

Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,
МЕХАНИКИ И ОПТИКИ”

Факультет ИКТ

Образовательная программа 45.03.04 - Интеллектуальные системы в гуманитарной сфере

Направление подготовки (специальность) 45.03.04 - Интеллектуальные системы в гуманитарной сфере

О Т Ч Е Т

по курсовой работе

Тема задания: Реализация web-сервисов средствами Django REST framework, Vue.js, Muse-UI

Обучающийся Андреева Е.А. К3343

Руководитель: Говоров А.И.

Оценка _____

Дата _____

Санкт-Петербург
20 20

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ.....	4
1.1. Описание предметной области	4
1.2. Описание функциональных требований.....	4
2. ОПИСАНИЕ СЕРВЕРНОЙ ЧАСТИ.....	6
2.1. Средства разработки серверной части	6
2.2. Разработка модели данных и моделей Django	6
2.3. Сериализация моделей	7
2.4. Создание отображений.....	7
2.5. Интерфейсы панели Django REST	8
3. ОПИСАНИЕ КЛИЕНТСКОЙ ЧАСТИ.....	12
3.1. Средства разработки клиентской части.....	12
3.2. Интерфейсы Vue	12
4. КОНТЕЙНЕРИЗАЦИЯ И ОРКЕСТРАЦИЯ WEB-ПРИЛОЖЕНИЯ.....	19
ЗАКЛЮЧЕНИЕ.....	20
СПИСОК ЛИТЕРАТУРЫ	21
ПРИЛОЖЕНИЯ	22

ВВЕДЕНИЕ

В рамках курсовой работы по дисциплине «Основы web-программирования» было предложено создать программную систему, предназначенную для администратора гостиницы. Такая система должна обеспечивать хранение сведений об имеющихся в гостинице номерах, о проживающих в гостинице клиентах и о служащих, убирающихся в номерах. Количество номеров в гостинице известно, и имеются номера трех типов: одноместный, двухместный и трехместный, отличающиеся стоимостью проживания в сутки. В каждом номере есть телефон. О каждом проживающем должна храниться следующая информация: номер паспорта, фамилия, имя, отчество, город, из которого он прибыл, дата поселения в гостинице, выделенный гостиничный номер. О служащих гостиницы должна храниться информация следующего содержания: фамилия, имя, отчество, где (этаж) и когда (день недели) он убирает. Служащий гостиницы убирает все номера на одном этаже в определенные дни недели, при этом в разные дни он может убирать разные этажи.

Работа с системой предполагает получение следующей информации:

- о клиентах, проживавших в заданном номере, в заданный период времени;
- о количестве клиентов, прибывших из заданного города,
- о том, кто из служащих убирал номер указанного клиента в заданный день недели,
- сколько в гостинице свободных номеров;
- список клиентов с указанием места жительства, которые проживали в те же дни, что и заданный клиент, в определенный период времени.

Администратор должен иметь возможность выполнить следующие операции:

- принять на работу или уволить служащего гостиницы;
- изменить расписание работы служащего;
- поселить или выселить клиента.

В стеке технологий для разработки web-приложения должны быть использованы: web-фреймворк Django языка программирования Python, web-фреймворк Vue.js языка программирования JavaScript и библиотека Muse-UI для создания пользовательского интерфейса.

1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

1.1. Описание предметной области

Предметной областью данной курсовой работы является администрирование гостиницы. Главным пользователем разрабатываемого сервиса является администратор гостиницы, в основные задачи которого входит:

- Заселять и выселять проживающих в номера
- Принимать на работу и увольнять служащих
- Составлять график уборки номеров

Помимо этого, администратору гостиницы может понадобиться следующая информация:

- о клиентах, проживавших в заданном номере, в заданный период времени;
- о количестве клиентов, прибывших из заданного города,
- о том, кто из служащих убирал номер указанного клиента в заданный день недели,
- сколько в гостинице свободных номеров;
- список клиентов с указанием места жительства, которые проживали в те же дни, что и заданный клиент, в определенный период времени.

Система должна учитывать, что в гостинице имеются разные типы номеров с разной посуточной оплатой на разных этажах. Разрабатываемый сервис должен хранить в базе данных информацию о номерах, о проживающих, о служащих и о графике уборке.

1.2. Описание функциональных требований

По описанию предметной области были составлены функциональные требования к разрабатываемому web-приложению:

- Реализовать функции добавления
 - Регистрация нового проживающего
 - Принятие на работу нового служащего
 - Составление расписания
- Реализовать функции просмотра
 - Информация о проживающих
 - Информация о служащих гостиницы
 - Информация о графике уборки
 - Информация об имеющихся в гостинице номерах

- Реализовать функции редактирования
 - Изменение информации о проживающем
 - Изменение информации о служащем
 - Изменение графика уборки
- Реализовать функции удаления
 - Увольнение служащего
 - Удаление проживающего
- Реализовать регистрацию и авторизацию
- Реализовать функции, возвращающие результаты выполнения запросов

2. ОПИСАНИЕ СЕРВЕРНОЙ ЧАСТИ

2.1. Средства разработки серверной части

Для реализации серверной части web-приложения были использованы следующие средства разработки:

- PostgreSQL – свободная объектно-реляционная система управления базами данных. Использование данной технологии позволяет создавать и хранить базу данных на сервере, что способствует гибкой передаче web-приложения на стадию производства.
- web-фреймворк Django REST языка программирования Python, основанный на технологии REST (сокр. англ. Representational State Transfer, «передача состояния представления») — стиль построения архитектуры распределенного приложения. Данные в REST должны передаваться в виде небольшого количества стандартных форматов (например HTML, XML, JSON). Сетевой протокол, как и HTTP, должен поддерживать кэширование, не должен зависеть от сетевого слоя, не должен сохранять информацию о состоянии между парами «запрос-ответ». Утверждается, что такой подход обеспечивает масштабируемость системы и позволяет ей эволюционировать с новыми требованиями.

2.2. Разработка модели данных и моделей Django

В соответствии с описанием предметной области и выявленными функциональными требованиями была разработана модель базы данных, представленная на рисунке 1.

