МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ
ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ им. А. И. ГЕРЦЕНА»

Основная профессиональная образовательная программа
Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль) «Технологии разработки программного обеспечения»
форма обучения – очная

Выпускная квалификационная работа

Разработка веб-сервиса для управления бизнес-процессами салона красоты «BeautyManage»

Обучающейся 4 курса
Елкиной Галины Александровны

_____

Научный руководитель:
кандидат физ.-мат. наук, доцент
Жуков Николай Николаевич

_____

Санкт-Петербург
2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

При ведении бизнеса любого размера нужно уметь управлять тем, что есть в компании или предприятии: нужно понимать, что делают сотрудники и как организовать их работу, какие ресурсы есть у компании, а чего не хватает, какой образ клиента и что ему предложить, как управлять ценами на товары компании и многое другое. Сейчас для таких задач крупные компании и бизнес среднего уровня создают свои собственные системы управления бизнес-процессами на базе системы Bitrix или разрабатывают решение «под себя», а малый бизнес и ИП пользуются готовыми решениями на различных платформах, в том числе используя мобильные приложения различного рода. Но для осуществления управления даже такой маленькой на первый взгляд компанией требуется огромное множество предлагаемых приложений, чтобы учесть все аспекты управления бизнес-процессами, что абсолютно неудобно для управляющего.

С такой проблемой сталкиваются многие управляющие салоном красоты, поэтому они ищут более подходящие решения, которые будут соответствовать их нуждам и требованиям.

В настоящее время есть немало подобных систем для управления бизнес-процессами салона. Но многие из них предоставляют слишком много функций, которые малому бизнесу или ИП совершенно не нужны, а плата за них все равно берется. Также в таких системах не всегда есть нужная отчетность или необходимые функции для ИП, такие как напоминания об оплате счетов.

Задача разработки сервиса для управления бизнес-процессами на базе веб-технологий является **актуальной** в связи с получением заказа от ИП Логинова С.А.

Предметом исследования является веб-сервис для управления бизнес-процессами салона красоты «BeautyManage».

Практическая значимость исследования заключается в разработке веб-сервиса, который позволит управлять деятельностью сотрудников, потоком клиентов, сопутствующими процессу статьями расходов и доходов, а также дополнительными процессами, аккумулированными в одном сервисе. Практическая значимость разработки данного веб-сервиса усиливается тем, что договоренность с заказчиком позволяет расширить сервис так, чтобы его могли использовать также другие компании, не зависящие от заказчика.

Целью выпускной квалификационной работы стала разработка веб-сервиса с реализованными функциями для управления первостепенными для заказчика бизнес-процессами, описанными им, который бы соответствовал требованиям заказчика.

Для достижения цели работы были поставлены следующие **задачи**:

1. Проанализировать бизнес-процессы в области управления салоном красоты и требования заказчика, чтобы реализовать нужную структуру веб-сервиса на основе требуемых бизнес-процессов.
2. Проанализировать архитектурные решения, различные веб-технологии и требования заказчика, чтобы выбрать подходящие для реализации веб-сервиса.
3. Проанализировать серверные решения и их технологии, чтобы выбрать подходящие для передачи сервиса заказчику и дальнейших его успешных работе и сопровождению со стороны разработчика.
4. Спроектировать архитектуру и макеты страниц MVP-сервиса в соответствии с требуемыми заказчиком функциями и выбранными технологиями.
5. Реализовать MVP-веб-сервис по управлению бизнес-процессами салона красоты и предоставить доступ к сервису заказчику.

Структура бакалаврской выпускной квалификационной работы. Выпускная квалификационная работа состоит из введения, двух глав,

заключения, списка литературы и приложений. Она содержит 50 страниц, 20 изображений и 10 таблиц.

ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ ПРОЕКТА

Предметная область проекта разделяется на несколько подобластей:

- бизнес-процессы салона красоты;
- архитектура веб-сервиса;
- веб-технологии для реализации сервиса;
- серверные решения и технологии.

Рассмотрим данные предметные области отдельно.

1.1 Анализ бизнес-процессов салона красоты

Для обеспечения достижения поставленной цели необходимо было определить, какие бизнес-процессы будут реализовываться через веб-сервис, разработанный в ходе данной работы. Для этого нужно было понять, что такое бизнес-процесс, какие виды бизнес-процессов могут быть, и какие именно бизнес-процессы присутствуют в организации, для которой создавался веб-сервис.

Определений бизнес-процесса существует несколько десятков. В различных источниках они отличаются. Приведем обобщенное определение, чтобы понимать саму суть термина. Бизнес-процесс – это организованная деятельность, которая, имея «на входе» определенные ресурсы, дает «на выходе» результат, представляющий ценность для потребителя или достигающий какую-либо цель бизнеса.

Бизнес-процессы разделяют на несколько категорий:

- основные – ориентированные на получение дохода;
- сопутствующие – вытекающие из основных бизнес-процессов и также ориентированные на получение дохода;

- вспомогательные – поддерживающие функционирование основных и сопутствующих бизнес-процессов;
- обеспечивающие – поддерживающие функционирование бизнеса в целом (финансовое обеспечение, кадровое обеспечение, техническое обеспечение и т.д.);
- управления – обеспечивающие долгосрочное и краткосрочное планирование и осуществляющие воздействия на любые процессы бизнеса;
- развития – совершенствующие как продукт бизнеса, так и нужные бизнес-процессы.

Для определения того, какие бизнес-процессы есть в салоне красоты, были проведены анализ предоставленного заказчиком технического задания, полный текст которого представлен в приложении А, и беседа с самим заказчиком. В результате были выявлены и классифицированы несколько бизнес-процессов, которые будут осуществляться с применением разрабатываемого веб-сервиса. Также выявленные бизнес-процессы были проанализированы на предмет того, кто будет задействован в каждом из них. Это было необходимо, чтобы определить задействованные роли – администратор, сотрудник, клиент, – которые влияют на работу внутри сервиса. Результаты анализа технического задания и беседы с заказчиком, которые представляют собой классифицированные бизнес-процессы салона красоты, представлены в таблице 1.

Таблица 1 – Бизнес-процессы салона красоты

Категория бизнес-процесса	**Бизнес-процесс салона красоты**	**Роли, задействованные в процессе**
Основные	Отправление заявки на предоставление услуги	Клиент, Сотрудник, Администратор
	Обработка заявки клиента	Администратор

	Регистрация (завершение) проведения услуги	Сотрудник
Вспомогательные	Процесс списания материала	Администратор, Сотрудник
	Процесс закупки материалов	Администратор
	Процесс фиксации количества материалов	Администратор, Сотрудник
	Проведение оплаты за материалы/другие услуги (составление записи)	Администратор
	Отправление заявки на смену	Администратор, Сотрудник
	Обработка заявки на смену	Администратор, Сотрудник
	Регистрация клиента	Администратор, Сотрудник, Клиент
Обеспечивающие	Проведение оплаты за услугу (составление записи)	Администратор, Сотрудник
	Процесс приема на работу	Администратор
Управления	Составление отчета по закупкам	Администратор
	Процесс открытия смены	Сотрудник
	Процесс закрытия смены	Сотрудник
	Управление списком услуг, в том числе ценообразование	Администратор

Для более детального понимания каждого бизнес-процесса необходимо для каждого из них составить диаграммы последовательности. Но в рамках выпускной квалификационной работы была реализована только первая версия

сервиса, которая требовала реализации не всех перечисленных бизнес-процессов. По договоренности с заказчиком необходимо было реализовать в первую очередь основные бизнес-процессы и некоторые бизнес-процессы из других категорий. На рисунке 1 приведена диаграмма последовательности для бизнес-процесса «Отправление заявки на предоставление услуги». Остальные необходимые для реализации проекта диаграммы последовательностей находятся в приложении Б.

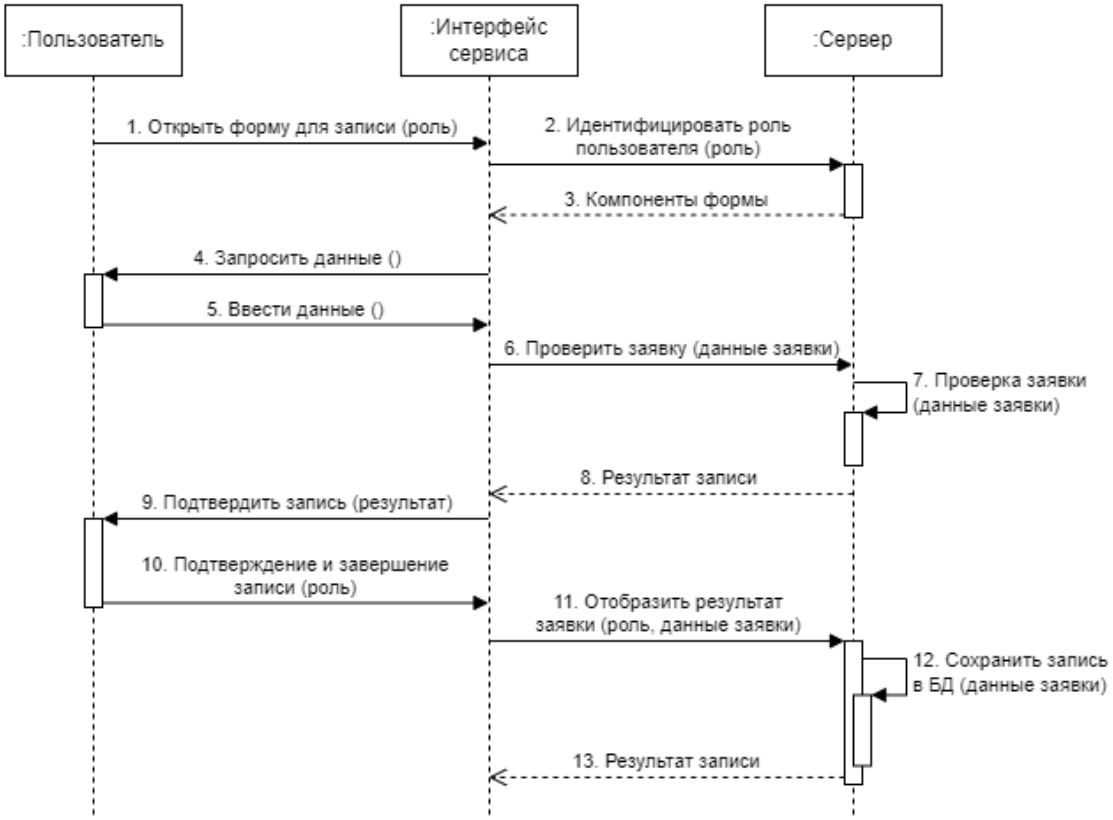


Рисунок 1 – Диаграмма последовательности для бизнес-процесса «Отправление заявки на предоставление услуги»

После понимания того, какие бизнес-процессы есть в салоне красоты и как именно они должны проводиться через разрабатываемый сервис, были проанализированы существующие решения, чтобы убедиться в актуальности разработки нового сервиса.

1.2 Анализ существующих решений

Для управления вышеописанными бизнес-процессами существует не мало CRM-систем. Необходимо было проанализировать, есть ли все необходимые заказчику функции в таких системах.

Для анализа были выбраны наиболее популярные CRM-системы [29]. Они были проанализированы по критериям, составленным на основе рассмотренных выше бизнес-процессов, а также на основе технического задания, так как выше были описаны не все требуемые заказчиком функции. В критерии были включены также различные возможности работы с клиентами, которые являются важной составляющей работы салона красоты.

Результаты анализа представлены в таблице 2.

Таблица 2 – Анализ существующих решений

Критерий	**EasyWeek**	**ПрофиГид**	**СБИС**
Основные бизнес-процессы салона красоты (из таблицы 1)	3/3	3/3	3/3
Вспомогательные бизнес-процессы салона красоты (из таблицы 1)	6/7	3/7	6/7
Обеспечивающие бизнес-процессы салона красоты (из таблицы 1)	2/2	2/2	2/2
Бизнес-процессы управления салона красоты (из таблицы 1)	1/4	1/4	4/4

Возможность иметь брендированный сайт / мобильное приложение для клиентов	Есть	Есть	Есть
Уведомления о необходимости оплаты счетов для администратора	Нет	Нет	Нет
Уведомления о необходимости закупки материалов	Нет	Нет	Нет
Система оценки персонала	Есть	Нет	Есть
Черный список клиентов	Нет	Нет	Нет

Из результатов анализа видно, что нет такой системы, которая удовлетворяла бы всем потребностям заказчика. Поэтому разработка нового сервиса актуальна и целесообразна.

