

**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 09.03.02. - Мобильные сетевые технологии

Направление подготовки (специальность) 09.03.02. - Мобильные сетевые технологии

## **О Т Ч Е Т**

по курсовой работе

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

Обучающийся Валенкевич В.Л. К3340

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 .....	7
2.3. Создание отображений .....	8
2.4. Интерфейсы панели Django REST .....	9
3. ОПИСАНИЕ КЛИЕНТСКОЙ ЧАСТИ .....	13
3.1. Средства разработки клиентской части .....	13
3.2. Интерфейсы Vue .....	13
4. РАБОТА С DOCKER .....	18
ЗАКЛЮЧЕНИЕ .....	19
СПИСОК ЛИТЕРАТУРЫ .....	20
ПРИЛОЖЕНИЯ .....	21

## ВВЕДЕНИЕ

В рамках курсовой работы по дисциплине «Основы web-программирования» необходимо было создать программную систему, предназначенную для администрации аэропорта некоторой компании-авиаперевозчика.

Рейсы обслуживаются бортами, принадлежащими разным авиаперевозчикам. О каждом самолете необходима следующая минимальная информация: номер самолета, тип, число мест, скорость полета, компания-авиаперевозчик. Один тип самолета может летать на разных маршрутах и по одному маршруту могут летать разные типы самолетов.

О каждом рейсе необходима следующая информация: номер рейса, расстояние до пункта назначения, пункт вылета, пункт назначения; дата и время вылета, дата и время прилета, транзитные посадки (если есть), пункты посадки, дата и время транзитных посадок и дат и время их вылета, количество проданных билетов. Каждый рейс обслуживается определенным экипажем, в состав которого входят командир корабля, второй пилот, штурман и стюардессы или стюарды. Каждый экипаж может обслуживать разные рейсы на разных самолетах. Необходимо предусмотреть наличие информации о допуске члена экипажа к рейсу. Администрация компании-владельца аэропорта должна иметь возможность принять работника на работу или уволить. При этом необходима следующая информация: ФИО, возраст, образование, стаж работы, паспортные данные. Эта же информация необходима для сотрудников сторонних компаний

Перечень возможных запросов:

- Определить наличие свободных мест на заданный рейс.
- Определить количество самолетов, находящихся в ремонте.
- Определить количество работников компания-авиаперевозчика.

Необходимо предусмотреть возможность получения отчета о бортах компании-владельца по маркам с характеристикой марки. Указать общее количество бортов и количество бортов по каждой марке.

# **1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ**

## **1.1. Описание предметной области**

Предметной областью данной курсовой работы является администрирование аэропорта. Основным пользователем разрабатываемого web-сервиса – это сотрудник аэропорта, который занимается наймом служащих: командиров экипажа, вторых пилотов, штурманов, бортпроводников; входящих в экипаж самолетов компаний-авиаперевозчиков, которые базируются в этом аэропорту. Этому сотруднику-пользователю должна быть доступна информация обо всех авиакомпаниях, которые относятся к данному аэропорту, о маршрутах, о рейсах и о самолетах.

Кроме того, пользователю может понадобиться информация, которая является результатом выполнения следующих запросов:

- Определить наличие свободных мест на заданный рейс.
- Определить количество самолетов, находящихся в ремонте.
- Определить количество работников компания-авиаперевозчика.

Также пользователю необходимо получать отчет об авиакомпаниях с информацией об их бортах.

## **1.2. Описание функциональных требований**

Функциональные требования к web-приложению должны отвечать всем установленным запросам и максимально удовлетворять потребностям предметной области. Все функциональные требования можно разделить на несколько групп по виду манипулирования информацией, а именно: блок требований, касающийся функций просмотра информации, блок требований для функций добавления информации, блок требований для функций редактирования информации, а также удаления информации в базе данных. Кроме того, отдельной группой являются такие функциональные требования как авторизация, регистрация, выполнение запросов и выдача отчета.