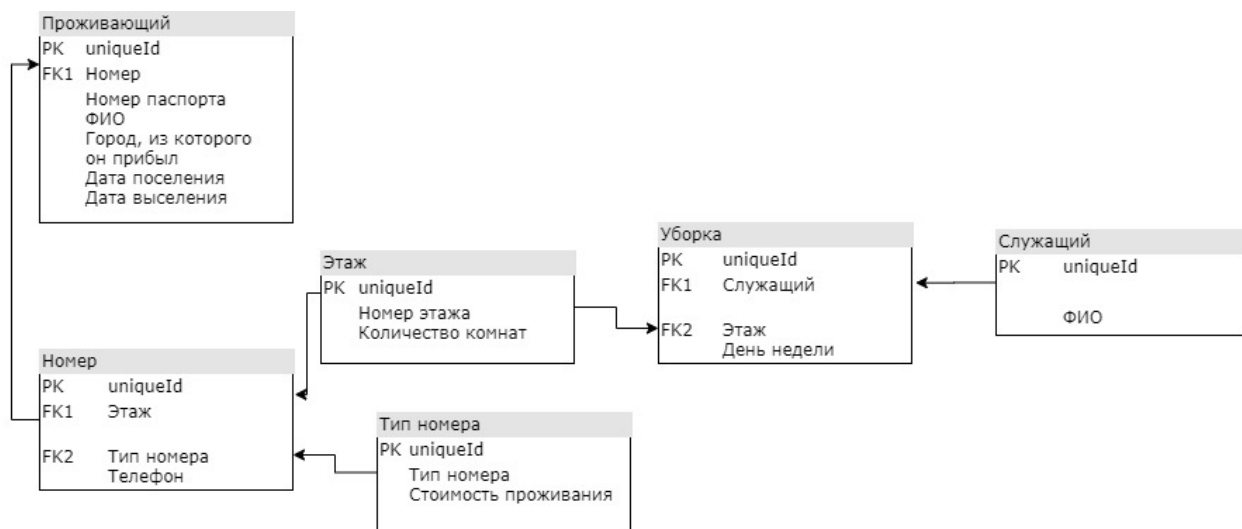


Рисунок 1 – модель базы данных

Спроектированная модель данных была реализована в качестве классов моделей Django в файле `models.py`, листинг кода которого представлен в приложении 1. Были созданы следующие классы моделей:

- Class Floor – Этаж. Данная модель содержит информацию о номере этажа и количестве комнат на этом этаже
- Class RoomType – Тип комнаты. Данная модель содержит информацию о стоимости комнаты определенного типа (одноместная, двухместная, трехместная)
- Class Room – Комната. Данная модель описывает номер комнаты, на каком этаже она находится и номер телефона.
- Class Resident – Проживающий. Данная модель содержит информацию о проживающем: имя, фамилия, отчество, номер паспорта, откуда он прибыл, в какую комнату поселен и на какой срок.
- Class Servant – Служащий. Данная модель хранит данные о служащем: фамилию, имя, отчество.
- Class Cleaning – Уборка. Данная модель содержит информацию о том, какой служащий убирает комнаты на каком этаже и в какой день недели.

2.3.Сериализация моделей

Далее были созданы сериализаторы для обеспечения обмена данными между серверной частью, написанной на Django REST Framework, и клиентской частью, написанной на Vue.js. Листинг кода файла `serializers.py`, в котором описаны все созданные сериализаторы, представлен в приложении 2.

2.4.Создание отображений

Для создания отображений использовался класс `ViewSet`, который обладает встроенными атрибутами для последующего создания функций CRUD для модели. Для создания отображений для выполнения запросов использовался класс `APIView`. Листинг кода со всеми реализованными отображениями представлен в приложении 3. Согласно описанной предметной области и выявленным функциональным требованиям, были созданы следующие отображения:

- `class FloorViewSet` – отображение для модели Этаж. В качестве класса сериализатора использовался `FloorSerializer`.

- class RoomTypeViewSet – отображение для модели Тип комнаты. В качестве класса сериализатора использовался RoomTypeSerializer.
- class RoomViewSet – отображение для модели Комната. В качестве класса сериализатора использовался RoomSerializer.
- class ResidentViewSet – отображение для модели Проживающий. В качестве класса сериализатора при вызове метода «создать» использовался ResidentCreateSerializer. При вызове остальных методов использовался ResidentSerializer.
- class ServantViewSet – отображение для модели Служащий. В качестве класса сериализатора использовался ServantSerializer.
- class CleaningViewSet – отображение для модели Уборка. В качестве класса сериализатора при вызове метода «создать» использовался CleaningCreateSerializer. При вызове остальных методов использовался CleaningSerializer.
- class Query1 – отображение для вывода результата выполнения запроса 1 «о клиентах, проживавших в заданном номере».
- class Query2 – отображение для вывода результата выполнения запроса 2 «о том, кто из служащих убирал номер указанного клиента в заданный день недели».
- class Query3 – отображение для вывода результата выполнения запроса 3 «о количестве клиентов, прибывших из заданного города».
- class Query4 – отображение для вывода результата выполнения запроса 1 «сколько в гостинице свободных номеров».

2.5.Интерфейсы панели Django REST

Результаты разработки серверной части можно проверить в панели Django REST:

а. Этажи

Выводятся данные обо всех этажах, а также о конатах, которые на них находятся. Скриншот представлен на рисунке 2.

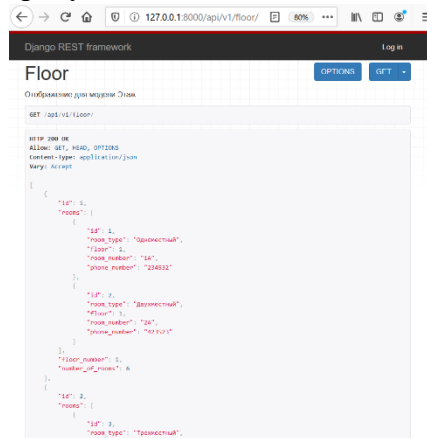


Рисунок 2 – Этажи в Django REST

б. Комната

Вывод заданной комнаты с подробной информацией о ней. Скриншот представлен на рисунке 3

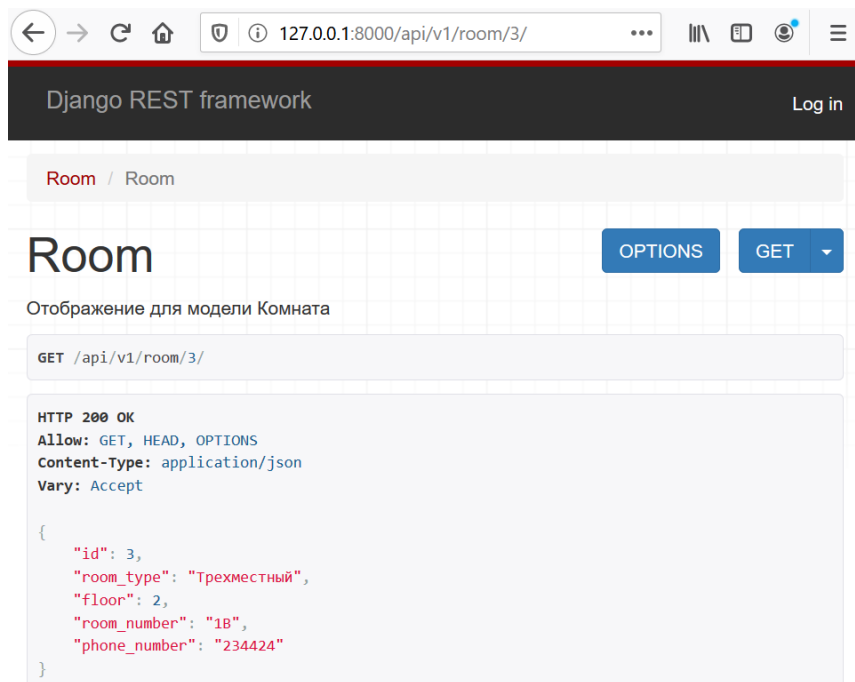


Рисунок 3 – заданная Комната в Django REST

с. Типы комнат

Выводится информация о типе комнат в гостинице. Скриншот представлен на рисунке 4