1.3 Анализ архитектурных решений веб-сервиса

В основе любого веб-сервиса всегда лежат одни и те же компоненты архитектуры: веб-клиент, веб-сервер и база данных.

- Веб-клиент – это приложение, через которое пользователь получает доступ к сервису и может им пользоваться.
- Веб-сервер – это программа, ориентированная на получение запросов от веб-клиента и обработку этих запросов посредством HTTP-протокола.
- База данных – набор структурированных данных, построенных по определенным принципам. Обычно базы данных управляются с помощью системы управления базами данных (СУБД), которая

представляет собой набор инструментов, позволяющих контролировать таблицы и записи баз данных.

На рисунке 2 представлена схема взаимодействия основных компонент веб-сервиса [2]:

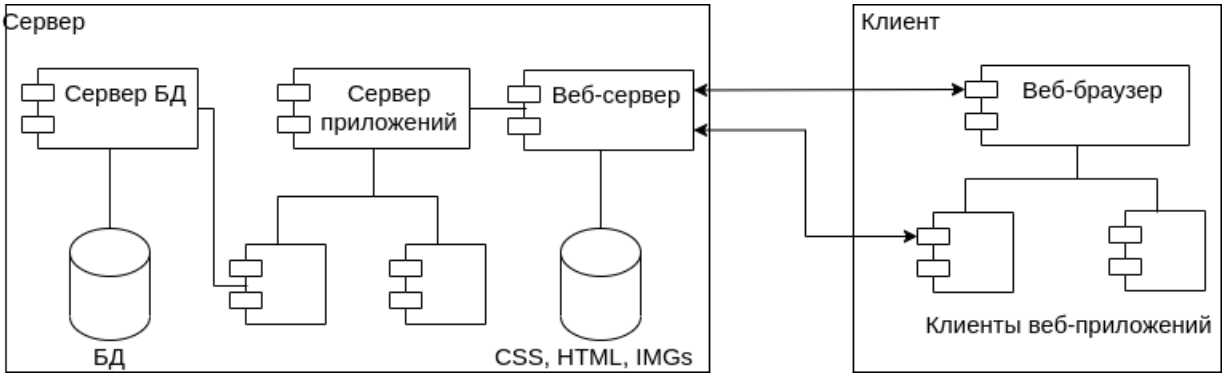


Рисунок 2 - Концептуальная схема технологий и компонент при взаимодействии сервера и клиента

При создании веб-сервиса необходимо соблюдать представленную выше структуру самих компонентов и их взаимодействия, что позволяют сделать различные подходы к построению веб-приложений. Наиболее популярными являются подходы на основе шаблона проектирования MVC («Model-View-Controller», или «Модель-Представление/Вид-Контроллер») и микросервисов.

Шаблон проектирования архитектуры приложения **MVC** предоставляет возможность разделить между собой модель данных, контроллер над моделью и представление модели для пользователя, что максимально хорошо подходит для веб-сервисов. Таким образом, в рамках данного шаблона проектирования можно соотнести веб-клиента с представлением (View), веб-сервер с контроллером (Controller) и базу данных с моделью (Model).

Такой подход является представлением монолитной архитектуры приложения, когда все функции приложения организованы в одном сервисе. Это дает не мало преимуществ, например, разработчикам не нужно задумываться о том, каким образом будут «общаться» компоненты сервера между собой. Но это также вызывает ряд проблем, таких как невозможность заменить какой-либо

отдельный модуль приложения без отключения всего сервиса или остановка всего сервиса при возникновении ошибки в какой-либо из его частей. С данными проблемами прекрасно справляется архитектура приложения на основе микросервисов.

Приложение на основе **микросервисов** создается чаще всего с помощью общего представления в виде интерфейса и общего регулятора в виде глобального переключателя маршрутов, подключающих пользователя к нужной функции приложения. Каждая отдельная функция приложения – большая или маленькая – при такой архитектуре реализуется через отдельный сервис, который выполняет только одну свою функцию. Преимущества такого архитектурного решения на лицо: если нужно отключить или обновить какой-либо микросервис, то это можно сделать без отключения всего приложения. Но также такой подход несет с собой большие сложности при коммуникации сервисов, с которыми новичок может не справиться.

Также существует способ организации архитектуры приложения, при котором оба вышеописанных подхода соединены в один: каркас сервиса написан «монолитом», но все-таки некоторые функции вынесены в отдельные микросервисы. Данный подход называется **комбинированным**. Такое решение удобно использовать, например, когда есть основная система, реализуемая главные функции приложения, и «необязательные» функции, которые пользователь может подключать самостоятельно в зависимости от его нужд.

В случае с веб-сервисом, разрабатываемом в ходе выпускной квалификационной работы, наилучшим подходом в проектировании архитектуры для начала работы было решение на основе MVC подхода, так как нужна реализация основы сервиса, разделить которую достаточно сложно. Многие бизнес-процессы, исполняемые внутри системы, тесно связаны между собой и выносить какие-либо функции или процессы во внешнюю среду системы, во-первых, сложно, во-вторых, нет необходимости. При расширении данной системы можно будет использовать комбинированный подход, создав и

подключив микросервисы к системе. Однако на начальном этапе работы создать монолитную структуру сервиса проще и быстрее, что особенно важно при сжатых сроках разработки.

1.4 Анализ веб-технологий

Для реализации выбранного архитектурного подхода в различных языках программирования существует множество решений и некоторые из них особенно выделяются. Для выявления подходящего решения были проанализированы следующие технологии:

- микрофреймворк Express (язык программирования JavaScript);
- микрофреймворк Flask (язык программирования Python);
- веб-фреймворк Sails (язык программирования JavaScript);
- веб-фреймворк Django (язык программирования Python).

Фреймворк – это программный каркас приложения, облегчающий разработку программных проектов. Микрофреймворк отличается от обычного фреймворка чаще всего своей масштабируемостью, количеством возможностей и свободой построения структуры расположения файлов внутри файловой системы. В зависимости от того, что именно разработчик хочет реализовать, можно выбрать разные технологии. Фреймворки также могут использоваться с другими технологиями, дополняющими их или расширяющими уже имеющиеся у фреймворков функции и возможности.

Express – минималистичный веб-фреймворк для backend-разработки, поддерживающий любые frontend-решения. Express считается классическим фреймворком для Node.js и пользуется большой популярностью у разработчиков, которые пишут на JavaScript.

Sails также является фреймворком для Node.js и языка программирования JavaScript. Sails основан на Express, но в отличие от его «предка» данный фреймворк предоставляет структуру шаблона MVC и использует ORM технологию для доступа к базе данных.

Flask – легкий и гибкий фреймворк на основе языка Python, который не обладает большим встроенным функционалом, но имеет огромные возможности для создания веб-сервера под любые нужды. Как часть реализации шаблона проектирования MVC Flask предоставляет поддержку шаблонов для отображения html-страниц, что является частью представления (View) в модели проектирования.

Django в отличие от Flask предоставляет полный набор инструментов для реализации базовых процессов веб-приложений, типа регистрации пользователя, панели администратора с пользовательским интерфейсом. Также, как Sails, Django имеет свою ORM и поддерживает полную архитектуру шаблона MVC.

Для выявления преимуществ, которые может предложить каждый из фреймворков, нужно понимать, что именно нужно для разрабатываемого сервиса. В техническом задании заказчик не прописал требований к технологической составляющей, однако прописал то, какие функции должны быть в сервисе, исходя из чего, можно составить критерии оценки рассматриваемых технологий – фреймворков.

В сервисе должны быть реализованы следующие функции:

- регистрация и аутентификация пользователя по логину-паролю;
- создание и регулирование роли пользователя на сервисе;
- создание и редактирование таблиц и отдельных записей баз данных;
- пользовательский интерфейс;
- разделение пользовательского интерфейса и доступа к базам данных по устанавливаемым ролям;
- отображение одних и тех же элементов по-разному в зависимости от роли пользователя на сервисе;
- администрирование контента баз данных.

Также необходимо учесть критерии чисто технологической направленности. В веб-сервисе важные такие возможности, как:

- выбор технологий СУБД;
- выбор технологий для реализации представления;
- тестирование серверной части;
- тестирование отображения (веб-части);
- создание навигации по страницам веб-приложения;
- защита от атак типа CSRF (Cross Site Request Forgery — межсайтовая подделка запроса);
- доступ к деталям HTTP-запроса;
- валидация форм и запросов от пользователя;
- инструменты «общения» с пользователем (например, при возникновении ошибок);
- инструменты обработки ошибок на стороне сервера для оповещения разработчика;
- информирование пользователя по email или SMS;
- кеширование;
- установка дополнительных плагинов и библиотек.

Исходя из вышеописанных функций и возможностей технологической направленности, был составлен список критериев для оценки фреймворков. Этот список был также дополнен критериями по организации работы разработчика:

- масштаб сообщества разработчиков;
- доступность и подробность документации;
- частота обновлений версий фреймворка;
- возможности расширения приложения в будущем.

Таким образом, было проведено сравнение по 17 критериям. По похожим критериям также сравнивались фреймворки в источниках [45, 39, 35, 38]. Результат сравнения приведен в таблицах 3.1 и 3.2.

Таблица 3.1 – Сравнение фреймворков. Анализ критериев

№	Express	Sails	Flask	Django
Встроенные технологии ORM и СУБД				
1	Нет	ORM «Waterline»	Нет	Django ORM, СУБД SQLite3
Встроенные технологии для реализации представления (пользовательского интерфейса)				
2	Нет	Нет	Шаблоны Jinja на основе HTML	Шаблоны Django на основе HTML
Встроенные инструменты тестирования				
3	Нет	Нет	Нет	Есть
Навигация по страницам веб-приложения				
4	Путь передается через обозначающий метод запроса метод объекта приложения	Путь вместе с методом запроса передается через специальный объект в специальном файле структуры проекта	Путь передается в декораторе над функцией, метод по умолчанию GET	Путь передается через специальную функцию в определенном файле структуры проекта, метод по умолчанию GET
Встроенная защита от атак типа CSRF				
5	Нет	Есть	Нет	Есть
Доступ к деталям HTTP-запроса				
6	Инструмент от Node.js	Инструмент от Node.js	Встроенный инструмент	Встроенный инструмент
Встроенные инструменты для валидации форм				
7	Есть, «ручная»	Есть, автоматическая	Есть, автоматическая	Есть, автоматическая

Инструменты обработки ошибок				
8	Есть встроенный обработчик ошибок, но нет стандартных страниц для их отображения; частая ошибка «404 Not Found» автоматически не обрабатывается	Есть встроенный обработчик ошибок, но нет стандартных страниц для их отображения; частая ошибка «404 Not Found» автоматически не обрабатывается	Есть встроенный обработчик ошибок, есть стандартные страницы для них; частая ошибка «404 Not Found» обрабатывается автоматически и также имеет стандартную страницу	Есть встроенный обработчик ошибок, есть стандартные страницы для них; частая ошибка «404 Not Found» обрабатывается автоматически и также имеет стандартную страницу
Встроенные инструменты для информирования пользователя по email или SMS				
9	Нет	Есть подключаемые модули от Sails для email и SMS оповещений	Есть подключаемые модули от Flask только для email оповещений	Есть подключаемые модули от Django только для email оповещений
Кеширование				
10	Есть условно	Есть условно	Нет	Есть
Регистрация пользователя				
11	Пишется вручную	Пишется вручную	Пишется вручную	Предоставляется стандартная регистрация:

				путь, страница, обработка
Аутентификация пользователя				
12	Пишется вручную	Есть подключаемый модуль для упрощения работы, но пишется вручную	Есть подключаемый модуль для упрощения работы, но пишется вручную	Предоставляется стандартная аутентификация: путь, страница, обработка
Пользователь как сущность базы данных				
13	Пишется вручную	Пишется вручную	Пишется вручную	Предоставляется стандартная база данных для хранения пользователей, которую можно дополнить (полями) вручную
Администрирование контента баз данных через пользовательский интерфейс				
14	Нет	Нет	Нет	Да
Масштаб сообщества разработчиков				
15	Многочисленное	Немногочис-ленное	Многочисленное	Многочисленное
Доступность и подробность документации				
16	Подробная, разбита на	Не очень подробная,	Подробная, разбита на шаги,	Подробная, разбита на

	модули, собрана в одном месте	сложная, собрана в одном месте	собрана в одном месте	модули и шаги, собрана в одном месте
Частота обновлений версий фреймворка				
17	Регулярно	Редко	Регулярно	Регулярно

Балльная система оценки фреймворков была выстроена, исходя из вышеперечисленных критериев. Каждый критерий был оценен от 1 до 3 баллов, в зависимости от его возможности разделения на подкритерии.