Таким образом, были выявлены следующие функциональные требования:

1. Функции просмотра
  - a. Информации об авиакомпаниях
  - b. Информации о самолетах
  - c. Информации о маршрутах

- d. Информации о рейсах
  - e. Информации об экипажах
  - f. Информации о командирах экипажа
  - g. Информации о вторых пилотах
  - h. Информации о штурманах
  - i. Информации о бортпроводниках
- 2. Функции добавления
    - a. Добавление нового командира экипажа
    - b. Добавление нового второго пилота
    - c. Добавление нового штурмана
    - d. Добавление нового бортпроводника
  - 3. Функции редактирования
    - a. Профиля командира экипажа
    - b. Профиля второго пилота
    - c. Профиля штурмана
    - d. Профиля бортпроводника
  - 4. Функции удаления
    - a. Удаление командира экипажа
    - b. Удаление второго пилота
    - c. Удаление штурмана
    - d. Удаление бортпроводника
  - 5. Функции регистрации и авторизации
  - 6. Функции выполнения запросов и выдачи отчета

## 2. ОПИСАНИЕ СЕРВЕРНОЙ ЧАСТИ

### 2.1. Средства разработки серверной части

Для разработки серверной части web-приложения был использован следующий стек технологий:

- PostgreSQL 12 – свободная объектно-реляционная система управления базами данных. Благодаря свободной лицензии и открытому коду, PostgreSQL разрешается использовать бесплатно, изменять и распространять всем и для любых целей: личных, коммерческих или учебных. В силу того, что хранение базы данных происходит на сервере, а не локально, переход web-приложения со стадии разработки на стадию продакшена происходит быстрее и проще.
- Django 3 – свободный фреймворк для web-приложений на языке программирования Python, использующий шаблон проектирования MVC (Model-View-Controller). Контроллер классической модели MVC примерно соответствует уровню, который в Django называется Представление (англ. View), а презентационная логика Представления реализуется в Django уровнем Шаблонов (англ. Template). Из-за этого уровневую архитектуру Django часто называют «Модель-Шаблон-Представление» (MTV). Сайт на Django строится из одного или нескольких приложений, которые рекомендуется делать отчуждаемыми и подключаемыми. Это одно из существенных архитектурных отличий этого фреймворка от некоторых других, например, Ruby on Rails.
- Django REST framework – удобный инструмент для работы с REST основанный на идеологии фреймворка Django. 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 предметной областью была разработана модель базы данных, представленная на рисунке 1.

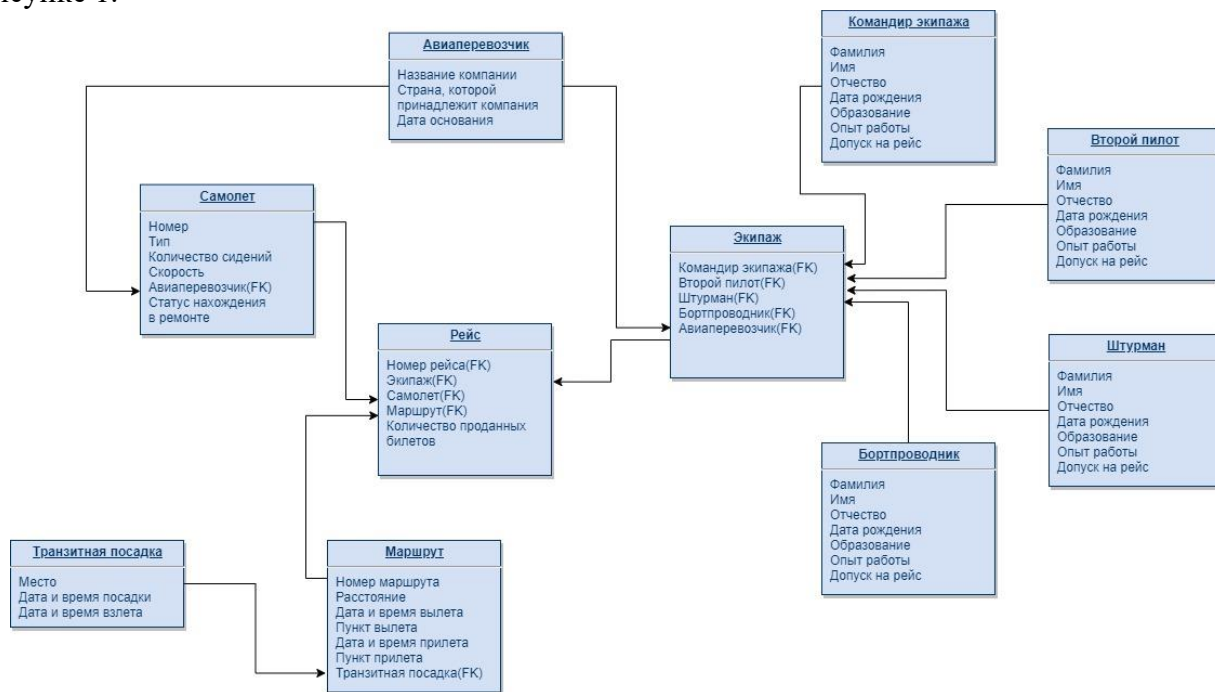


Рисунок 1 – модель базы данных

Спроектированная модель базы данных реализована в качестве моделей Django в файле `models.py`, листинг кода данного файла приведен в приложении 1. Были созданы следующие модели:

- Class `AirCarrier` – Компания-авиаперевозчик – содержит информацию о названии авиакомпании, стране, которой принадлежит авиакомпания и годе основания авиакомпании.
- Class `TransitLanding` – Транзитная посадка – содержит информацию о месте посадки, дате и времени прилета и вылета.
- Class `Route` – Маршрут – содержит информацию о номере маршрута, расстоянии в километрах, месте вылета, дате и времени вылета, месте прилета, дате и времени прилета и о наличии транзитной посадки.
- Class `Plane` – Самолет – содержит информацию о номере самолета, типе самолета, количестве сидений, средней скорости и о нахождении в ремонте.
- Class `CrewCommander` – Командир экипажа – содержит информацию о фамилии, имени, отчестве, дате рождения, образовании, опыте и о допуске на рейс.

- Class SecondPilot – Второй пилот – содержит информацию о фамилии, имени, отчестве, дате рождения, образовании, опыте и о допуске на рейс.
- Class Navigator – Штурман – содержит информацию о фамилии, имени, отчестве, дате рождения, образовании, опыте и о допуске на рейс.
- Class Steward – Бортпроводник – содержит информацию о фамилии, имени, отчестве, дате рождения, образовании, опыте и о допуске на рейс.
- Class Crew – Экипаж – содержит информацию о составе экипажа.
- Class Flight – Рейс – содержит информацию о номере рейса, экипаже, самолете, маршруте и количестве проданных билетов.

### 2.3. Создание отображений

Для того, чтобы обеспечить обмен данных между сервером, написанным на Django, и клиентской частью, написанной на Vue.js, необходимо провести сериализацию. Сериализация – процесс преобразования структур данных или объекта состояния в формат, который может быть сохранен или передан и реконструирован позже. Среда сериализации в Django предоставляет механизм для «перевода» моделей Django в другие форматы, например, в JSON, который принимает Vue.js. Все сериализаторы для созданных моделей описаны в файле `serializers.py`, приведенном в приложении 2. 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 FlightViewSet(viewsets.ModelViewSet)` – класс отображения для модели Рейс. . В качестве класса сериализатора использовался `FlightSerializer`.
- `class AirCarrierViewSet(viewsets.ModelViewSet)` – класс отображения для модели Авиаперевозчик. В качестве класса сериализатора использовался `AirCarrierSerializer`.
- `class CrewViewSet(viewsets.ModelViewSet)` – класс отображения для модели Экипаж. В качестве класса сериализатора использовался `CrewSerializer`.
- `class RouteViewSet(viewsets.ModelViewSet)` – класс отображения для модели Маршрут. В качестве класса сериализатора использовался `RouteSerializer`.



- class TransitViewSet(viewsets.ModelViewSet) – класс отображения для модели Транзитная посадка. В качестве класса сериализатора использовался TransitLandingSerializer.
- class PlaneViewSet(viewsets.ModelViewSet) – класс отображения для модели Самолет. В качестве класса сериализатора использовался PlaneSerializer.
- class CrewCommanderViewSet(viewsets.ModelViewSet) – класс отображения для модели Командир экипажа. В качестве класса сериализатора при вызове метода «update» использовался CrewFIOCommanderSerializer, при вызове остальных методов использовался CrewCommanderSerializer.
- class SecondPilotViewSet(viewsets.ModelViewSet) – класс отображения для модели Второй пилот. В качестве класса сериализатора при вызове метода «update» использовался SecondPilotFIOSerializer, при вызове остальных методов использовался SecondPilotSerializer.
- class StewardViewSet(viewsets.ModelViewSet) – класс отображения для модели Бортпроводник. В качестве класса сериализатора при вызове метода «update» использовался StewardFIOSerializer, при вызове остальных методов использовался StewardSerializer.
- class NavigatorViewSet(viewsets.ModelViewSet) – класс отображения для модели Штурман. В качестве класса сериализатора при вызове метода «update» использовался NavigatorFIOSerializer, при вызове остальных методов использовался NavigatorSerializer.
- class Query1 – отображение для вывода результата выполнения запроса 1 «Определить наличие свободных мест на заданный рейс.».
- class Query2 – отображение для вывода результата выполнения запроса 2 «Определить количество самолетов, находящихся в ремонте.».
- class Query3 – отображение для вывода результата выполнения запроса 3 «Определить количество работников аэропорта».
- class AirCarrierReport – отображение для вывода отчет о бортах компании-владельца.