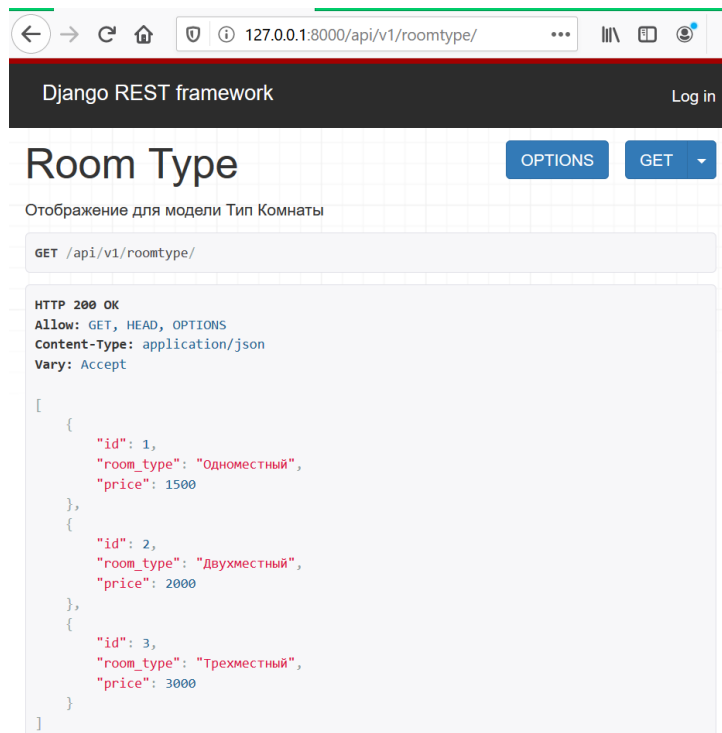


Рисунок 4 – Типы комнат в Django REST

d. Служащие

Вывод информации обо всех служащих. Скриншот представлен на рисунке 5

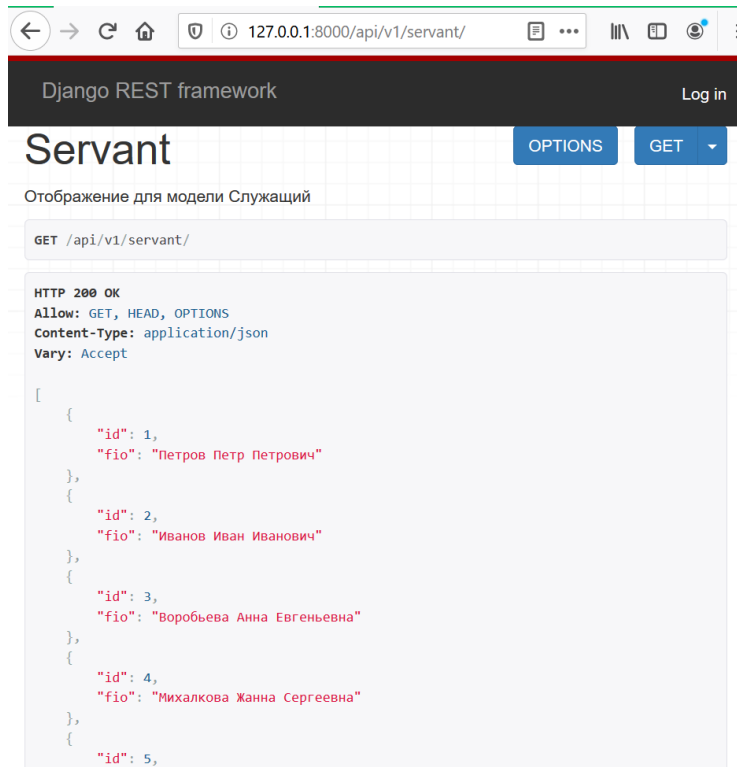


Рисунок 5 – Служащие в Django REST

е. Уборка

Добавление новой записи в модель Уборка. Скриншот представлен на рисунке 6

Django REST framework Log in

Cleaning / Cleaning

Cleaning

Отображение для модели Уборка

GET /api/v1/cleaning/create

HTTP 405 Method Not Allowed
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

```
{
  "detail": "Метод \"GET\" не разрешен."
}
```

Raw data HTML form

День недели: Четверг

Servant: Петров Петр Петрович

Floor: Floor object (1)

POST

Рисунок 6 – добавление новой записи в Django REST

ф. Проживающий

Добавление нового проживающего. Скриншот представлен на рисунке 7

Raw data HTML form

ФИО проживающего: Зеленский Владимир Александрович

Серия и номер паспорта: 6876873672

Прибыл из города: Кривой Рог

Дата заселения: 01.06.2020

Дата выселения: 18.06.2020

Room: Номер 1В

POST

Рисунок 7 – добавление Проживающего в Django REST

3. ОПИСАНИЕ КЛИЕНТСКОЙ ЧАСТИ

3.1. Средства разработки клиентской части

Клиентская часть была реализована с помощью web-фреймворка Vue.js языка программирования JavaScript и его библиотеки MUSE-UI для дизайна пользовательского интерфейса, основанной на Material Design.

3.2. Интерфейсы Vue

Пользовательские интерфейсы, созданные на базе Vue.js и MUSE-UI представлены ниже:

а. Стартовая страница

Стартовая страница web-сервиса с верхним меню навигации и описанием варианта лабораторной работы. Скриншот представлен на рисунке 8.

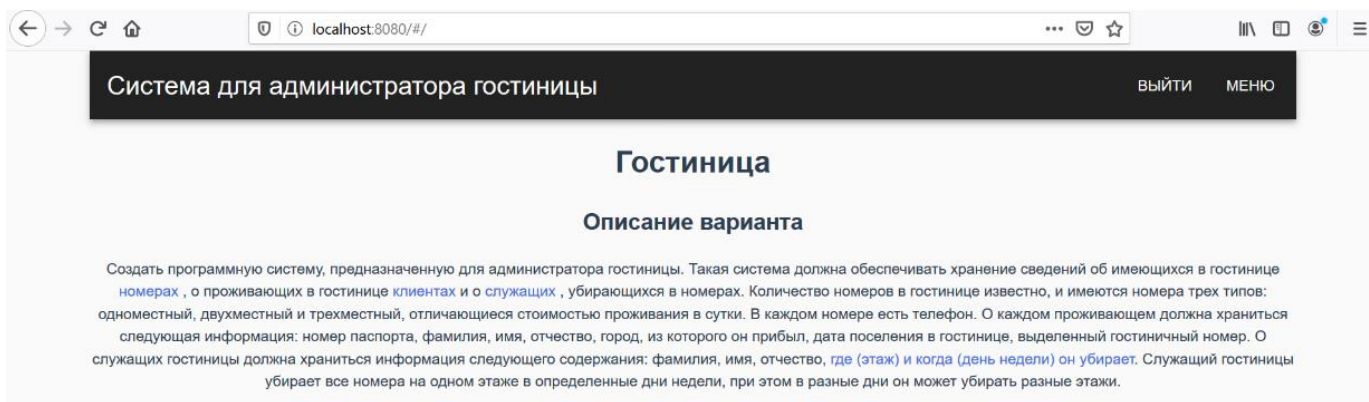


Рисунок 8 – Стартовая страница web-сервиса

б. Вход

Страница авторизации пользователя имеет форму входа, а также ссылку на страницу регистрации. Скриншот представлен на рисунке 9