Таблица 3.2 – Сравнение фреймворков. Оценка фреймворков по критериям в балльной шкале

№ критерия	**Макс. кол-во баллов**	**Express**	**Sails**	**Flask**	**Django**
1	2	0	1	0	2
2	1	0	0	1	1
3	1	0	0	0	1
4	2	2	1	2	1
5	1	0	1	0	1
6	2	1	2	2	2
7	2	1	1	2	2
8	3	1	1	3	3
9	2	0	2	1	1
10	2	1	1	0	2
11	2	0	0	0	2
12	2	0	1	1	2

13	1	0	0	0	1
14	1	0	0	0	1
15	2	2	1	2	2
16	2	2	1	2	2
17	1	1	0	1	1
Итого	29	11	13	17	**27**

По результатам сравнения фреймворков был выбран **Django**. Данный фреймворк предоставляет множество полезных функций для работы с базами данных, интерфейсом, пользователями. Он имеет наибольшее количество встроенных инструментов для работы с информационной безопасностью и ошибками сервера или пользователей. Также огромным плюсом является наличие пользовательского интерфейса для администрирования баз данных, а выбранная по умолчанию СУБД и собственная ORM увеличивает темп разработки в разы, так как не нужно задумываться об установлении зависимостей или углубляться в написание сложных SQL-запросов.

1.5 Анализ серверных решений и технологий

После создания сервиса его нужно передать заказчику в виде URL-ссылки на опубликованный сайт, через который заказчик и пользователи с правом доступа к сервису будут им пользоваться. Для того чтобы это сделать, готовый сервис (пакет программ) должен быть отправлен на хостинг и запущен там. Это можно сделать также различными способами, что чаще всего это зависит от выбранного хостинга и инструментов, которые он предлагает.

Хостинг – это услуга, которая предоставляет место на каком-либо сервере для размещения в нем файлов и данных. Есть 5 видов хостингов: общий или виртуальный, VPS, VDS, облачный хостинги и выделенный сервер.

Выделенный сервер – это физический сервер, то есть компьютер, который работает только для одного клиента: разработчика, команды или компании. Это самая дорогая услуга в сфере хостингов. Такой сервер чаще всего нужен для больших и сложных проектов, когда располагающееся на сервере приложение или сервис потребляет много ресурсов или у него жесткие требования к безопасности.

Виртуальные и VPS/VDS хостинги представляют собой физический сервер, поделенный между многими пользователями. Но разделяется сервер в зависимости от вида хостинга по-разному.

Виртуальные, или общие, хостинги предоставляются путем разделения одного физического сервера на папки, покупка которых дает клиентам место для своих проектов. Этот вид хостинга самый дешевый, однако и самый простой. У клиента нет возможности регулировать настройки сервера, от которых зависит поддержка тех или иных технологий проекта, производительность, быстродействие проекта. Также здесь важно заметить, что ресурсы физического сервера в данном случае будут распределяться на все проекты – на всех клиентов именно этого сервера, которых может быть тысячи, – одинаково. А это значит, что некоторые проекты могут потреблять больше ресурсов, в то время как другим проектам этих ресурсов может не хватить. Такие зависимости связаны с большими рисками, поэтому на таких хостингах лучше всего хранить только маленькие проекты, которые не требуют ничего особенного от сервера, на котором находятся.

Хостинги типа **VPS или VDS**, в отличие от виртуальных, разделяются на отдельные виртуальные серверы, у которых есть определенное количество выделенных физическим сервером ресурсов, которое никаким образом не зависит от действий «соседей». Также этот вид хостинга предоставляет достаточно широкие возможности по настройке своего виртуального сервера. Но между этими типами хостингов есть небольшая разница. VPS (англ. Virtual Private Server, перев. Виртуальный Персональный Сервер) тип характеризуется

тем, что предоставляет клиентам сервер в рамках операционной системы с помощью технологий контейнеризации. A VDS (англ. Virtual Dedicated Server, перев. Виртуальный Выделенный Сервер) тип предоставляет полностью абстрагированный сервер, на котором клиент может самостоятельно устанавливать необходимые настройки, вплоть до операционной системы. Такие виртуальные серверы строятся на основе виртуальных машин, которые дают практически все возможности физического сервера, но по более доступной цене. Хостинги типа VPS/VDS подходят для практически любого размера проектов, а широкие возможности настройки сервера и доступная цена делают их привлекательными для большого пласта клиентов.

И **облачный** хостинг – услуга по предоставлению в пользование клиенту не один виртуальный сервер, а кластер серверов, который связывает между собой физические и виртуальные серверы. Данный тип хостинга интересен тем, что оплата за его аренду идет по факту потребления за час, то есть сколько сервер потребил в данный час ресурсов, за это количество ресурсов клиент и будет платить. Это удобно, если потребление ресурсов проекта не равномерно в течении дня или в течении года («сезонные» приложения), потребление ресурсов можно отрегулировать в зависимости от нужд проекта. Но у этого типа хостинга есть ограничения, связанные со способом размещения проектов на серверах: проекты хранятся в контейнерах. Также при таком виде хостинга у разработчика проекта не всегда есть все права для настройки сервера.

Для проекта выпускной квалификационной работы и его дальнейшего развития, с учетом того, что он разрабатывался для небольшой организации, но должен иметь возможности для беспрепятственного роста, был выбран VPS/VDS тип хостинга для реализации запуска сервиса.

Также при аренде хостинга нельзя забывать про доменное имя сервиса – адрес сайта/сервиса в сети Интернет, по которому пользователи смогут его найти и посетить. Поэтому при выборе провайдера виртуального сервера необходимо учитывать и аспект предоставления домена.

После анализа теоретической информации о хостингах был проведен анализ рейтингов провайдеров, предоставляющих услуги регистрации домена и хостинга. Таким образом, для сравнения были выбраны 4 провайдера: REG.RU, Timeweb, Beget, VDS Selectel. Критериями сравнения стали минимальная цена и ресурсы, которые предоставляются за данную цену. Сравнительный анализ провайдеров приведен в таблице 4.

Таблица 4 – Сравнение провайдеров хостингов

№	Критерий	REG.RU	Timeweb	Beget	VDS Selectel
1	Минимальная цена за доменное имя	199 руб.	99 руб.	179 руб. - регистрация + 289р./год - продление	**Бесплатно**
2	Ограничения в выборе доменной зоны	Есть	Есть	Есть	**Нет**
3	Минимальная цена за VDS/VPS хостинг	370р./мес.	290р./мес.	210р./мес.	**200р./мес.**
4	Оперативная память	**1 ГБ**	**1 ГБ**	**1 ГБ**	521 МБ
5	Хранилище (память)	15 ГБ	10 ГБ	10 ГБ	**20 ГБ**
6	Процессор (кол-во ядер)	**1**	**1**	**1**	**1**

7	Локация физических серверов	**Россия**	**Россия, Казахстан**	**Россия**	**Россия**

В результате анализа сравнительной таблицы 4 было принято решение использовать провайдера VDS Selectel. За предлагаемую минимальную цену данный провайдер не ограничивает регистрацию доменного имени, при расширении масштабов проекта предоставляет возможность докупить необходимые ресурсы для своего виртуального сервера.

Также для большей надежности и лучшей «расширяемости» в будущем было принято решение использовать технологии контейнеризации – процесс, когда приложение или проект упаковывается в единицу программного обеспечения, содержащую всё необходимое для его нормальной работы. Таких технологий развито не очень много и самой популярной является технология Docker, которую было принято решение использовать для контейнеризации разрабатываемого веб-сервиса.

1.6 Выводы к главе 1

В ходе анализа предметных областей проекта, таких как бизнес-процессы салона красоты, архитектура веб-сервиса, веб-технологии и серверные технологии, были решены три из поставленных в начале работы задач. По итогам анализа архитектурных решений для веб-сервисов было принято решение реализовать архитектуру сервиса на основе модели MVC. Этот способ организации архитектуры проекта способствует эргономичной реализации веб-сервиса. На основе этого решения и анализа веб-технологий далее было принято решение о фреймворке, с помощью которого должен разрабатываться сервис. Этим фреймворком был выбран Django, написанный для языка программирования Python. При помощи данного фреймворка необходимые салону красоты бизнес-процессы могут быть реализованы достаточно быстро, что очень важно для разработки в сжатые сроки. И для публикации веб-сервиса

в сети Интернет был выбран провайдер VDS хостингов и доменов VDS Selectel, который предоставляет удобную работу с виртуальным сервером, а также не требует много вложений. Для упрощения дальнейшей работы сервиса также было принято решение об использовании технологий Docker для контейнеризации проекта перед отправкой его на сервер.

После принятия решений о выборе стека технологий, на которые будет опираться проект, была начата работа над реализацией MVP (*англ.* Minimum Viable Product, *перев.* Минимальный жизнеспособный продукт) веб-сервиса.

ГЛАВА 2. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ВЕБ-СЕРВИСА

2.1 Проектирование архитектуры MVP-сервиса

Для нормальной работы сервиса и возможности его безболезненного расширения в будущем необходимо продумать его архитектуру – какие компоненты будут в проекте, какие связи между этими компонентами должны быть и каким образом они должны поддерживаться.

Разрабатываемый веб-сервис было решено реализовывать по шаблону проектирования MVC, но с возможностью расширить его в будущем с помощью микросервисов. Эта возможность должна быть поддержана изначально, поэтому при проектировании связей между компонентами архитектуры это также стоит учесть.

Большую часть работы по созданию архитектуры приложения выполняет выбранный фреймворк Django. При создании проекта Django изначально создает некоторую структуру внутри проекта и приложений, которые находятся внутри этого проекта. Рассмотрим структуру проекта, которую предоставляет Django по умолчанию (рис. 3) [41].

Рисунок 3 – Структура проекта Django

В дефолтном проекте Django создает корневую папку *mysite*, в которой содержится весь ваш проект. В этой папке лежат файл *manage.py*, являющийся утилитой командной строки для работы с проектом различными способами, и папка *mysite,* являющаяся пакетом Python. Внутри данного пакета содержаться файлы конфигураций проекта, главным из которых является *settings.py*, он содержит главные настройки проекта. Также не маловажным является файл *urls.py* – в нем прописываются главные роуты – пути запросов – вашего проекта.

При такой структуре Django уже может вполне нормально функционировать, если далее организовывать всю работу приложения внутри пакета *mysite*. Однако это не позволит проекту расширяться. Поэтому для полноценного функционирования Django предоставляет такой инструмент как создание приложений внутри проекта со своей архитектурой. Приложения в Django – это отдельные пакеты, которые могут находится где угодно и подключаться или отключаться в зависимости от нужд вашего проекта. Такое приложение по умолчанию имеет структуру, представленную на рисунке 4 [41].

```
__init__.py
admin.py
apps.py
migrations/
    __init__.py
models.py
tests.py
views.py
```

Рисунок 4 – Структура приложения Django

Рассмотрим подробнее, что из себя представляет каждый из компонентов этого пакета (приложения):

- *__init__.py* – пустой файл, который показывает, что приложение Django – это пакет Python.

- *admin.py* – файл для регистраций созданных моделей в панели администратора, предоставляемой Django.
- *apps.py* – файл, создающий конфигурацию приложения для Django. В этом файле указывается имя приложения, по которому оно будет подключаться к проекту Django.
- *migrations* – папка, в которой хранятся все логи изменений моделей приложения.
- *models.py* – файл, в котором создаются модели Django. Модель – это представление таблицы базы данных.
- *tests.py* – файл для тестирования приложения.
- *views.py* – файл для создания отображений страниц приложения.