## 2.4. Интерфейсы панели Django REST

### а. Авиакомпания

Выводятся данные обо всех авиакомпаниях, а также о самолетах и экипажах, которые к ним относятся. Скриншот представлен на рисунке 2.

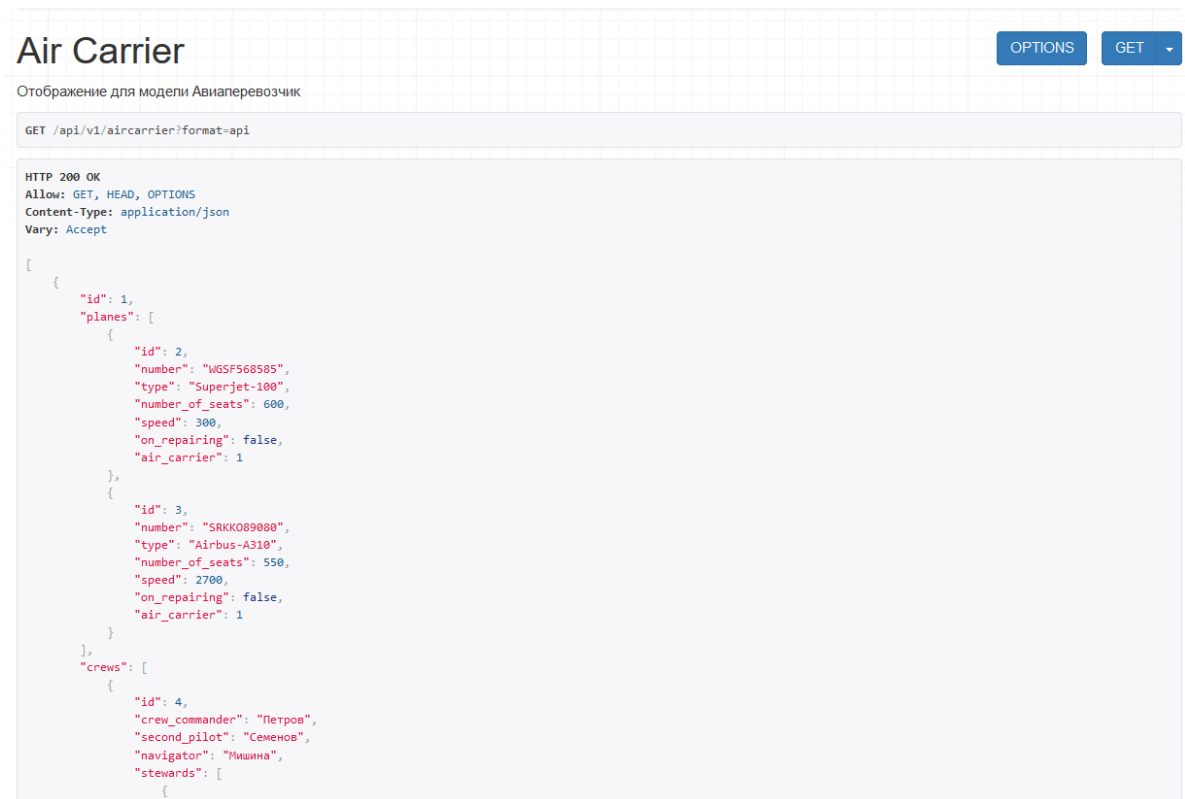


Рисунок 2 – Авиакомпании в Django REST

## б. Экипаж

Вывод заданного экипажа с подробной информацией о каждом его члене. Скриншот представлен на рисунке 3.