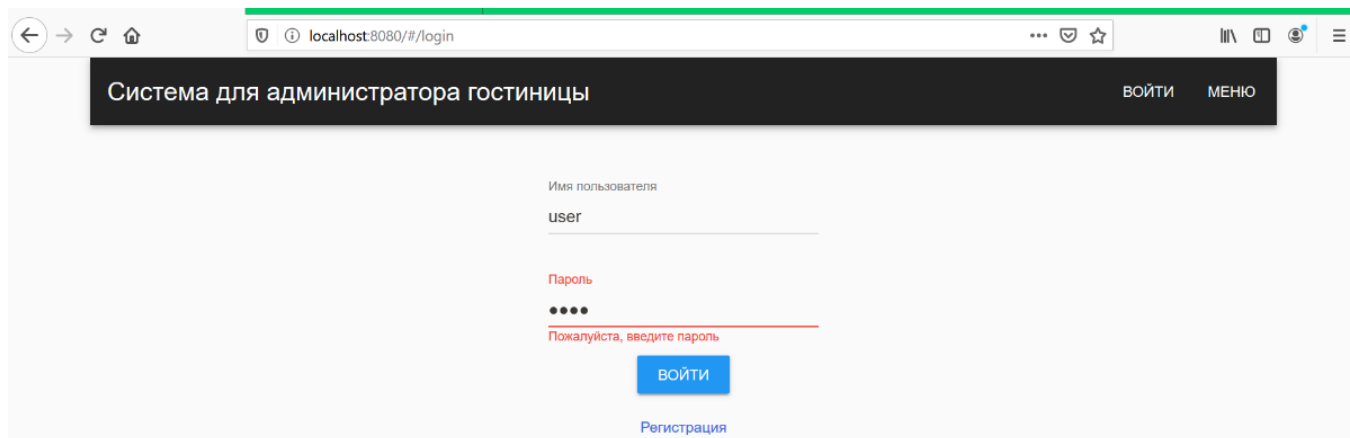


Рисунок 9 – Страница входа

с. Регистрация новых пользователей

Страница регистрации нового пользователя представляет собой форму, которую необходимо заполнить для создания нового пользователя. В случае правильного заполнения всех полей и отсутствия пользователя с таким же username, пользователь будет зарегистрирован и перенаправлен на главную страницу. Скриншот представлен на рисунке 10

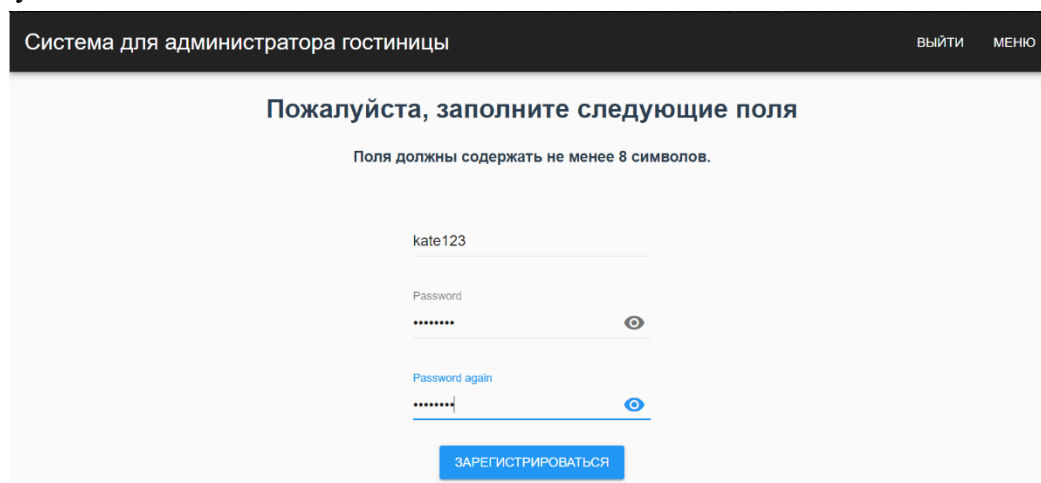


Рисунок 10 – Страница регистрации нового пользователя

d. Просмотр номеров гостиницы

На данной странице показывается информация о номерах. При нажатии на кнопку «Показать все номера» появляется список номеров, которые находятся в данной гостинице, при нажатии на кнопку «Показать информацию о номерах» появляется подробная информация о номерах. Скриншот представлен на рисунке 11

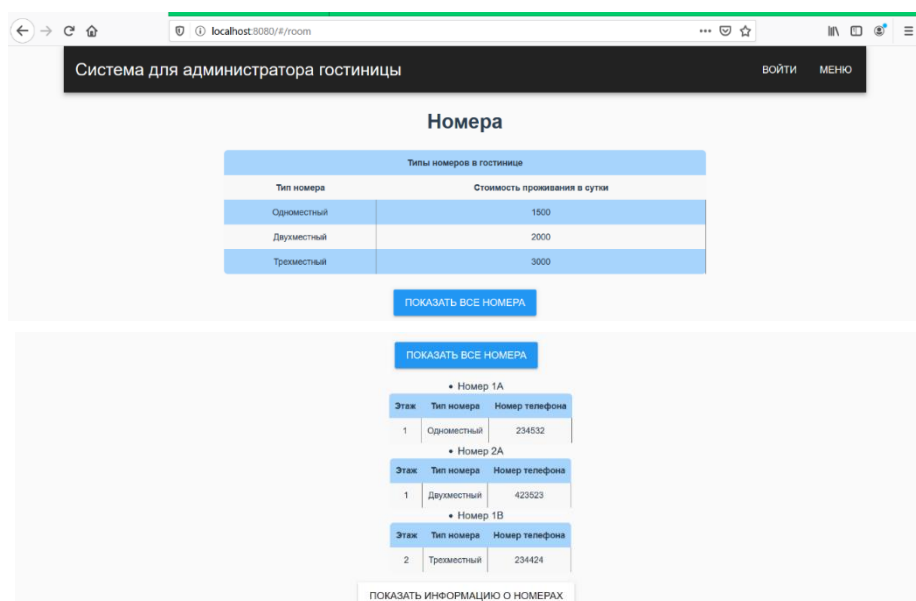


Рисунок 11 – Страница «Номера»