Также в приложение Django добавляют:

- файл *urls.py*, если внутри него необходимо организовать несколько представлений по разным путям-роутам;
- папку *templates* для организации html-шаблонов;
- папку *static* для организации файлов js, css, а также картинок, которые необходимы для сервиса;
- файл *forms.py*, если необходимо составить кастомную форму для передачи каких-либо данных от пользователя на сервер.

Исходя из вышеописанной структуры проекта и приложений Django можно говорить о том, что архитектура проекта Django полностью соответствует выбранному в части 1.3 данной работы шаблону проектирования MVC. Это выражается следующим образом:

- Файлы *models.py* соответствуют компоненте «Модель» шаблона, так как именно в этом файле посредством ORM Django сервис может создавать таблицы базы данных и впоследствии управлять ими.

- Файлы *views.py* и *urls.py* соответствуют компоненте «Контроллер», так как именно этот файл «ответственен» за то, что будет показано пользователю, какие данные должны быть отображены пользователю, а какие данные должны быть получены от пользователя и внесены в базу данных.
- Шаблоны представления, которые представляют из себя файлы html, js, css стека, соответствуют компоненте «Представление/Вид», так как они берут на себя функцию отображения данных, передаваемых через Контроллер.

В связи с такой структурой проекта и приложений Django целесообразно для начала разобраться с тем, какие приложения нужны в разрабатываемом MVP-сервисе и как именно они будут связываться между собой и с базой данных, а также какие роуты нужны будут для реализации полного функционала MVP-сервиса.

Главный пакет (приложение) проекта – *BeautyManageService* – должен подключать другие приложения, нужные для реализации сервиса. В данном случае это два приложения:

- *main* – приложение для отображения главной страницы сайта салона, а также для работы с базой пользователей.
- *office* – приложение для создания кабинетов пользователей для разных ролей.

В связи с «ролями» этих приложений в них должны содержаться все необходимые модели и представления, то есть файлы *models.py* и *views.py*, а также шаблоны html должны содержать необходимый код для реализации наполнения базы данных и отображения необходимых данных на страницах пользователей.

Таким образом, в приложении *main* в файле *models.py* должна быть создана модель *Profile (Профиль)*, которая будет дополнять модель *User*

(Пользователь). А в файле *views.py* должны содержаться отображения регистрации пользователя и его профиль, которые также должны быть реализованы через html-шаблоны. В приложении *office* в файле *models.py* должны содержаться модели *Shift (Смена)* для составления смен мастеров салона, где необходима связь с моделью *Profile*, и *Appointment (Запись)* для создания записей клиентов в салон, где должны быть реализованы две связи: с моделью *Shift* для определения мастера и с моделью *Service (Услуга)*, которая будет определять стоимость услуги и ее длительность, а также с моделью *Profile* для определения клиента данной записи. Модель *Service* должна быть также реализована внутри приложения *office*. А в файле *views.py* этого приложения и в предоставляющихся им шаблонах отображения необходимо реализовать кабинеты администратора, сотрудника и клиента в зависимости от роли пользователя.

Данное описание архитектуры проекта можно представить в виде схемы (рис. 5):

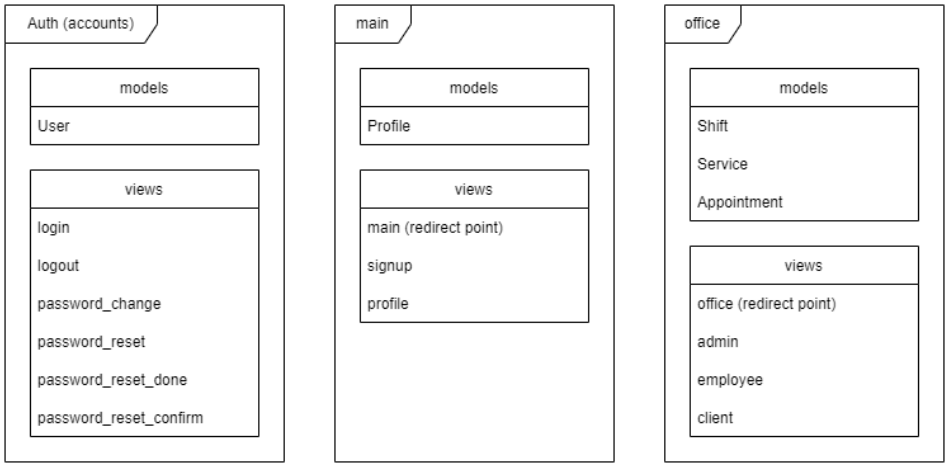


Рисунок 5 – Архитектура разрабатываемого сервиса

Также, чтобы понимать связи между моделями и их сущностями в базе данных, нужно составить диаграмму классов, которая в полной мере сможет показать, какие таблицы будут в базе данных, а также какие поля необходимо

предусмотреть в различных формах, таких как форма для регистрации пользователя. Диаграмму классов для разрабатываемого MVP-сервиса представлена на рисунке 6.

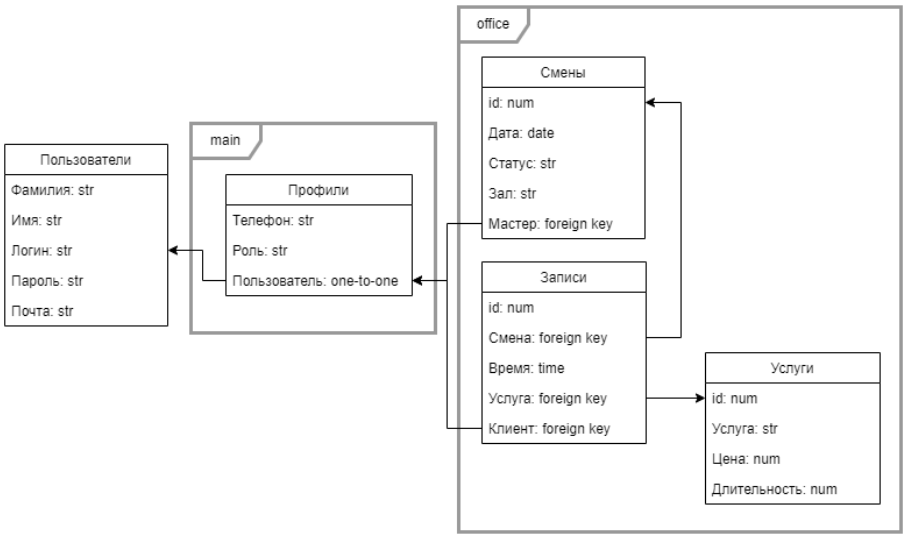


Рисунок 6 – Диаграмма классов для разрабатываемого сервиса

Затем, когда общая архитектура MVP-сервиса стала понятна, необходимо продумать, какие роуты (пути) будет обслуживать сервис и что именно по этому пути будет получать пользователь. Для описания этих роутов была составлена таблица 5, представленная ниже [27].

Таблица 5 – Описание роутов сервиса

№	Роут	Название роута	Описание роута
Приложение *main*			
1	/	main	• Роут главной страницы сайта салона. Для MVP-сервиса она не должна ничего отображать, но она является важной составляющей для дальнейшего развития сервиса. • Перенаправлять пользователя на страницу входа, если пользователь не авторизован.

			• Переправлять пользователя в личный кабинет его роли, если он авторизован.
2	accounts/ login/	login	• Роут страницы входа для зарегистрированного пользователя. Должна отображать также переход на страницу регистрации. • Есть страница по умолчанию. • После авторизации перенаправлять в личный кабинет.
3	accounts/ logout/	logout	• Роут для выхода из аккаунта. • Не обязательно иметь отдельную страницу. • Перенаправление на главную страницу.
4	accounts/ password _change/	password _change	• Роут страницы изменения пароля пользователя. • Есть страница по умолчанию.
5	accounts/ password _change/ done/	password _change _done	• Роут для оповещения об успешном изменении пароля. • Не обязательно иметь отдельную страницу. • Перенаправление на страницу профиля с уведомлением об успешном изменении пароля.
6	accounts/ password _reset/	password _reset	• Роут для сброса пароля на электронную почту пользователя. На странице пользователь вводит свою электронную почту. • Есть страница по умолчанию.
7	accounts/ password _reset/ done/	password _reset _done	• Роут для оповещения об успешном сбросе пароля. Страница с просьбой проверить свою почту пользователя. • Есть страница по умолчанию.

8	accounts/ reset/ <uidb64>/ <token>/	password _reset _confirm	• Роут для подтверждения сброса пароля. На этой странице пользователь вводит новый пароль после нажатия ссылки в электронном письме с возвратом пароля. • Есть страница по умолчанию.
9	accounts/ reset/ done/	password _reset _complete	• Роут для оповещения об успешном изменении пароля после его сброса. • Не обязательно иметь отдельную страницу. • Перенаправление на страницу профиля с уведомлением об успешном изменении пароля.
10	accounts/ signup/	signup	• Роут страницы регистрации пользователя. На этой странице пользователь вводит все необходимые данные для заполнения модели *User* и модели *Profile*. • После прохождения регистрации перенаправлять на страницу входа.
11	accounts/ profile/	profile	• Роут страницы профиля. На этой странице отображаются данные пользователя, а также есть возможность эти данные изменить через ссылки на другие роуты.
Приложение *office*			
12	office/	office	• Роут для перенаправления пользователя в его личный кабинет по его роли. • Если пользователь авторизован, то перенаправляет по роутам admin, client или employee. • Если пользователь не авторизован, перенаправляет на главную страницу.

13	office/ admin/	admin	• Роут личного кабинета администратора салона. На странице отображаются календари смен и записей, а также меню с возможностью перехода на другие страницы, например, страницу профиля.
14	office/ client	client	• Роут личного кабинета клиента салона. На странице отображается возможность онлайн-записи к мастеру.
15	office/ employee	employee	• Роут личного кабинета сотрудника салона. На страницы отображаются календарь смен и записей пользователя-мастера, а также меню с возможностью перехода на другие страницы, например, страницу профиля.

После того, как стало понятно, какие страницы должны быть реализованы в MVP-сервисе, необходимо было подготовить макеты страниц вместе с их дизайном, чтобы предоставить их заказчику на утверждение.

2.2 Подготовка макетов и дизайна страниц сервиса

Чтобы начать реализацию проекта необходимо понимать не только какие страницы будут на сервисе, но и как они будут выглядеть. Для этого необходимо составить макеты страниц и общий концепт дизайна.

При создании макетов и дизайна сервиса необходимо разобрать требования заказчика из технического задания. Так как в техническом задании не было конкретных правил для дизайна сервиса, была проведена беседа с заказчиком. По итогу беседы было выявлено, что жестких требований по дизайну у заказчика нет, но было достигнуто понимание того, какие цвета необходимо использовать в дизайне сервиса, чтобы соответствовать бренду. Цвета, которые необходимы для реализации дизайна веб-сервиса, представлены в таблице 6.

Таблица 6 – Цвета сервиса

№	Название цвета	Hex-код
1	белый	#ffffff
2	черный	#000000
3	фиолетовый	#8c6dbd
4	розовый	#da75f5

На каждой странице необходимо использовать одинаковую структуру: шапка *(header)* – верхняя часть страницы, контент *(content)* – часть, где располагается вся основная информация страницы, и подвал *(footer)* – нижняя часть страницы. В качестве общей концепции дизайна страниц было решено использовать одинаковый дизайн для каждой шапки и подвала всех страниц. Общая концепция дизайна для шапки представлена на рисунке 7, а для подвала – на рисунке 8.

Рисунок 7 – Дизайн шапки

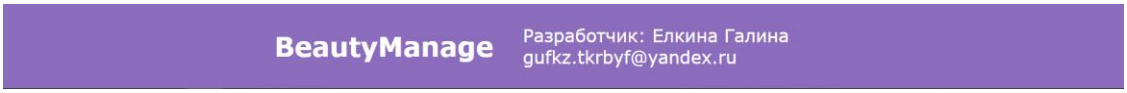


Рисунок 8 – Дизайн подвала

Помимо шапки и подвала на странице необходимо продумать, каким образом будут расположены компоненты внутри основного блока *(content)*. Для этого необходимо понять, что будет внутри каждой страницы и создать на этой основе макет страницы.