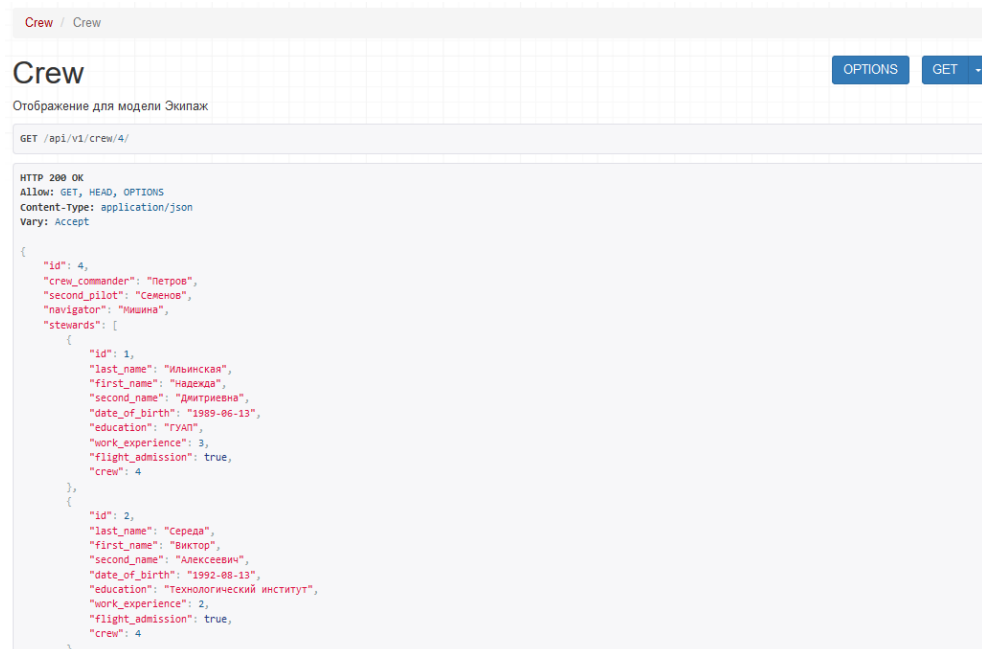


Рисунок 3 – заданный Экипаж в Django REST

### с. Командир экипажа

Добавление командира экипажа. Аналогичные интерфейсы имеют второй пилот, штурман и бортпроводник. Скриншот представлен на рисунке 4.

Django REST framework Log in

Crew Commander / Crew Commander

## Crew Commander

Отображение для модели Командир экипажа

GET /api/v1/crewcommander/create

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

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

Raw data HTML form

Фамилия Синичкин

Имя Вадим

Отчество Сергеевич

Дата рождения 17. 04. 1968

Образование СПбГУ

Опыт работы в годах 25

Допуск на рейс ☒

POST

Рисунок 4 – создание Командира экипажа в Django REST

### d. Самолеты

Вывод информации обо всех самолетах. Скриншот представлен на рисунке 5.

Django REST framework Log in

Plane

## Plane

Отображение для модели Самолет

GET /api/v1/plane/

HTTP 200 OK  
Allow: GET, HEAD, OPTIONS  
Content-Type: application/json  
Vary: Accept

```
[
  {
    "id": 1,
    "number": "ЧР6798877",
    "type": "Ty-204",
    "number_of_seats": 500,
    "speed": 2200,
    "on_repairing": false,
    "air_carrier": 2
  },
  {
    "id": 2,
    "number": "W05F68555",
    "type": "Superjet-100",
    "number_of_seats": 600,
    "speed": 280,
    "on_repairing": false,
    "air_carrier": 1
  },
  {
    "id": 3,
    "number": "D0KX08988",
    "type": "Airbus-A320",
    "number_of_seats": 550,
    "speed": 2700,
    "on_repairing": false,
    "air_carrier": 1
  }
]
```

Рисунок 5 – Самолеты в Django REST

## е. Маршруты

Вывод информации обо всех маршрутах. Скриншот представлен на рисунке 6.

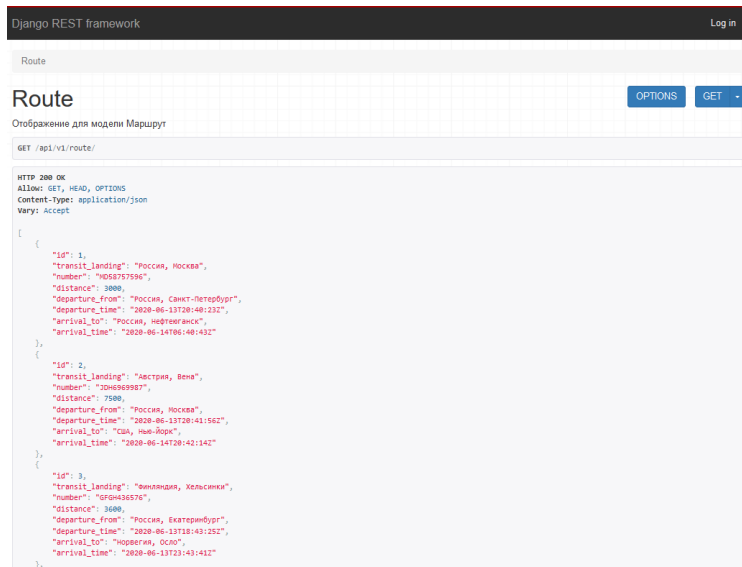


Рисунок 6 – Маршруты в Django REST

## ф. Рейсы

Вывод информации обо всех рейсах. Скриншот представлен на рисунке 7.