е. Просмотр всех проживающих

Страница со списком проживающих. При нажатии на кнопку «Добавить проживающего» появляется форма для создания нового проживающего, при нажатии на кнопку «Удалить проживающего» - форма удаления проживающего. Скриншот представлен на рисунке 12

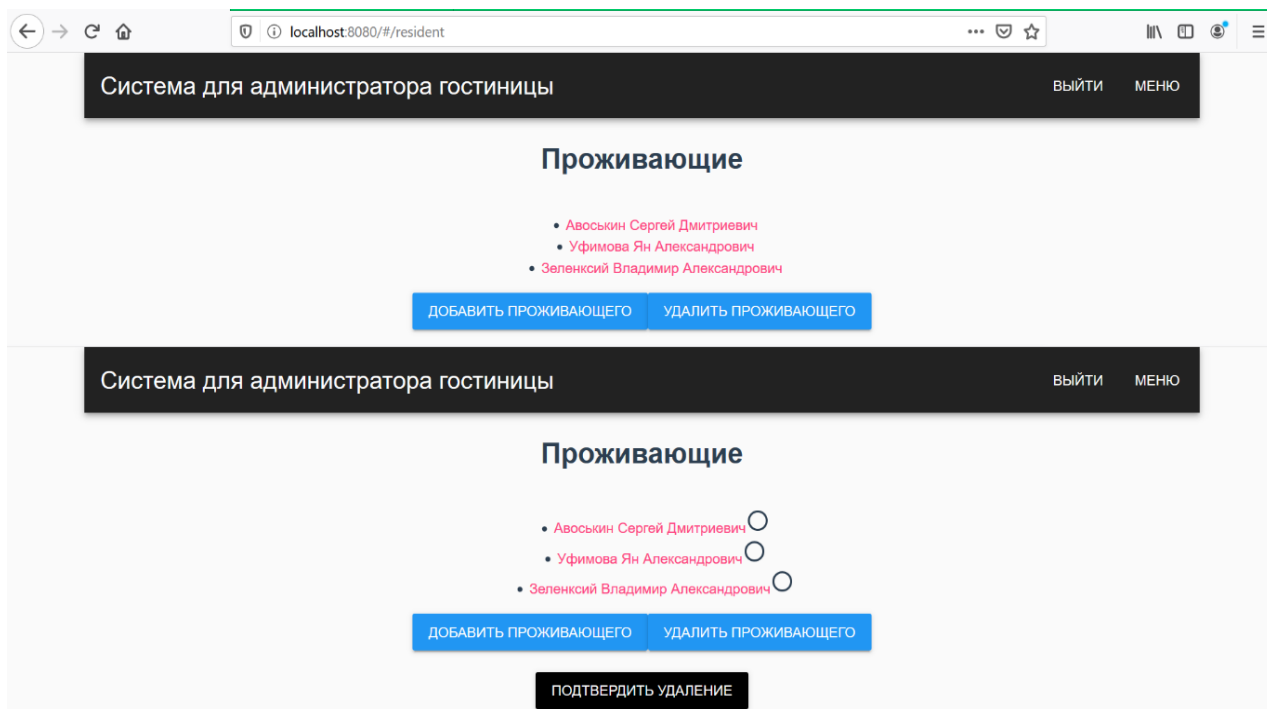


Рисунок 12 – Страница «Проживающие»

ф. Просмотр выбранного проживающего

Страница с подробной информацией о проживающем. При нажатии на кнопку «Изменить данные» - появляется форма для изменения информации о проживающем, при нажатии на «Удалить проживающего» - данный проживающий удаляется и происходит перенаправление на страницу «Проживающие». Скриншот представлен на рисунке 13.

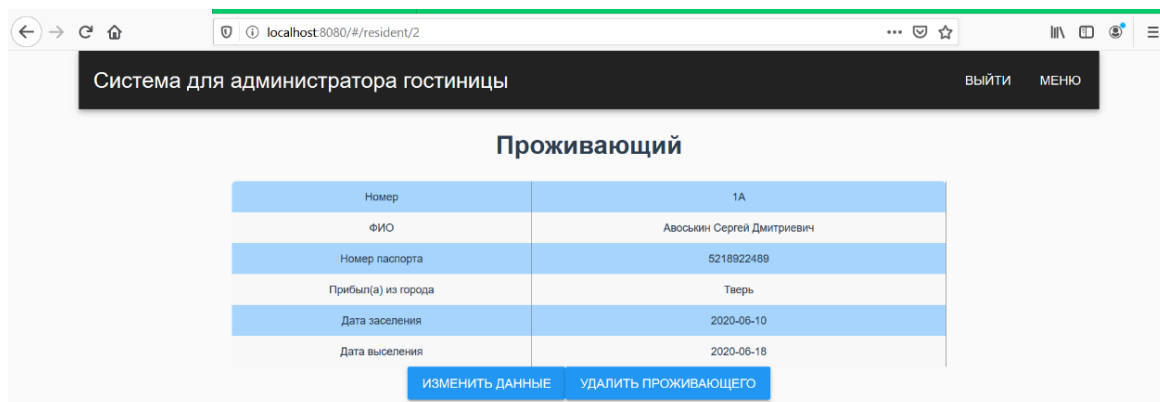


Рисунок 13 – Страница «Проживающий»

g. Просмотр всех служащих гостиницы

Страница со списком служащих, при нажатии на каждую запись происходит перенаправление на личную страницу служащего. Имеются также кнопки добавления и удаления служащего. Скриншот представлен на рисунке 14.

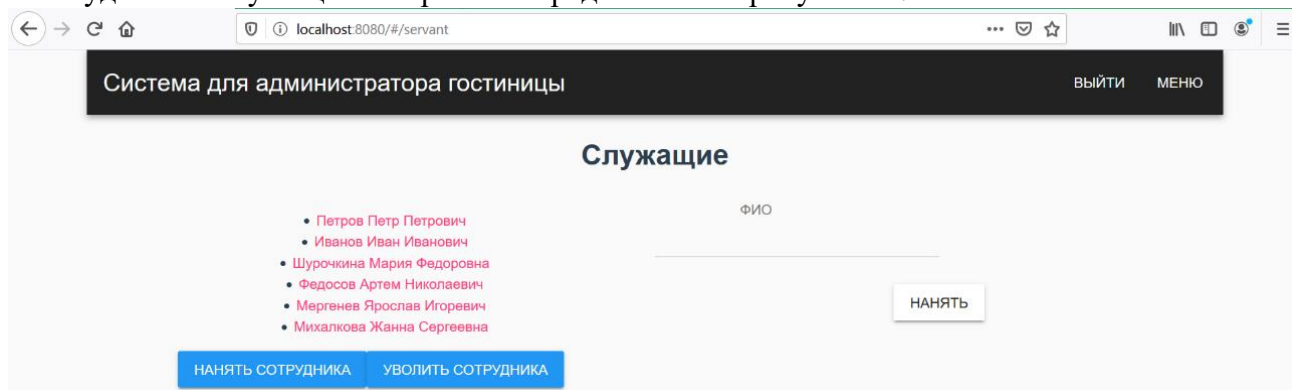


Рисунок 14 – Страница со всеми служащими гостиницы

h. Просмотр графика уборки

Страница с графиком уборки номеров. Кнопки «Добавить запись», «Изменить запись», «Удалить запись» реализуют соответствующие функции. Скриншот представлен на рисунке 15