Исходя из требований заказчика, были составлены макеты 5 страниц: страница для формы (любой формы, которую понадобиться отобразить), страницы личных кабинетов администратора, сотрудника, клиента и страница профиля пользователя.

Так как страниц с формами необходимо реализовать 6 штук (роуты, такие как password_change), макет данной страницы должен быть максимально универсальным. На странице для формы было необходимо учесть размеры самой формы, так как поля формы могут быть абсолютно разными. Данную особенность было решено оформить через начертание в макете некоторых полей пунктирными линиями, что также указано в описании к рисунку макета.

Для страницы профиля пользователя необходимо было учесть, какая информация о пользователе должна отображаться в профиле, какие возможности должны быть отражены в профиле. В данном случае под возможностями подразумеваются переходы на страницы с формой, например, для изменения пароля.

Представления личных кабинетов зависят от роли пользователя. Поэтому для личных кабинетов по каждой роли необходимо было разработать свой макет.

На странице администратора должны быть два календаря (смен и записей), с возможностью редактирования записей в календарях через модальное окно. Макет модального окна не отличается от макета страницы для формы, так как модальное окно будет представлять из себя форму.

На странице личного кабинета сотрудника должны быть отражены список непринятых смен, календарь всех смен, предлагаемых мастеру, и календарь записей клиентов, которые есть у мастера. Так же, как у администратора, у мастера есть возможность редактирования некоторых записей через подобное модальное окно.

На странице клиента в MVP-сервисе должно отображаться только окно для записи в салон, которое представляет из себя форму, которое само по себе не требует отдельной реализации макета. Однако, как и в других кабинетах, в кабинете клиента есть возможность переключиться в свой профиль, которую также необходимо учесть.

Все реализованные макеты страниц находятся в приложении В. На основе данных макетов далее были реализованы html-шаблоны для соответствующих страниц.

2.3 Реализация MVP-сервиса с использованием выбранных технологий

Перед началом работы над сервисом необходимо было удостовериться в том, что установлен язык программирования Python и фреймворк Django, так как с данными инструментами разработки уже велась работа. После проверки, что все необходимые инструменты установлены на компьютер и их версии актуальные, был подготовлен Git-репозиторий, в котором содержится вся работа над проектом. Git – это консольная утилита, которая используется для сохранения и ведения истории изменений проекта. Репозиторий – это папка, в которой хранится проект [44]. Исходя из этих определений, Git-репозиторий представляет собой не простое хранилище, а хранилище с возможностью вернуть старую версию проекта, если это необходимо. Локальный Git-репозиторий также может быть соединен с удаленным репозиторием, который будет находится на каком-либо сайте-хранилище. Чаще всего для этих целей используют GitHub. Поэтому для данной выпускной квалификационной работы был создан как локальный, так и удаленный репозиторий для хранения проекта не только локально, но и в Интернете.

Создание проекта Django

Далее необходимо было в соответствии с разработанной архитектурой MVP-сервиса создать проект Django, внутри которого создать два приложения: *main* и *office*. Данные действия также делались через командную строку терминала редактора VS Code (рис. 9).

```
PS C:\MyCodes\Study\Diploma> django-admin startproject BeautyManageService
PS C:\MyCodes\Study\Diploma> python manage.py startapp main
C:\Users\gufkz\AppData\Local\Programs\Python\Python38-32\python.exe: can't open file 'manage.py': [Errno 2] No such file or directory
PS C:\MyCodes\Study\Diploma> cd beautymanageservice
PS C:\MyCodes\Study\Diploma\beautymanageservice> python manage.py startapp main
PS C:\MyCodes\Study\Diploma\beautymanageservice> python manage.py startapp office
```

Рисунок 9 – Создание проекта и его приложений через терминал

Затем были созданы папки для шаблонов и статических файлов, таких как картинки и файлы js, css.

Далее созданные приложения были подключены к проекту через файл *setting.py*, и в нем же были сделаны некоторые изменения, которые влияют на язык сервиса, особенно на встроенную панель администратора, добавлена строчка для обозначения пути к папке *templates* и проверено наличие пути к папке *static*.

В первую очередь был создан суперпользователь Django, который получает все права редактирования сайта, что необходимо разработчику. Это было сделано при помощи консоли (см. рис. 10).

```
PS C:\MyCodes\Study\Diploma\diplomaproject> python manage.py createsuperuser
Имя пользователя (leave blank to use 'gufkz'): beautiful_admin
Адрес электронной почты:
Password:
Password (again):
Superuser created successfully.
PS C:\MyCodes\Study\Diploma\diplomaproject>
```

Рисунок 10 – Создание суперпользователя Django

Затем были настроены роуты-пути. Для этого в папке *BeautyManageService* в файле *urls.py* был написан следующий код (рис. 11):

```
16  from django.contrib import admin
17  from django.urls import include, path
18
19  urlpatterns = [
20      path('', include('main.urls')),
21      path('admin/', admin.site.urls),
22      path('accounts/', include('django.contrib.auth.urls')),
23      path('office/', include('office.urls'))
24  ]
```

Рисунок 11 – URL-пути в главном файле

- 20 строка – подключение приложения *main* по корневому пути;
- 21 строка – подключение страницы панели администратора;
- 22 строка – подключение роутов для работы с данными пользователя;
- 23 строка – подключение приложения *office* по пути "office/".

В папках приложений также были созданы файлы *urls.py* для создания в них роутов для внутренних страниц приложений в соответствии с роутами, описанными в таблице 5. Полный код проекта можно посмотреть по ссылке из приложения Г.

Затем были созданы html-шаблоны и css-файлы в папках *templates* в обоих приложениях. Шаблоны и стили были созданы, исходя из составленных ранее макетов. Также были созданы базовый шаблон для всех страниц, шаблон шапки и два шаблона для подвалов для разных ролей, так как было принято решение, что подвалы сотрудников и клиентов должны отличаться. Таким образом, в проекте появились следующие файлы:

- base.html – базовый шаблон, по которому будут строиться все остальные страницы;
- index.css – базовые стили для всего проекта, подключаемые к каждой странице проекта;
- form.html – шаблон для страниц форм;
- profile.html – шаблон страницы профиля пользователя;
- admin.html – шаблон для страницы личного кабинета администратора;
- employee.html – шаблон для страницы личного кабинета сотрудника;
- client.html – шаблон для страницы личного кабинета клиента;
- admin_footer.html – шаблон подвала страницы для сотрудников;
- client_footer.html – шаблон подвала страницы для клиентов.

Также важно отметить, что шаблоны *form*, *profile* и *client* включают в себя базовый шаблон и шаблон подвала для клиентов, а шаблоны *admin* и *employee* вместо подвала клиента содержат шаблон подвала для сотрудников. Файл *index.css* подключен к файлу *base.html*, поэтому подключен к каждому из имеющихся шаблонов.

Затем с помощью имеющейся в Django панели администратора были созданы дополнительные 3 пользователя для тестирования системы с разными ролями: test_client, test_employee и test_employee2. На рисунке 12 представлено то, как выглядит административная панель Django при регистрации нового пользователя или изменении имеющегося.

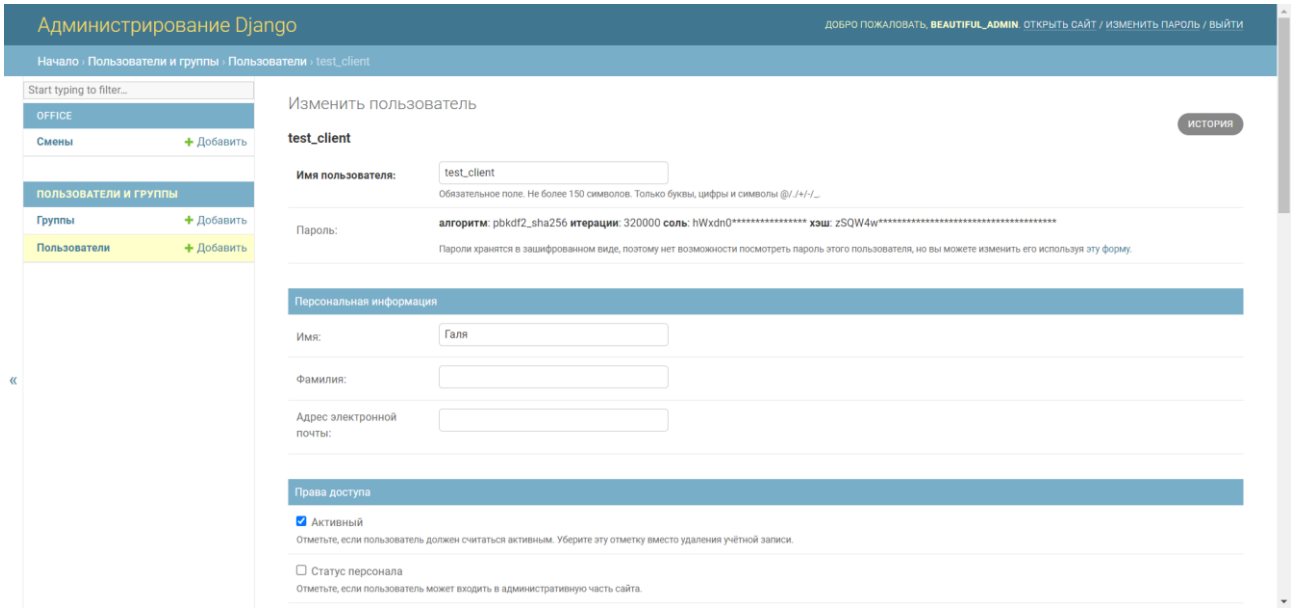


Рисунок 12 – Административная панель Django. Пользователь test_client

Создание моделей MVP-сервиса

Следующим шагом необходимо было добавить роли пользователям. Чтобы добавить роли, необходимо было расширить имеющуюся в Django модель пользователя. Для того чтобы выяснить, как это сделать, были изучены источники 25, 23, 22. В ходе изучения были выявлены 4 пути расширения модели [22]:

- использование прокси-модели;
- использование связи «один-к-одному» с пользовательской моделью;
- создание модели пользователя с помощью расширения класса AbstractBaseUser;

- создание модели пользователя с помощью расширения класса AbstractUser.

Наиболее подходящим оказался способ с использованием связи «один-к-одному», так как переписывать пользовательскую модель Django нет необходимости, как это предполагает способ с расширением класса AbstractBaseUser, изменять в ней что-то также не нужно (способ через расширение класса AbstractUser), но при этом добавить в модель только новые методы, как в случае с прокси-моделью, недостаточно. Поэтому в приложении *main* в файле *models.py* была создана модель Профиль *(Profile)*, соответствующая классу Профили на диаграмме классов, которая была создана на предыдущем этапе работы.

Данная модель организована так, чтобы при удалении записи пользователя, данные записи этой модели, связанные с удаленным пользователем, также удалялись. Роль пользователя определяется списком значений, но по умолчанию для новых пользователей выставляется роль клиента. Это связано с тем, что клиенты салона в последствии будут регистрироваться на сайте салона самостоятельно, но при этом у них должен оставаться доступ только к ограниченной информации. Роли сотрудников определяются администратором, а роли администраторам назначает суперпользователь через административную панель Django. Телефонный номер пользователя проверяется через валидатор с регулярным выражением. Телефон является необязательным полем в модели, так как не может быть обязательным из-за правил Django, однако при регистрации пользователя через формы это поле будет считаться обязательным по требованию заказчика.

Затем модель была зарегистрирована в файле *admin.py* для того, чтобы она могла быть обнаружена Django и отображалась в панели администратора. Чтобы отобразить ее корректно, в файле был написан следующий код (рис. 13):

```
DiplomaProject > main > admin.py > ...
1  from django.contrib import admin
2  from django.contrib.auth.admin import UserAdmin
3  from django.contrib.auth.models import User
4  from .models import Profile
5
6  class UserInline(admin.StackedInline):
7      model = Profile
8      can_delete = False
9      verbose_name = 'Доп. информация'
10
11 class UserAdmin(UserAdmin):
12     inlines = (UserInline, )
13
14 admin.site.unregister(User)
15 admin.site.register(User, UserAdmin)
```

Рисунок 13 – Код файла admin.py из приложения main

После такого дополнения через административную панель можно было поменять роли уже созданным пользователям. Пользователю test_client была выдана роль клиента, суперпользователю – роль администратора, а остальным пользователям – роли сотрудников (см. рис. 14).

