username=username,
                password=password,
                email=email,
                first_name=first_name,
                last_name=last_name
            )

            # Create a service provider linked to the user
            service_provider = ServiceProvider.objects.create(
                user=user,
                birth_date=birth_date,
                phone=phone,
                specialization=specialization
            )

            # Assign all services to the service provider
            service_provider.service.add(*all_services)

            self.stdout.write(self.style.SUCCESS(f'Successfully created
service provider {username} with all services assigned'))


# views\review.py
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth.decorators import login_required
from ..models import Booking, Review, Service
from ..forms import ReviewForm


@login_required  # Ensures that only authenticated users can access this
view
def add_review(request, booking_id):
    # Retrieve the booking object, or return a 404 error if not found
    booking = get_object_or_404(Booking, id=booking_id)
    try:
        # Try to retrieve an existing review for the booking
        review = Review.objects.get(booking=booking)
        form = ReviewForm(request.POST or None, instance=review)
        update = True  # Flag to indicate that this is an update operation
```

```
    except Review.DoesNotExist:
        # If no review exists, create a new review instance
        review = Review(booking=booking)
        form = ReviewForm(request.POST or None, instance=review)
        update = False  # Flag to indicate that this is a create operation

    if request.method == 'POST':
        if form.is_valid():
            # Save the form data to the review object
            review = form.save(commit=False)
            if not update:
                # Set additional fields if this is a new review
                review.customer = booking.customer
                review.service = booking.service
                review.service_provider = booking.service_provider
            review.save()  # Save the review to the database
            return redirect('list_booking')  # Redirect to the booking list
after saving
        else:
            # If the form is not valid, render the page again with the form
            return render(request, 'profile/user_review.html', {'form':
form, 'booking_id': booking_id})
    else:
        # If not a POST request, render the page with the form
        return render(request, 'profile/user_review.html', {'form': form,
'booking_id': booking_id})


def service_reviews(request, service_id):
    # Retrieve the service object, or return a 404 error if not found
    service = get_object_or_404(Service, id=service_id)
    # Retrieve all reviews related to the service
    reviews =
Review.objects.filter(booking__service=service).select_related('booking',
'booking__customer')
    # Render the reviews page with the service and its reviews
    return render(request, 'reviews.html', {'service': service, 'reviews':
reviews})


# templates\base.html
<!DOCTYPE html>
{% load static %}
<html lang="en">
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="{% static 'css/style.css' %}">
    <title>{% block title %}Beauty Saloon{% endblock %}</title>
</head>
<body>
<header>
    <nav class="nav-container">
        <ul class="navlist">
            <li><a href="{% url 'about_us' %}" class="nav-link">ABOUT
US</a></li>
            <li><a href="{% url 'services' %}" class="nav-
link">SERVICES</a></li>
            <li><a href="{% url 'main' %}" class="nav-link"><img src="{%
static 'images/main_logo.png' %}"alt="Logo"></a></li>
            {% if request.user.is_authenticated %}
            {% if request.user.is_staff %}
```

```
            <li><a href="{% url 'manager_profile' %}" class="nav-
link">MANAGER PROFILE</a></li>
            {% else %}
            <li><a href="{% url 'user_profile' %}" class="nav-
link">PROFILE</a></li>
            {% endif %}
            <li><a href="{% url 'logout' %}" class="nav-
link">LOGOUT</a></li>
            {% else %}
            <li><a href="{% url 'signup' %}" class="nav-
link">REGISTER</a></li>
            <li><a href="{% url 'login' %}" class="nav-link">SIGN
IN</a></li>
            {% endif %}
        </ul>
    </nav>
</header>
<main>
    {% block content %}
    {% endblock %}
</main>
<footer>
    <p>Â© 2024 Beauty Saloon. All rights reserved.</p>
    {% block footer %}
    {% endblock %}
</footer>
{% block javascript %}
{% endblock %}
</body>
</html>


# management\commands\fill_bookings.py
from django.core.management.base import BaseCommand
from booking.models import Booking, Service, Customer, ServiceProvider
from django.utils import timezone
import random

class Command(BaseCommand):
    help = 'Fills the Booking database with structured data'

    def handle(self, *args, **options):
        customers = Customer.objects.all()
        services = list(Service.objects.all())
        service_providers = list(ServiceProvider.objects.all())

        if len(services) < 3 or len(service_providers) < 3:
            self.stdout.write(self.style.ERROR('Not enough services or
service providers to assign 3 each.'))
            return

        for customer in customers:
            for i in range(3):  # Create 3 bookings per customer
                service = services[i % len(services)]
                service_provider = service_providers[i %
len(service_providers)]
                days_ahead = i * 7  # Bookings spaced one week apart
                appointment_datetime = timezone.now() +
timezone.timedelta(days=days_ahead)