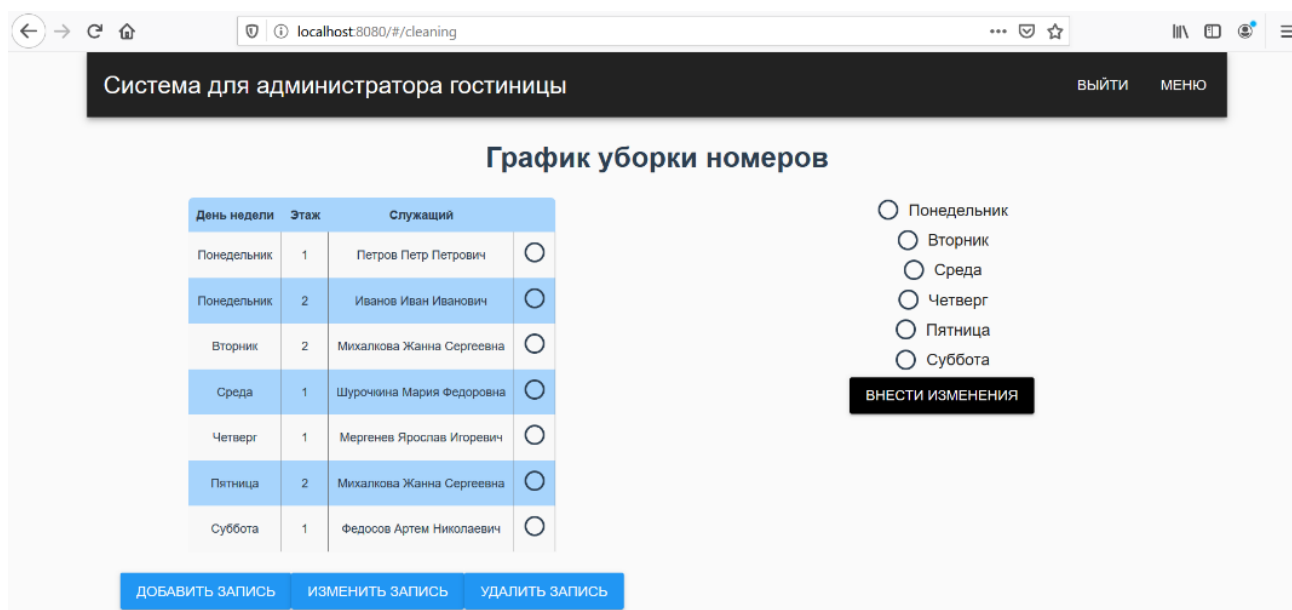


Рисунок 15 – Страница с графиком уборки номеров

i. Вход

Страница входа включает в себя функцию авторизации пользователя. В случае правильного ввода имени пользователя и пароля пользователь получит уведомление, после чего будет перенаправлен на главную страницу web-сервиса. Скриншот представлен на рисунке 16.

Система для администратора гостиницы

ВОЙТИ МЕНЮ

Имя пользователя

Пароль
Пожалуйста, введите пароль

ВОЙТИ

Регистрация

Рисунок 16 – Вход

j. Регистрация

Страница регистрации имеет поля для ввода имени нового пользователя и пароля. При правильном вводе данных новый пользователь добавляется в базу данных и происходит перенаправление на страницу входа, если данные введены неверно, то появляется соответствующее предупреждение. Скриншот представлен на рисунке 17.

Система для администратора гостиницы

ВОЙТИ МЕНЮ

Пожалуйста, заполните следующие поля

Поля должны содержать не менее 8 символов.

Username

Password

Password again

ЗАРЕГИСТРИРОВАТЬСЯ

Рисунок 17–Регистрация

k. Результат выполнения запроса 3 представлен на рисунке 18

Запрос №3

о том, кто из служащих убирал номер указанного клиента в заданный день недели

Клиент

Авоськин Сергей Дмитриевич

День недели

☐ Понедельник ☐ Вторник ☒ Среда ☐ Четверг

☐ Пятница ☐ Суббота

ВЫПОЛНИТЬ ЗАПРОС

Результат выполнения запроса

Шурочкина Мария Федоровна

Рисунок 18–Запрос 3

1. Фильтрация графика уборки

Функция фильтрации является расширением интерфейса «График уборки». При нажатии на кнопку «Показать фильтры» появляются поля для выбора способа фильтрации: по сл, по автору и по шифру. Скриншот работы фильтров представлен на рисунках 19, 20 и 21.

The screenshot shows the 'График уборки номеров' (Cleaning Schedule of Rooms) interface. At the top, there is a button 'ПОКАЗАТЬ ФИЛЬТРЫ' (Show Filters). Below it, there are three filter dropdowns: 'Фильтр по служащему' (Filter by employee) set to 'Михалкова Жанна Сергеевна', 'Фильтр по этажу' (Filter by floor), and 'Фильтр по дню недели' (Filter by day of the week). Below the filters is a button 'ОТФИЛЬТРОВАТЬ' (Filter). Underneath is a table with the following data:

День недели	Этаж	Служащий
Вторник	2	Михалкова Жанна Сергеевна
Пятница	2	Михалкова Жанна Сергеевна

At the bottom of the interface are three buttons: 'ДОБАВИТЬ ЗАПИСЬ' (Add Record), 'ИЗМЕНИТЬ ЗАПИСЬ' (Edit Record), and 'УДАЛИТЬ ЗАПИСЬ' (Delete Record).

Рисунок 19—фильтрация по служащему

The screenshot shows the 'График уборки номеров' (Cleaning Schedule of Rooms) interface. At the top, there is a button 'ПОКАЗАТЬ ФИЛЬТРЫ' (Show Filters). Below it, there are three filter dropdowns: 'Фильтр по служащему' (Filter by employee), 'Фильтр по этажу' (Filter by floor) set to '1', and 'Фильтр по дню недели' (Filter by day of the week). Below the filters is a button 'ОТФИЛЬТРОВАТЬ' (Filter). Underneath is a table with the following data:

День недели	Этаж	Служащий
Понедельник	1	Петров Петр Петрович
Среда	1	Шурочкина Мария Федоровна
Четверг	1	Мергенов Ярослав Игоревич
Суббота	1	Федосов Артем Николаевич

At the bottom of the interface are three buttons: 'ДОБАВИТЬ ЗАПИСЬ' (Add Record), 'ИЗМЕНИТЬ ЗАПИСЬ' (Edit Record), and 'УДАЛИТЬ ЗАПИСЬ' (Delete Record).