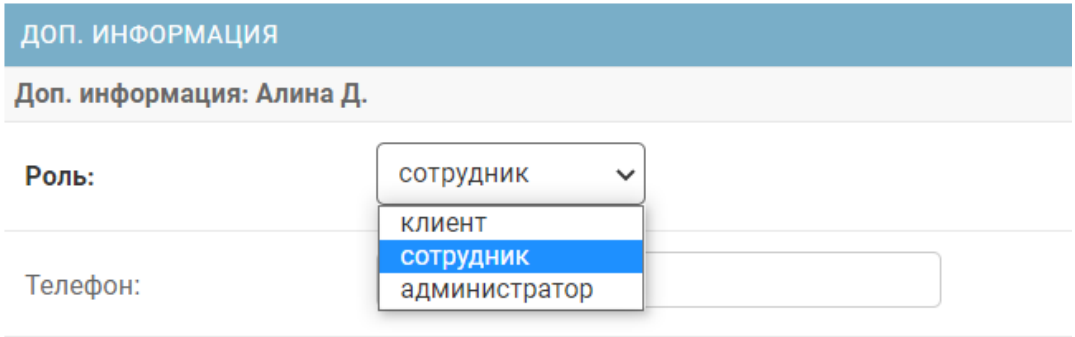


Рисунок 14 – Добавление роли пользователю test_employee

Далее была начата работа над страницей администратора. И для того, чтобы данная страница могла функционировать необходимо было реализовать оставшиеся модели базы данных: *Shift* (Смена), *Appointment* (Запись) и *Service* (Услуга) – в приложении *Office*.

Для модели *Shift* (Смена) необходимо было создать два поля с выбором значений из выпадающего списка: статус смены и зал смены, – поле для выбора даты смены и поле с выбором мастера. Последнее поле должно быть связано с моделью пользователя через модель *Profile*, так как необходимо делать выбор

только среди сотрудников, то есть составлять фильтр по пользователям с ролью «сотрудник». Но при этом необходимо учесть, что сами смены исчезать при удалении пользователя не должны. Поэтому при удалении пользователя, то есть увольнении мастера, смена переходит к удаленному пользователю, который создается единожды, а затем используется, с помощью данного кода (рис. 15):

```python
from django.db import models
from django.contrib.auth import get_user_model
from django.utils.timezone import now
from main.models import Profile

def get_deleted_user():
    return get_user_model().objects.get_or_create(username='deleted')[0]
```

Рисунок 15 – Код для удаленного пользователя

Код поля для выбора мастера в модели при этом выглядит, как представлено на рисунке 16.

```python
master = models.ForeignKey(
    Profile,
    on_delete=models.SET(get_deleted_user),
    limit_choices_to={'role': 'E'},
    verbose_name='Мастер смены'
)
```

Рисунок 16 – Код поля master в модели Shift

Модель *Service* (Услуга) была реализована с помощью простых полей:

- название услуги – поле для записи текста;
- цена – число с плавающей точкой;
- длительность – число минут.

Модель *Appointment* (Запись) включает в себя простое поле времени для записи, а также внешние поля для связи смены, профиля клиента и предоставляемой услуги с записью. Внешние поля связываются через внешние ключи также, как связывался мастер в модели *Shift*. Это дает возможность гибко работать с записями, даже если записи внешних таблиц данных будут удалены.

Полный код моделей можно посмотреть по ссылке из приложения Г.

Форма входа

После создания моделей MVP-сервиса необходимо было создать форму авторизации пользователя для дальнейшего тестирования продукта. Поэтому был создан в приложении *main* файл *login.html* в папке *registration* для отображения формы входа с использованием шаблона формы.

В файле *views.py* для перенаправления пользователя был написан код, представленный на рисунке 17:

```
from django.shortcuts import render, redirect

def index(request):
    if request.user.is_authenticated:
        return redirect('office')
    return redirect('login')
```

Рисунок 17 – Код функции index из приложения main

Таким образом, если пользователь авторизован, то он будет перенаправлен в свой личный кабинет, а если нет, то на страницу входа.

Пути личного кабинета пользователя

Следующим этапом была реализация путей к личному кабинету пользователя. Для этого необходимо было предусмотреть отображение страницы по нужному URL. Авторизованный пользователь будет переходить в личный кабинет по роуту с названием *office*, как было написано выше. Этот роут относится к приложению *office* и является его корневым роутом. Так как в файле *urls.py* уже были созданы необходимые пути, их необходимо было обработать соответствующим образом. При переходе в корневой путь приложения пользователя необходимо перенаправлять в личный кабинет по его роли, если он авторизован, и на страницу входа, если пользователь не авторизован. Для этого в файле *views.py* был написан следующий код (рис. 18):

```
def index(request):
    if request.user.is_authenticated:
        if request.user.profile.role == 'A':
            return redirect('admin')
        elif request.user.profile.role == 'E':
            return redirect('employee')
        elif request.user.profile.role == 'C':
            return redirect('client')
    return redirect('login')
```

Рисунок 18 – Код функции index из приложения office

Также для каждого роута была организована своя функция, подобная представленной ниже (рис. 19):

```
def client(request):
    if request.user.is_authenticated and request.user.profile.role == 'C':
        return render(request, 'client.html')
    return redirect('office')
```

Рисунок 19 – Код функции client из приложения office

Функционал личного кабинета администратора

Далее был реализован непосредственно функционал личного кабинета администратора. В работу входили реализация двух календарей и модального окна, реализация логик переключения между компонентами и переходов на другие страницы, работа сервера с базой данных и работа веб-клиента с полученной от сервера информацией.

Работа была начата с реализации отображения самого календаря, причем составить реализацию необходимо было таким образом, чтобы ее можно было использовать не только для отображения одного календаря, например, смен, но и для отображения всех остальных календарей, в том числе на странице сотрудника. Поэтому было принято решение создать относительно независимый пакет файлов, работающих на стороне клиента. Такое решение было принято в связи с тем, что «сборка» отображения календаря должна происходить в одном месте, так как необходимо обработать каждый день отображаемого месяца. Делать данную операцию на стороне сервера слишком долго, поэтому в реализации отображения календаря используется следующий алгоритм: веб-

сервер отправляет веб-клиенту вместе с шаблоном страницы данные в формате JSON, а клиент эти данные, обрабатывая, заносит в нужные ячейки календаря в нужном ему формате. Данная реализация календаря была выполнена в трех файлах:

- *calendar.html* – шаблон для отображения табличного вида календаря с кнопками;
- *calendar.css* – файл стилей отображения календаря: кнопок, надписей, ячеек;
- *calendar.js* – скриптовый файл, который отвечает за прорисовку внутри html-файла информации от сервера – от составления ячеек календаря и расстановки дней до стилевого оформления, а также за переключение месяцев по кнопкам.

Таким образом, отрисовка календаря превратилась в одну единственную функцию, вызов которой выглядит следующим образом:

```
createCalendar(shiftsCalendar, 'Календарь смен', renderShifts);
```

где первый аргумент – это место, куда именно должен «встать» календарь, второй аргумент – название календаря, третий – функция отрисовки дополнительной информации на день, например, функция отрисовки смены.

Следующим шагом стала реализация функций по отрисовке дополнительной информации: смен мастеров и записей клиентов.

Для этого на стороне сервера в файле *views.py* была реализована передача данных всех смен и записей клиентов, хранящихся в базе данных, в формате JSON, а на стороне клиента в файле *admin.js* и сопутствующих ему файлах – получение этих данных и их обработка в отдельных функциях, которые, передаваясь в функцию отрисовки календаря третьим аргументом, отображали всю необходимую информацию. В ходе составления реализации этих функций в модели смен и записей были также добавлены новые методы, которые упрощали работу по получению данных.

Затем для работы с записями и базой данных – для изменений существующих записей и добавления новых – был реализован шаблон модального окна в файле *modal.html*. На стороне клиента был реализован механизм показа и скрытия модального окна, а также передача на сторону сервера запроса для отображения необходимой информации в данном окне. На стороне сервера была реализована логика получения и обработки запроса с параметрами, которые отвечали бы за получение информации из базы данных по определенной записи и отправление этих данных веб-клиенту.

Для форм, которые должны отображаться в модальном окне, был создан файл *forms.py* в приложении *office*. Внутри этого файла были созданы формы для создания и редактирования смен мастеров и записей клиентов.

Для полного функционирования личного кабинета администратора необходимо было также добавить работу с пользователями: регистрация и профиль пользователя, что стало следующим шагом в реализации MVP-сервиса.

Форма регистрации и профиль пользователя

В связи с требованиями заказчика форму регистрации пользователя, которую предоставляет Django, использовать было нельзя, так как в ней не отображалось достаточно полей. Поэтому необходимо было добавить собственную форму для регистрации. Для этого были созданы следующие файлы:

- *signup.html* в папке *registration* - шаблон для отображения формы регистрации;
- *forms.py* – файл приложения main, в котором хранится код кастомной формы.

Внутри файла *forms.py* необходимо было составить две формы, которые бы отвечали за наполнение двух моделей базы данных: модель *User* и модель *Profile*. В данном файле были обозначены обязательные к заполнению поля,

подсказки для пользователя и отображаемые в форме поля в соответствии с требованиями заказчика. В файле *views.py* была реализована проверка валидности введенных данных, их сохранение и ответ пользователю в случае ошибки или перенаправление на страницу входа в случае успеха завершения регистрации. Также на стороне сервера было реализовано сохранение пользователя с различными ролями в зависимости от типа регистрации, которая определялась параметрами запроса.

Для реализации профиля пользователя были добавлены дополнительные html-шаблоны также в папку *registration* [51, 27]:

- *password_change_form.html* для изменения пароля пользователя;
- *password_reset_form.html* для сброса пароля пользователя;
- *password_reset_done.html* для уведомления пользователя о том, что ему нужно подтвердить через электронную почту сброс пароля;
- *password_reset_confirm.html* для изменения пароля пользователя после его сброса;

А также в шаблон страницы профиля добавлена возможность отображения закрывающихся уведомлений о том, что пароль пользователя был изменен.

Так как все необходимые роуты, кроме роута *profile*, поддерживаются Django самостоятельно, заниматься их обработкой было не нужно. Обработка роута *profile* была написана в файле *views.py* в приложении *main*.

Личные кабинеты сотрудника и клиента

Следующим этапом работы стала реализация личного кабинета сотрудника. Для реализации кабинета сотрудника уже были подготовлены шаблоны самой страницы кабинета и модального окна для редактирования записей и отображение календаря. Поэтому в ходе этого шага были написаны:

- код обработки запроса в файле *views.py* в приложении *office* в функции *employee* так, чтобы шаблон страницы и модального окна могли получать необходимую им информацию о самом сотруднике и его записях в базе данных из других таблиц;
- файл *employee.js*, который обрабатывал информацию, полученную от сервера и также, как файл *admin.js*, преобразовывал ее для отображения внутри страницы кабинета сотрудника.

Как уже было описано ранее, кабинет клиента в MVP-сервисе должен представлять из себя лишь форму для онлайн записи в салон. Поэтому в файле *forms.py* приложения *office* был написан код данной формы, а на стороне сервера реализовано обработка и сохранение записи клиента. Также в шаблон client.html была добавлена возможность показа закрывающегося уведомления о том, что запись была успешно сохранена.

Полный код проекта можно посмотреть по ссылке из приложения Г.

Предоставление доступа заказчику к MVP-сервису

После успешной апробации сервиса на локальном хосте и встречи с заказчиком для принятия работы, сервис был отправлен на VDS-сервер.

Чтобы это сделать, на сайте провайдера VDS Selectel был создан виртуальный сервер VDS типа. Также был создан домен *beautymanage.com* для этого сервера на этом же сайте. Затем на основе изученных источников [40, 7, 9] были написаны файлы для Docker-контейнера:

- Dockerfile;
- requirements.txt;
- docker-compose.yml.

Далее через консоль виртуального сервера был клонирован Git-репозиторий сервиса вместе с файлами для Docker-контейнера и установлены необходимые инструменты для дальнейшего развертывания контейнера. И после

проверки того, что все действительно установилось, Django-проект был упакован в Docker-контейнер, а контейнер запущен на виртуальном сервере, который доступен по адресу *beautymanage.com*.