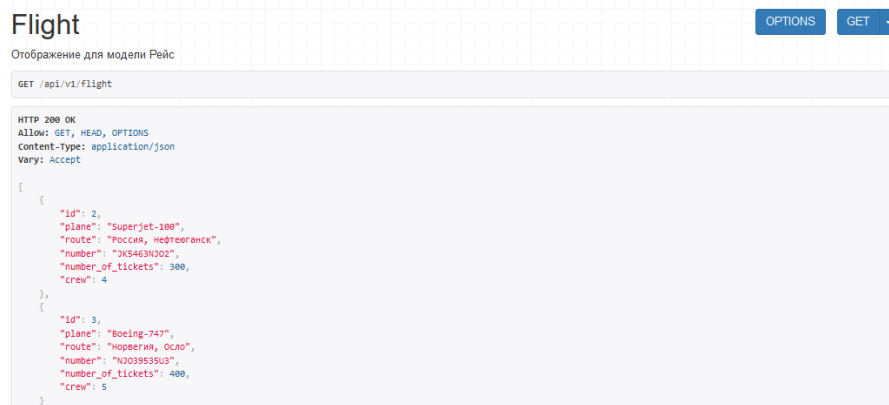


Рисунок 7 – Рейсы в Django REST

## 3. ОПИСАНИЕ КЛИЕНТСКОЙ ЧАСТИ

### 3.1. Средства разработки клиентской части

Для создания клиентской части был использован JavaScript-фреймворк Vue.js и его библиотека Muse-ui. Vue.js — это JavaScript библиотека для создания веб-интерфейсов с использованием шаблона архитектуры MVVM (Model-View-ViewModel).

Поскольку Vue работает только на «уровне представления» и не используется для промежуточного программного обеспечения и бэкэнда, он может легко интегрироваться с другими проектами и библиотеками. Vue.js содержит широкую функциональность для уровня представлений и может использоваться для создания мощных одностраничных веб-приложений.

Библиотека Muse UI, имеющая около 5 тысяч звёзд на GitHub, представляет собой набор компонентов для Vue 2.0, использующих Material Design.

### 3.2. Интерфейсы Vue

#### а. Стартовая страница

Стартовая страница web-сервиса с верхним меню навигации и списком компаний-авиаперевозчиков. Скриншот представлен на рисунке 8.

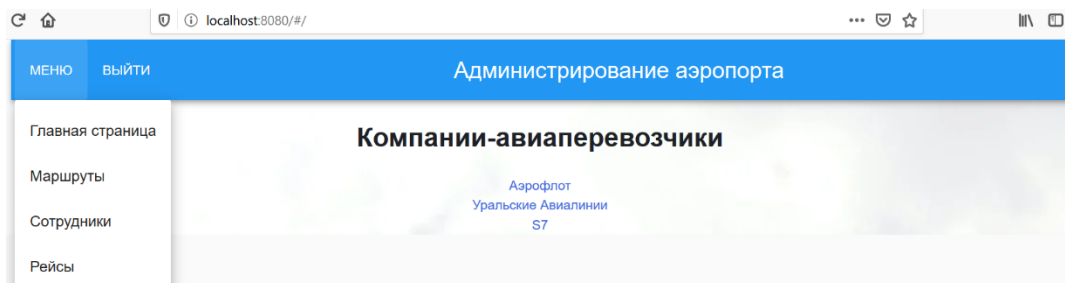


Рисунок 8 – Стартовая страница web-сервиса

#### б. Вход

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

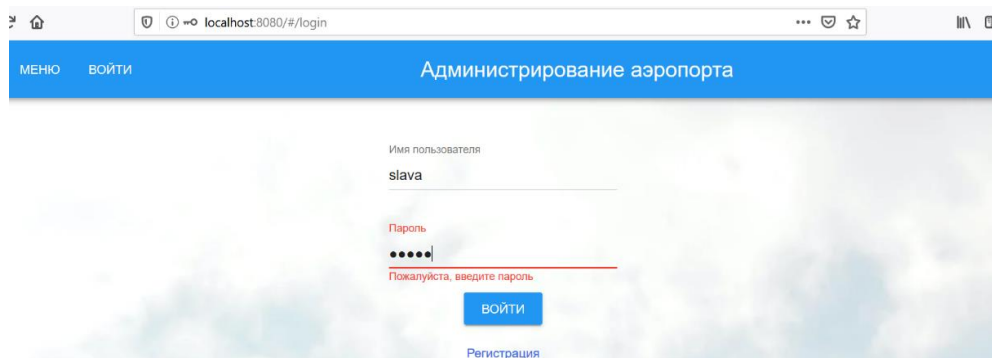


Рисунок 9 – Страница входа

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

МЕНЮ ВОЙТИ Администрирование аэропорта

**Пожалуйста, заполните следующие поля**

Поля должны содержать не менее 8 символов.