Рисунок 20—фильтрация по этажу

График уборки номеров

ПОКАЗАТЬ ФИЛЬТРЫ

Фильтр по служащему

Фильтр по этажу

Фильтр по дню недели

▼ Понедельник ▼

ОТФИЛЬТРОВАТЬ

День недели	Этаж	Служащий
Понедельник	1	Петров Петр Петрович
Понедельник	2	Иванов Иван Иванович

ДОБАВИТЬ ЗАПИСЬ

ИЗМЕНИТЬ ЗАПИСЬ

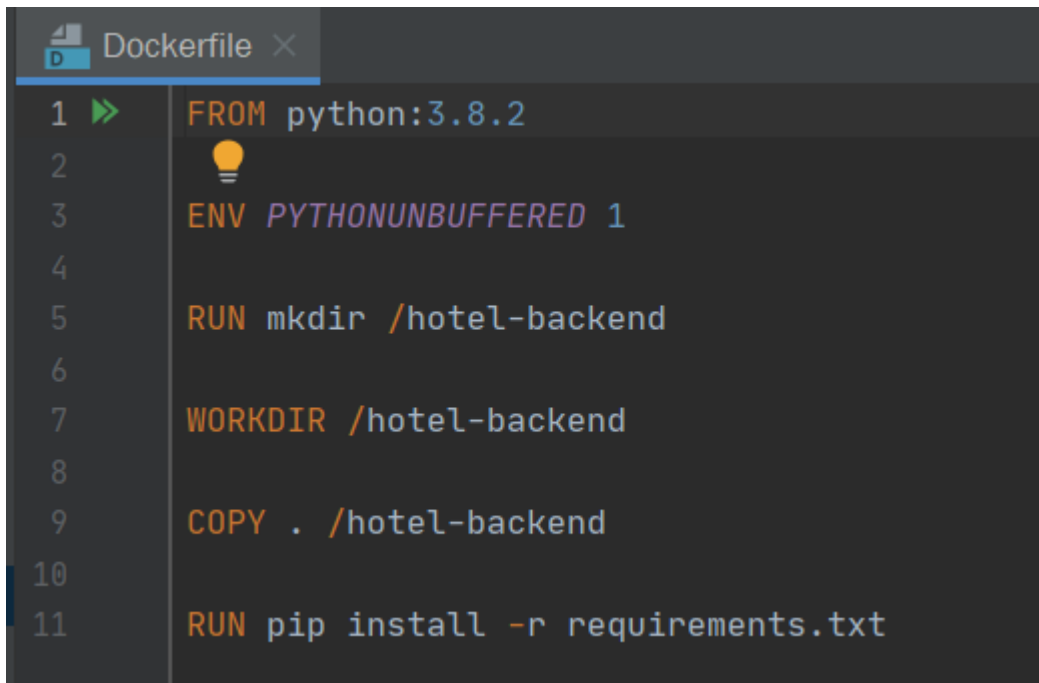
УДАЛИТЬ ЗАПИСЬ

Рисунок 21–фильтрация по дню недели

4. КОНТЕЙНЕРИЗАЦИЯ И ОРКЕСТРАЦИЯ WEB-ПРИЛОЖЕНИЯ

Docker - это ПО с открытым исходным кодом, который упрощает создание контейнеров и приложений на основе контейнеров. Первоначально разработанный для Linux, Docker теперь работает также на Windows и MacOS. Чтобы понять, как работает Docker, нужно рассмотреть компоненты, которые используются для создания контейнеризованных приложений.

Каждый контейнер Docker начинается с Dockerfile – это текстовый файл, который включает инструкции по созданию образа Docker. Dockerfile определяет операционную систему, которая будет лежать в основе контейнера, а также языки, переменные среды, расположение файлов, сетевые порты, необходимые библиотеки и действия контейнера после его запуска. Dockerfile для бэкенда реализованного web-приложения представлен на рисунке 22.



```
1 FROM python:3.8.2
2
3 ENV PYTHONUNBUFFERED 1
4
5 RUN mkdir /hotel-backend
6
7 WORKDIR /hotel-backend
8
9 COPY . /hotel-backend
10
11 RUN pip install -r requirements.txt
```

Рисунок 22 –Dockerfile

Далее создается файл docker-compose, который используется для управления несколькими контейнерами, входящими в состав приложения.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы был создан web-сервис для администрирования гостиницы. Стек используемых технологий включал в себя:

- PostgreSQL – для работы с базой данных
- Django и Django REST Framework – web-фреймворк языка программирования Python для создания web-приложений
- Vue.js – web-фреймворк языка программирования JavaScript для создания пользовательских интерфейсов
- MUSE-UI – библиотека Vue.js для дизайна пользовательского интерфейса, основанная на Material Design

Были реализованы все заявленные функциональные требования, согласно которым были созданы пользовательские интерфейсы:

- Авторизация и регистрация пользователей
- Выполнение запросов
- Интерфейс с информацией об имеющихся в гостинице номерах
- Интерфейс с информацией о проживающих с функциями создания, редактирования и удаления
- Интерфейс с информацией о служащих функциями создания, редактирования и удаления

СПИСОК ЛИТЕРАТУРЫ

1. Django Rest Framework. Документация Django Rest Framework [Электронный ресурс]. URL: <https://www.django-rest-framework.org> (дата обращения: 29.06.2020).
2. WebDevBlog. Создание Django API используя Django Rest Framework [Электронный ресурс]. URL: <https://webdevblog.ru/sozдание-django-api-ispolzuya-django-rest-framework-apiview/> (дата обращения: 29.06.2020).
3. Evantotuts+. JWT Аутентификация в Django [Электронный ресурс] URL: <https://code.tutsplus.com/ru/tutorials/how-to-authenticate-with-jwt-in-django--cms-30460> (дата обращения: 29.06.2020).
4. Vue.js. Документация Vue.js [Электронный ресурс]. URL: <https://vuejs.org> (дата обращения: 29.06.2020).