2.4 Выводы к главе 2

На данном этапе работы были спроектирована архитектура сервиса и составлены макеты страниц. Дизайн сервиса был обсужден с заказчиком и в итоге был реализован в страницах сервиса. MVP-сервис был реализован в соответствии с требованиями заказчика и выявленными в ходе данной работы требованиями.

ЗАКЛЮЧЕНИЕ

Для создания MVP-сервиса были проанализированы предметные области не только в сфере IT, но и в сфере бизнеса. Архитектура сервиса была спроектирована на основе популярной модели проектирования MVC, а сам проект построен с помощью одного из самых быстрорастущих и развивающихся фреймворков Django. Для упрощения и удешевления разработки и поддержки сервис был запущен на виртуальном хостиге и был успешно внедрен в салон красоты «BeautyManage», от владельца которого был получен заказ на разработку данного сервиса.

Таким образом, в ходе выпускной квалификационной работы были успешно выполнены все поставленные задачи, и цель работы была достигнута. Был разработан MVP-сервис, который располагается по адресу *beautymanage.com* в сети Интернет. Заказчику были переданы логин и пароль от пользователя-администратора для начала работы с сервисом. Код полученного проекта можно посмотреть по ссылке из приложения Г, а скриншоты страниц сервиса – в приложении Д.

ЛИТЕРАТУРА

1. Анализ и управление бизнес-процессами // Варзунов А. В., Торосян Е. К., Сажнева Л. П. Учебное пособие. – СПб: Университет ИТМО, 2016. – 112 с.

2. Архитектура и фреймворки веб-приложений: учебное электронное пособие // К. А. Кулаков, В. М. Димитров; М-во науки и высшего образования Рос. Федерации, Федер. гос. бюджет. образоват. учреждение высш. образования Петрозавод. гос. ун-т. — Электрон. дан. — Петрозаводск: Издательство ПетрГУ, 2020. – 59 с.

3. Архитектура микросервисов // Хабр URL: https://habr.com/ru/company/vk/blog/320962/ (дата обращения: 8.05.2022)

4. Веб-фреймворки для начинающих: простое объяснение с примерами // Tproger URL: https://tproger.ru/translations/web-frameworks-how-to-get-started/ (дата обращения: 8.05.2022)

5. Выбор лучшего Node.js фреймворка: Express, Koa или Sails // Umbrella IT URL: https://umbrellait.ru/blog/choosing-the-best-nodejs-framework/ (дата обращения: 9.05.2022)

6. Добавить поля в модель User в Django // Уроки по Django URL: https://djangosimple.blogspot.com/2014/05/user-django.html (дата обращения: 22.05.2022)

7. Запуск Django-приложения в Docker контейнере // PythonRU URL: https://pythonru.com/uroki/docker-django (дата обращения: 23.05.2022)

8. Зачем и как использовать контейнеры: разбираемся с Docker, Kubernetes и другими инструментами // Tproger URL: https://tproger.ru/articles/containers-explained/ (дата обращения: 10.05.2022)

9. Как правильно сделать сборку на Docker уже готового Django проекта со стандартной базой Sqlite? // stack overflow на русском URL: https://ru.stackoverflow.com/questions/1065775/Как-правильно-сделать-сборку-

на-docker-уже-готового-django-проекта-со-стандартно (дата обращения: 23.05.2022)

10. Как проектируют приложения: разбираемся в архитектуре // Skillbox Media URL: https://skillbox.ru/media/code/kak-proektiruyut-prilozheniya-razbiraemsya-v-arkhitekture/ (дата обращения: 19.05.2022)

11. Как работает WEB. Клиент-серверная модель и архитектура веб-приложения // Яндекс.Дзен URL: https://zen.yandex.ru/media/merion_networks/kak-rabotaet-web-klientservernaia-model-i-arhitektura-vebprilojeniia-604cc2f7126a3d455aa100dc (дата обращения: 8.05.2022)

12. Как разместить готовый сайт в Интернете // REG.RU URL: https://help.reg.ru/hc/ru/articles/4408047610513-Как-разместить-готовый-сайт-в-Интернете (дата обращения: 10.05.2022)

13. КЛАССИФИКАЦИЯ БИЗНЕС-ПРОЦЕССОВ ОРГАНИЗАЦИИ // Антон Пискун URL: https://www.antonpiskun.pro/klassifikacziya-biznes-proczessov-organizaczii/ (дата обращения: 8.05.2022)

14. Лекция 1: Основы функционирования веб-приложений // НОУ ИНТУИТ URL: https://intuit.ru/studies/courses/4455/712/lecture/21291?page=3 (дата обращения: 8.05.2022)

15. Лучшие фреймворки Node.js в 2021 году // proglib URL: https://proglib.io/p/luchshie-freymvorki-node-js-v-2021-godu-2021-04-14 (дата обращения: 9.05.2022)

16. Методы и средства моделирования бизнес-процессов: объектно-ориентированная методология: учеб.-метод. пособие / С. В. Рындина. – Пенза: Изд-во ПГУ, 2017. – 48 с.

17. Микросервисная архитектура: теория и практика // vc.ru URL: https://vc.ru/dev/295980-mikroservisnaya-arhitektura-teoriya-i-praktika (дата обращения: 8.05.2022)

18. Микрофреймворки— Ключевые аспекты веб-разработки на Python // ru.hexlet.io URL: https://ru.hexlet.io/courses/python-web-development/lessons/microframework/theory_unit (дата обращения: 9.05.2022)

19. Облачные серверы для разработчиков // VDS by Selectel URL: https://vds.selectel.ru/ru/ (дата обращения: 10.05.2022)

20. Охота на мифический MVC. Обзор, возвращение к первоисточникам и про то, как анализировать и выводить шаблоны самому // Хабр URL: https://habr.com/ru/post/321050/ (дата обращения: 19.05.2022)

21. Раджпут Динеш. Spring. Все паттерны проектирования. — СПб.: Питер, 2019. — 320 с. Перевели с английского Е. Иконникова, И. Пальти

22. Расширение модели пользователя в Django: сравнение нескольких стратегий с примерами кода // Tproger URL: https://tproger.ru/translations/extending-django-user-model/ (дата обращения: 22.05.2022)

23. Расширяем модель User в Django // apirobot.me URL: https://apirobot.me/posts/extending-user-model-in-django (дата обращения: 22.05.2022)

24. Регистраторы доменов. Рейтинг 2020-2021 года // HostingHUB.ru URL: https://hostinghub.ru/top/domain-registrator (дата обращения: 10.05.2022)

25. Регистрации и аутентификации пользователя на Django // python-scripts.com URL: https://python-scripts.com/user-accounts-django (дата обращения: 22.05.2022)

26. Регистрация на Express.js // ProgNote.ru URL: https://prognote.ru/web-dev/back-end/register-for-express-js/ (дата обращения: 9.05.2022)

27. Руководство Django Часть 8: Аутентификация и авторизация пользователя // MDN Web Docs URL: https://developer.mozilla.org/ru/docs/Learn/Server-side/Django/Authentication (дата обращения: 20.05.2022)

28. СБИС автоматизация салона красоты: Возможности // СБИС URL: https://sbis.ru/salons/opportunities (дата обращения: 19.05.2022)

29. ТОП-10 CRM-систем для салонов красоты: обзор и сравнение // Postium URL: https://postium.ru/crm-dlya-salonov-krasoty/ (дата обращения: 17.05.2022)

30. Топ-10 бэкенд-фреймворков для веб-разработки в 2022 году // medium.com URL: https://medium.com/nuances-of-programming/топ-10-бэкенд-фреймворков-для-веб-разработки-в-2022-году-f7d5172695dd (дата обращения: 9.05.2022)

31. Учебник Express часть 6: Работа с формами // MDN Web Docs URL: https://developer.mozilla.org/ru/docs/Learn/Server-side/Express_Nodejs/forms (дата обращения: 9.05.2022)

32. Фреймворк // Википедия URL: https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA (дата обращения: 9.05.2022)

33. Хостинг: варианты, сравнения, пользовательская статистика // Хабр URL: https://habr.com/ru/company/ruvds/blog/443522/ (дата обращения: 10.05.2022)

34. Хостинг: что это, зачем и как выбрать // vc.ru URL: https://vc.ru/services/74241-hosting-chto-eto-zachem-i-kak-vybrat (дата обращения: 10.05.2022)

35. Чем Джанго лучше/хуже Фласка? // devman URL: https://dvmn.org/encyclopedia/qna/18/chem-dzhango-luchshehuzhe-flaska/ (дата обращения: 9.05.2022)

36. Что такое база данных? // Oracle URL: https://www.oracle.com/cis/database/what-is-database/ (дата обращения: 8.05.2022)

37. Beget - платный хостинг // beget URL: https://beget.com/ru/ (дата обращения: 10.05.2022)

38. Comparing express vs. meteor vs. node-rest-client vs. restify vs. sails // NMPCompare URL: https://npmcompare.com/compare/express,meteor,node-rest-client,restify,sails (дата обращения: 9.05.2022)

39. Django vs Flask: что выбрать для своего проекта? // Pythonist URL: https://pythonist.ru/django-vs-flask-chto-vybrat-dlya-svoego-proekta/ (дата обращения: 9.05.2022)

40. Docker Documantation // docker docs URL: https://docs.docker.com/ (дата обращения: 23.05.2022)

41. Documentation // django URL: https://docs.djangoproject.com/en/4.0/ (дата обращения: 23.05.2022)

42. Documentation // sails URL: https://sailsjs.com/documentation/reference (дата обращения: 10.05.2022)

43. Express - Node.js web application framework // Express URL: https://expressjs.com/ (дата обращения: 10.05.2022)

44. Git для новичков (часть 1) // Хабр URL: https://habr.com/ru/post/541258/ (дата обращения: 22.05.2022)

45. How to choose the best Node.js framework: Express.js, Koa.js or Sails.js // Cleveroad URL: https://www.cleveroad.com/blog/the-best-node-js-framework-for-your-project--express-js--koa-js-or-sails-js (дата обращения: 9.05.2022)

46. Model-View-Controller // Википедия URL: https://ru.wikipedia.org/wiki/Model-View-Controller (дата обращения: 8.05.2022)

47. ORM // Википедия URL: https://ru.wikipedia.org/wiki/ORM (дата обращения: 9.05.2022)

48. REG.RU - Домашняя // REG.RU URL: https://www.reg.ru/ (дата обращения: 10.05.2022)

49. Sails.js // Википедия URL: https://en.wikipedia.org/wiki/Sails.js (дата обращения: 9.05.2022)

50. Timeweb - хостинг для сайтов и регистрация доменов // timeweb> URL: https://timeweb.com/ru/ (дата обращения: 10.05.2022)

51. Using the Django authentication system // django URL: https://docs.djangoproject.com/en/4.0/topics/auth/default/ (дата обращения: 23.05.2022)

52. Welcome to Flask — Flask Documentation (2.1.x) // flask.palletsprojects.com URL: https://flask.palletsprojects.com/en/2.1.x/ (дата обращения: 10.05.2022)

ПРИЛОЖЕНИЕ А

1. Общие сведения

1.1. Название системы

Полное название разрабатываемой — система управления бизнес-процессами «BeautyManage».

Сокращенное название — СУБП «BM»

1.2. Назначение системы

Система предназначена для автоматизации управления внешними и внутренними бизнес-процессами в салоне красоты.

1.3. Организации, участвующие в разработке

Система является частным заказом индивидуального предпринимателя физическому лицу. Регулирование отношений происходит в соответствии с гражданско- правовым договором.

Заказчик – ИП Логинова Софья Андреевна.

Исполнитель – Елкина Галина Александровна.

1.4. Плановые сроки начала и окончания работы по созданию системы

Сроки начала и окончания работ определяются после передачи данного Технического Задания Исполнителю.

Дата начала работ: не позднее 7 дней с момента передачи Технического Задания Исполнителю.