                # Randomize the creation date within the past year
```

```
                days_back = random.randint(0, 365)  # Random number of days
up to a year back
                created_at = timezone.now() -
timezone.timedelta(days=days_back)  # Subtract days to go back in time

                Booking.objects.create(
                    customer=customer,
                    service=service,
                    service_provider=service_provider,
                    appointment_datetime=appointment_datetime,
                    created_at=created_at  # Added the created_at field with
a random date within the last year
                )
                self.stdout.write(self.style.SUCCESS(f'Successfully created
booking {i+1} for {customer.user.username} with {service.name} at
{appointment_datetime}, created on {created_at}'))


# templates\profile\user_booking.html
{% extends 'base.html' %}
{% load static %}
{% block content %}
<div class="order-history">
    <h1 class="order-history__header">Booking history</h1>
    <div class="order-history__table-container">
        <table class="order-history__table">
            <thead>
            <tr>
                <th class="order-history__column-header">Service</th>
                <th class="order-history__column-header">Master</th>
                <th class="order-history__column-header">Date and time of
service</th>
                <th class="order-history__column-header">Actions</th>
            </tr>
            </thead>
            <tbody>
            {% for booking in bookings %}
            <tr class="order-history__row">
                <td class="order-history__cell">{{ booking.service.name
}}</td>
                <td class="order-history__cell">{{
booking.service_provider.user.username }}</td>
                <td class="order-history__cell">{{
booking.appointment_datetime|date:"Y-m-d H:i" }}</td>
                <td class="order-history__cell">
                    <div class="order-history__actions-container">
                        {% if booking.appointment_datetime > now %}
                        <a href="{% url 'user_booking_update' booking.id %}"
                           class="order-history__link order-history__link--
edit">Change booking</a>
                        <a href="#" class="order-history__link order-
history__link--delete" data-id="{{ booking.id }}"
                           onclick="confirmDelete(event, this)">Cancel
booking</a>
                        {% else %}
                        {% if booking.review_set.exists %}
                        <a href="{% url 'add_review' booking.id %}"
class="order-history__link order-history__link--review">Change review</a>
                        {% else %}
                        <a href="{% url 'add_review' booking.id %}"
class="order-history__link order-history__link--review">Leave review</a>
```

```
                        {% endif %}
                        {% endif %}
                    </div>
                </td>
            </tr>
            {% empty %}
            <tr class="order-history__row">
                <td colspan="4" class="order-history__cell">No
bookings.</td>
            </tr>
            {% endfor %}
            </tbody>
        </table>
    </div>
</div>
<script src="../static/js/profile/user_booking.js"></script>
{% endblock %}


# templates\about_us.html
{% extends 'base.html' %}

{% block title %}About Us - Beauty Saloon{% endblock %}

{% block content %}
<div class="header-content">
    <h1>About Us</h1>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
eiusmod tempor incididunt ut labore et dolore
        magna aliqua. Vitae suscipit tellus mauris a. Libero nunc consequat
interdum varius sit amet mattis. Lectus sit
        amet est placerat in egestas. Interdum consectetur libero id
faucibus nisl. Nulla malesuada pellentesque elit
        eget gravida cum sociis natoque. Ultricies mi eget mauris pharetra
et ultrices neque ornare aenean. Aliquet nibh
        praesent tristique magna sit amet purus gravida. Pellentesque id
nibh tortor id aliquet lectus proin nibh.
        Suspendisse ultrices gravida dictum fusce ut. Sed lectus vestibulum
mattis ullamcorper velit sed ullamcorper
        morbi tincidunt. Feugiat pretium nibh ipsum consequat nisl vel.</p>
</div>
{% endblock %}


# management\commands\fill_administrators.py
from django.core.management.base import BaseCommand
from django.contrib.auth.models import User
from booking.models import Administrator


class Command(BaseCommand):
    help = 'Fills the Administrator database with dummy data'

    def handle(self, *args, **options):
        # Number of administrators to create
        num_administrators = 3

        for i in range(num_administrators):
            # Create a superuser for each administrator
            username = f'admin{i}'
            first_name = f'AdminFirst{i}'
```

```
            last_name = f'AdminLast{i}'
            email = f'admin{i}@example.com'
            password = 'password'
            phone = f'7927333111{i}'
            birth_date = f'200{i}-01-0{i + 1}'

            user = User.objects.create_superuser(
                username=username,
                password=password,
                email=email,
                first_name=first_name,
                last_name=last_name
            )
            user.is_staff = True  # Mark the user as staff
            user.is_superuser = True  # Mark the user as superuser
            user.save()

            Administrator.objects.create(
                user=user,
                birth_date=birth_date,
                phone=phone
            )
            # Create an administrator linked to the user

            self.stdout.write(self.style.SUCCESS(f'Successfully created
administrator {username}'))
```