user4321

Password  
..... visibility

Password again  
..... visibility

ЗАРЕГИСТРИРОВАТЬСЯ

d. Просмотр определенной компании-авиаперевозчика

На данной странице показывается информация о выбранной авиакомпании. При нажатии на кнопку «Самолеты» появляется информация о самолетах, которые относятся к данной авиакомпании, при нажатии на кнопку «Экипажи» - об экипажах. Скриншот представлен на рисунке 11

МЕНЮ ВЫЙТИ Администрирование аэропорта

**Авиакомпания "Аэрофлот"**

Страна, в которой основана компания: Российская Федерация Дата основания компании: 1965-06-12

САМОЛЕТЫ ЭКИПАЖИ

Самолет номер WGSF568585

Тип самолета	Superjet-100
Количество сидений	600
Скорость самолета	300 км/ч
Находится в ремонте	false

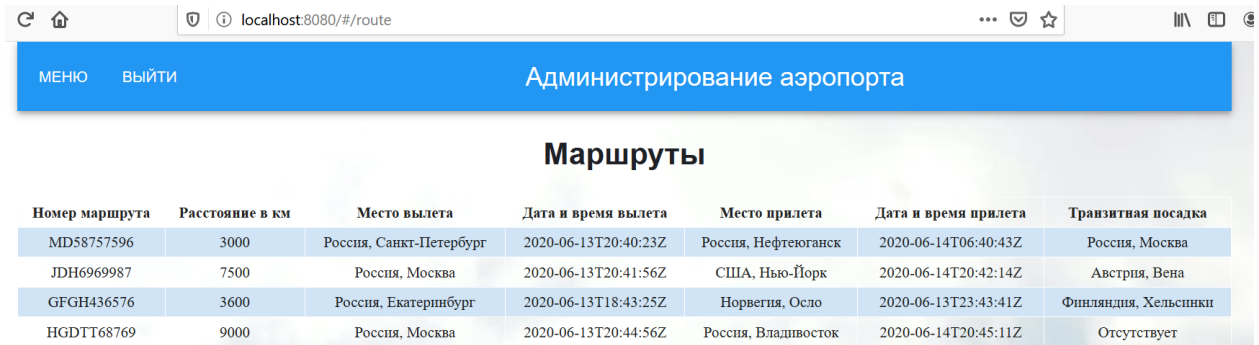
Самолет номер SRKKO89080

Тип самолета	Airbus-A310
Количество сидений	550
Скорость самолета	2700 км/ч
Находится в ремонте	false

Рисунок 11 – Страница авиакомпании Аэрофлот

е. Просмотр всех маршрутов

Страница с таблицей маршрутов. .Скриншот представлен на рисунке 12



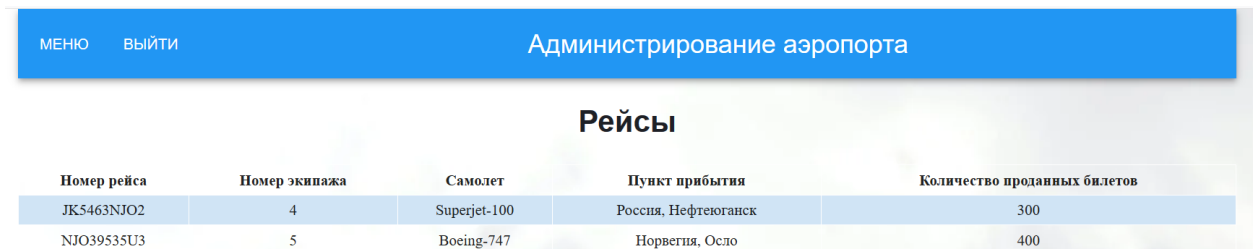
The screenshot shows a web application interface for airport administration. At the top, there is a blue header bar with the text 'Администрирование аэропорта' and two links: 'МЕНЮ' and 'ВЫЙТИ'. Below the header, the title 'Маршруты' is centered. A table with 7 columns is displayed, showing flight routes with details like route number, distance, departure/arrival locations, times, and transit stops.