Дата окончания работ: не позднее 1 июня 2022 года

2. Требования к системе

2.1. Требования к функционалу

Требуется разработать веб-базированную систему, в которой будут реализованы следующие элементы и функции:

- вход пользователя-администратора в личный кабинет по уникальной связке логин-пароль;
- подключение по уникальной связке логин-пароль мастеров салона;
- автоматическая запись клиентов по номеру телефона;
- календарь записи клиентов с доступом к календарю каждому мастеру;
- календарь смен мастеров;
- систему учета материалов;
- напоминания о необходимости оплаты счетов;
- напоминания о необходимости заказа новых расходных материалов (в автоматическом режиме, с формированием списка закончившихся позиций);
- систему оценки персонала (опоздания, переработки, премии и депремирование).
- система финансовой регулировки.

Также должно быть реализовано два или более типов учетных записей:

- кабинет администратора: с полным доступом к системе и ее функциям;
- кабинет мастера: с ограниченным функционалом;
- кабинет клиента: с доступом только к системе записи и истории процедур.

Таблица 1. Функционал каждой категории пользователей

Тип пользователя	**Функционал кабинета**
Администратор	Добавление и удаление мастеров. Внесение в черный список клиентов. Внесение изменений в смены мастеров (удаление, редактирование, добавление). Внесение изменений в календарь записей (удаление, редактирование, добавление). Внесение изменений в журнал закупок (удаление, редактирование, добавление). Модерация контента (удаление, редактирование, добавление).

	Доступ к системе оценки персонала (удаление, редактирование, добавление). Доступ к системе кассовой выручки (удаление, редактирование, добавление). Доступ к базе клиентов (удаление, редактирование, добавление). Добавление администраторов и генерирование связки логин-пароль
Мастер	Запросы на изменение смен. Система фиксирования времени пребывания на рабочем месте (пришел на смену *время*, открыл смену *время*, закрыл смену *время*, комментарии). Доступ к календарю записей клиентов (без указания номера клиента) в режиме просмотра. Доступ к системе кассовой выручки (сколько было средств в начале смены, сколько в конце). Доступ к журналу закупок (фиксирование количества материала).
Клиент	Доступ к системе записи (по номеру телефона). Доступ к истории услуг (какие услуги были оказаны, когда, стоимость).

Исполнитель должен предоставить схему реализации с датами и версиями. Единовременная реализация продукта не требуется. Первостепенной реализации требуют личные кабинеты мастеров и администраторов с доступом в календарь смен и календарь записей.

2.2. Требования к дизайну

Основным требованием по эргономике является комфортность и интуитивная понятность интерфейса, унифицированное расположение основных функциональных кнопок, единообразие и соответствие общесистемным соглашениям, используемых диалогов, сообщений и экранов помощи. При выполнении стандартных запросов пользователь должен работать с системой в реальном масштабе времени.

Разрабатываемая Система должна иметь графический пользовательский интерфейс. Диалог с пользователем должен быть оптимизирован для выполнения типовых операций по формированию запросов и вводу в БД системы соответствующей информации.

Необходимо предусмотреть отображение на экране информации о ходе длительных процессов обработки.

При обнаружении Системой каких–либо ошибок в действиях пользователя должно выдаваться сообщение с пояснениями, достаточными для исправления ошибки.

Взаимодействие пользователей с Системой должно осуществляться на русском языке.

Все элементы управления, выполняющие одинаковые функции, должны называться одинаково.

2.3. Требования к безопасности

Необходимо обеспечить блокировку доступа к базе клиентов со стороны мастера и клиента. Доступ к базе клиентов и системе оценивания персонала должен быть исключительно у администратора.

Первичный вход в систему должен осуществляться путем ввода уникального логина-пароля, выдаваемого конкретному списку администраторам разработчиком (Исполнителем).

Первичный вход в систему мастеров должен осуществляться через уникальные логины-пароли, генерируемые администраторами.

ПРИЛОЖЕНИЕ Б

Диаграмма последовательности «Обработка заявки клиента»

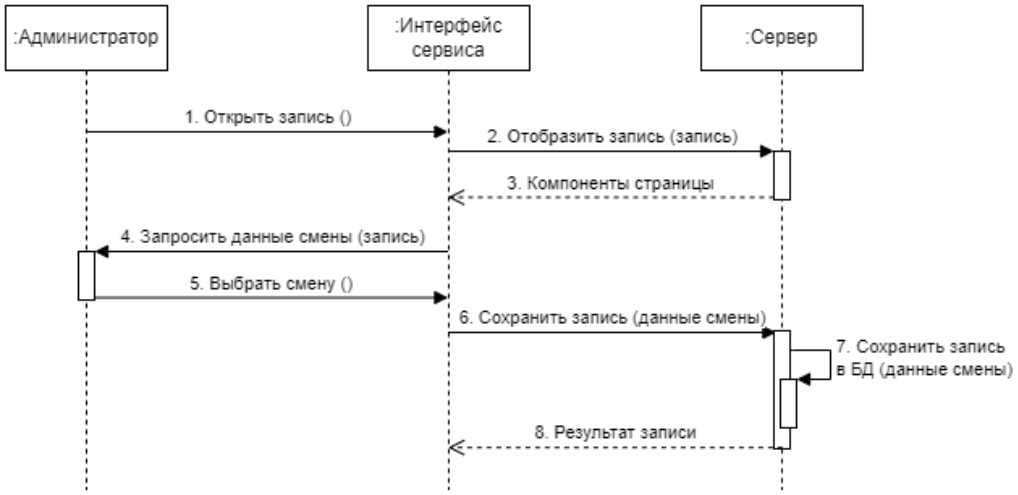


Диаграмма последовательности «Регистрация (завершение) проведения услуги»

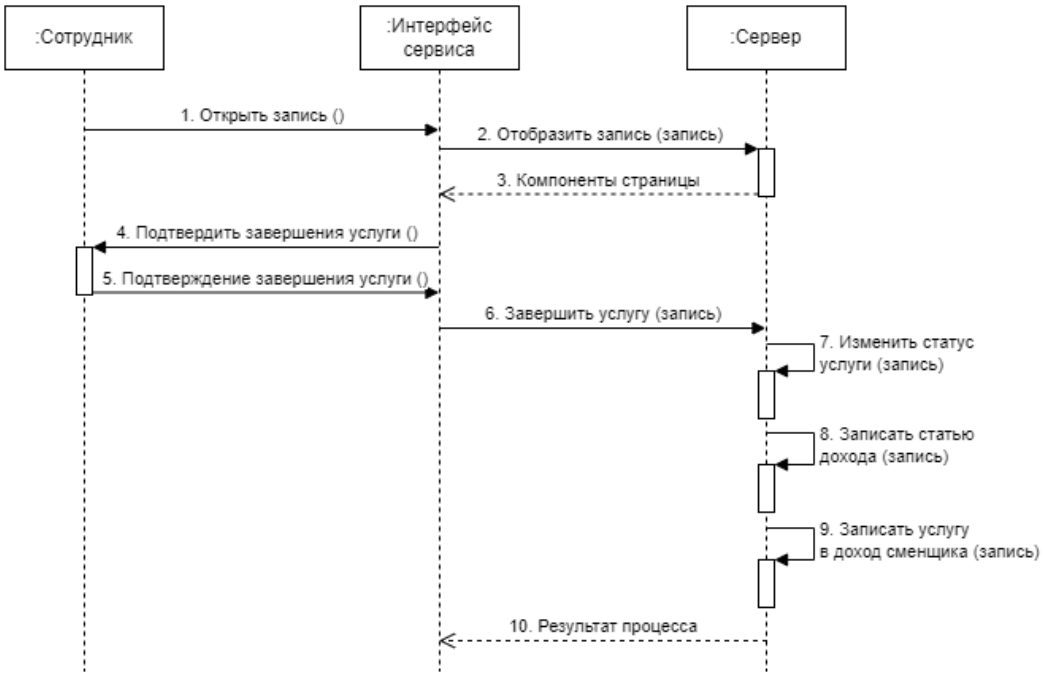


Диаграмма последовательности «Отправление заявки на смену»

картинка

Диаграмма последовательности «Обработка заявки на смену»

картинка

Диаграмма последовательности «Регистрация клиента»

картинка

Диаграмма последовательности «Процесс приема на работу»

картинка

Диаграмма последовательности «Управление списком услуг»

картинка

ПРИЛОЖЕНИЕ В

макет 1

макет 2

макет 3

макет 4

макет 5

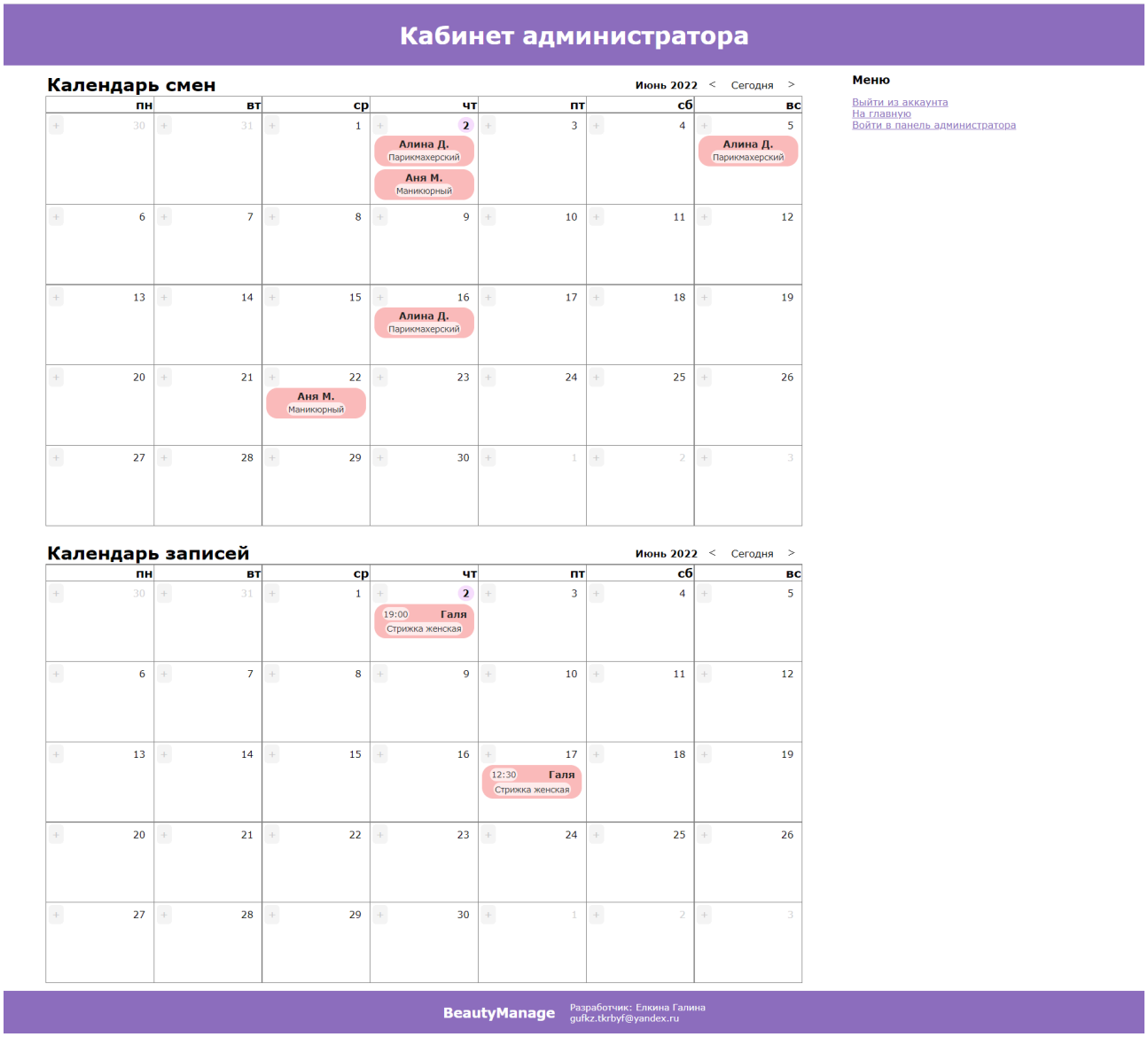
ПРИЛОЖЕНИЕ Г

QR-код на Git-репозиторий с полным кодом приложения:

Ссылка на тот же Git-репозиторий: https://github.com/iamgo100/diploma

ПРИЛОЖЕНИЕ Д

Страница личного кабинета администратора



Страница личного кабинета сотрудника

Страница личного кабинета клиента