ПРИЛОЖЕНИЯ

```

from django.db import models

class Floor(models.Model):
    floor_number = models.IntegerField("Номер этажа")
    number_of_rooms = models.IntegerField("Количество комнат на этаже")

    class Meta:
        verbose_name = "Этаж"
        verbose_name_plural = "Этажи"

class Servant(models.Model):
    fio = models.CharField("ФИО служащего", max_length=300)

    class Meta:
        verbose_name = "Служащий"
        verbose_name_plural = "Служащие"

    def __str__(self):
        return self.fio

class Cleaning(models.Model):
    servant = models.ForeignKey(Servant, on_delete=models.CASCADE)
    floor = models.ForeignKey(Floor, on_delete=models.CASCADE)
    week_day = models.TextChoices('week_day', 'Понедельник Вторник Среда Четверг Пятница Суббота Воскресенье')
    day = models.CharField("День недели", blank=True, choices=week_day.choices, max_length=20)

    class Meta:
        verbose_name = "График уборки"
        verbose_name_plural = "Графики уборки"

class RoomType(models.Model):
    types = models.TextChoices('types', 'Одноместный Двухместный Трехместный')
    room_type = models.CharField("Тип номера", blank=True, choices=types.choices, max_length=20)
    price = models.IntegerField("Цена проживания в сутки в рублях")

    class Meta:
        verbose_name = "Тип номера"
        verbose_name_plural = "Типы номеров"

    def __str__(self):
        return self.room_type

class Room(models.Model):
    room_number = models.CharField("Номер комнаты", max_length=4)
    floor = models.ForeignKey(Floor, on_delete=models.CASCADE, related_name="rooms")
    room_type = models.ForeignKey(RoomType, on_delete=models.CASCADE)
    phone_number = models.CharField("Номер телефона", max_length=6)

    class Meta:
        verbose_name = "Номер"

```

```
    verbose_name_plural = "Номера"

    def __str__(self):
        return "Номер "+self.room_number

class Resident(models.Model):
    room = models.ForeignKey(Room, on_delete=models.CASCADE)
    fio = models.CharField("ФИО проживающего", max_length=200)
    passport_number = models.CharField("Серия и номер паспорта", max_length=10)
    from_town = models.CharField("Прибыл из города", max_length=50)
    check_in = models.DateField("Дата заселения")
    check_out = models.DateField("Дата выселения")

    class Meta:
        verbose_name = "Проживающий"
        verbose_name_plural = "Проживающие"

    def __str__(self):
        return self.fio
# Create your models here.

# Create your models here.
```


Приложение 2. Файл serializers.py

```
from rest_framework import serializers
from .models import Floor, RoomType, Room, Resident, Servant, Cleaning

class RoomTypeSerializer(serializers.ModelSerializer):
    """Сериализатор для модели Тип Комнаты"""

    class Meta:
        model = RoomType
        fields = "__all__"

class RoomSerializer(serializers.ModelSerializer):
    """Сериализатор для модели Комната"""
    room_type = serializers.SlugRelatedField(slug_field="room_type", read_only=True)
    floor = serializers.SlugRelatedField(slug_field="floor_number", read_only=True)

    class Meta:
        model = Room
        fields = "__all__"

class FloorSerializer(serializers.ModelSerializer):
    """Сериализатор для модели Этаж"""
    rooms = RoomSerializer(many=True)

    class Meta:
        model = Floor
        fields = "__all__"

class ResidentSerializer(serializers.ModelSerializer):
    """Сериализатор для модели Проживающий"""
    room = serializers.SlugRelatedField(slug_field="room_number", read_only=True)

    class Meta:
        model = Resident
        fields = "__all__"

class ResidentCreateSerializer(serializers.ModelSerializer):
    """Сериализатор для создания новой записи в модели Проживающий"""

    class Meta:
        model = Resident
        fields = "__all__"

class ServantSerializer(serializers.ModelSerializer):
    """Сериализатор для модели Служащий"""

    class Meta:
        model = Servant
        fields = "__all__"

class CleaningSerializer(serializers.ModelSerializer):
```

```
"""Сериализатор для модели Уборка"""
floor = serializers.SlugRelatedField(slug_field="floor_number", read_only=True)
servant = serializers.SlugRelatedField(slug_field="fio", read_only=True)

class Meta:
    model = Cleaning
    fields = "__all__"

class CleaningCreateSerializer(serializers.ModelSerializer):
    """Сериализатор для создания новой записи в модели Уборка"""
    class Meta:
        model = Cleaning
        fields = "__all__"
```

```

from django.shortcuts import render
from rest_framework import generics, permissions, viewsets, renderers
from rest_framework.views import APIView
from rest_framework.response import Response
from collections import Counter
from django.db.models import Count, Avg

from .models import Floor, RoomType, Room, Resident, Servant, Cleaning
from .serializers import FloorSerializer, RoomTypeSerializer, RoomSerializer, ResidentSerializer,\
    ResidentCreateSerializer, ServantSerializer, CleaningSerializer, CleaningCreateSerializer


class FloorViewSet(viewsets.ModelViewSet):
    """Отображение для модели Этаж"""
    queryset = Floor.objects.all()
    serializer_class = FloorSerializer


class RoomTypeViewSet(viewsets.ModelViewSet):
    """Отображение для модели Тип Комнаты"""
    queryset = RoomType.objects.all()
    serializer_class = RoomTypeSerializer


class RoomViewSet(viewsets.ModelViewSet):
    """Отображение для модели Комната"""
    queryset = Room.objects.all()
    serializer_class = RoomSerializer


class ResidentViewSet(viewsets.ModelViewSet):
    """Отображение для модели Проживающий"""
    queryset = Resident.objects.all()

    def get_serializer_class(self):
        if self.action == 'create':
            return ResidentCreateSerializer
        elif self.action != 'create':
            return ResidentSerializer


class ServantViewSet(viewsets.ModelViewSet):
    """Отображение для модели Служащий"""
    queryset = Servant.objects.all()
    serializer_class = ServantSerializer


class CleaningViewSet(viewsets.ModelViewSet):
    """Отображение для модели Уборка"""
    queryset = Cleaning.objects.all()

    def get_serializer_class(self):
        if self.action == 'create':
            return CleaningCreateSerializer
        elif self.action != 'create':
            return CleaningSerializer

```

```
"""Запросы к курсовой работе"""
```

```
class Query1(APIView):
```

```
    """о клиентах, проживавших в заданном номере"""
```

```
    def get(self, request):
```

```
        room = request.GET.get('room_number')
```

```
        resident_list = Resident.objects.filter(room=room)
```

```
        serializer = ResidentSerializer(resident_list, many=True)
```

```
        return Response({'result': serializer.data})
```

```
class Query3(APIView):
```

```
    """о том, кто из служащих убирал номер указанного клиента в заданный день недели"""
```

```
    def get(self, request):
```

```
        room = request.GET.get('resident')
```

```
        day = request.GET.get('day')
```

```
        floor1 = Room.objects.filter(room_number=room)[0].floor
```

```
        servant1 = Cleaning.objects.filter(floor=floor1, day=day)[0].servant
```

```
        result = str(servant1)
```

```
        return Response({'result': result})
```

```
class Query2(APIView):
```

```
    """о количестве клиентов, прибывших из заданного города"""
```

```
    def get(self, request):
```

```
        results = Resident.objects.values('from_town').order_by('from_town').annotate(Count('flo'))
```

```
        return Response({'result': results})
```

```
class Query4(APIView):
```

```
    """сколько в гостинице свободных номеров"""
```

```
    def get(self, request):
```

```
        floor1 = Floor.objects.filter(floor_number=1)[0].number_of_rooms
```

```
        floor2 = Floor.objects.filter(floor_number=2)[0].number_of_rooms
```

```
        rooms = Room.objects.all().aggregate(Count('id'))['id_count']
```

```
        results = floor1+floor2-rooms
```

```
        return Response({'result': results})
```

```
# Create your views here.
```