Номер маршрута	Расстояние в км	Место вылета	Дата и время вылета	Место прилета	Дата и время прилета	Транзитная посадка
MD58757596	3000	Россия, Санкт-Петербург	2020-06-13T20:40:23Z	Россия, Нефтеюганск	2020-06-14T06:40:43Z	Россия, Москва
JDH6969987	7500	Россия, Москва	2020-06-13T20:41:56Z	США, Нью-Йорк	2020-06-14T20:42:14Z	Австрия, Вена
GFGH436576	3600	Россия, Екатеринбург	2020-06-13T18:43:25Z	Норвегия, Осло	2020-06-13T23:43:41Z	Финляндия, Хельсинки
HGDTT68769	9000	Россия, Москва	2020-06-13T20:44:56Z	Россия, Владивосток	2020-06-14T20:45:11Z	Отсутствует

Рисунок 12 – Страница с таблицей маршрутов

f. Просмотр всех рейсов

Страница с таблицей с информацией об авиарейсах. .Скриншот представлен на рисунке 13.



The screenshot shows the same web application interface as Figure 12, but with the title 'Рейсы' centered. A table with 5 columns is displayed, showing flight details such as flight number, crew member, aircraft type, destination, and the number of sold tickets.

Номер рейса	Номер экипажа	Самолет	Пункт прибытия	Количество проданных билетов
JK5463NJO2	4	Superjet-100	Россия, Нефтеюганск	300
NJO39535U3	5	Boeing-747	Норвегия, Осло	400

Рисунок 13 – Страница со всеми авиарейсами

g. Просмотр всех сотрудников аэропорта

Страница со списком сотрудников, при нажатии на каждую должность появляется таблица с ФИО сотрудника и кнопка, при нажатии на которую осуществляется переход на страницу выбранного сотрудника. Имеются также кнопки добавления сотрудника каждой должности, при нажатии на которые появляется форма добавления. .Скриншот представлен на рисунке 14, 15 и 16.

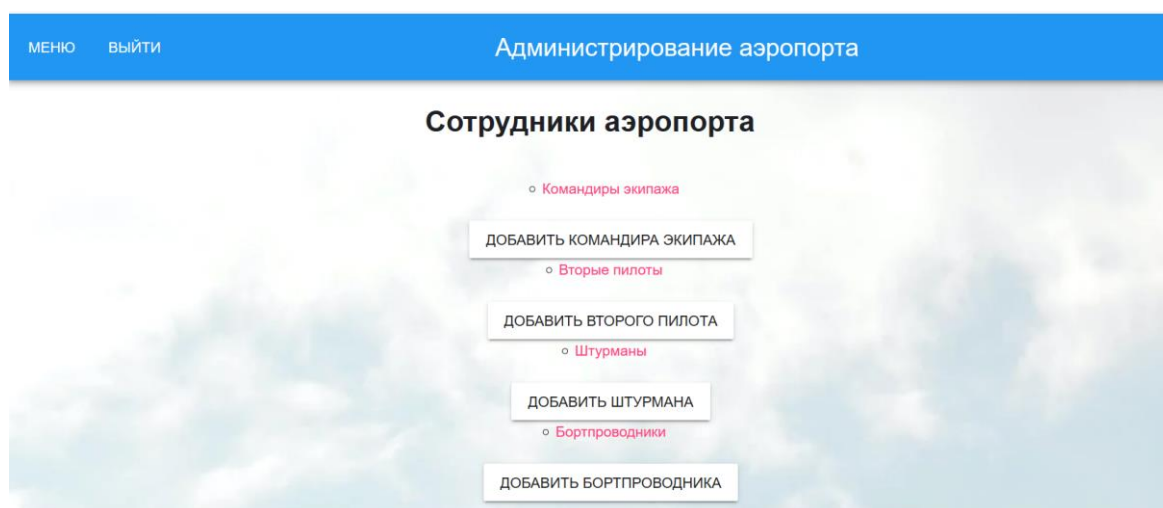


Рисунок 14 – Страница со всеми сотрудниками аэропорта

◦ Штурманы

Фамилия  
Алексеев

Имя  
Артём

Отчество  
Михайлович

Дата рождения гггг-мм-дд  
1987-11-30

Образование  
Высшая летная академия

Опыт работы в годах  
5

Допуск на рейс  
☒ Есть ☐ Нет

ДОБАВИТЬ

Рисунок 15 – Добавление нового Штурмана

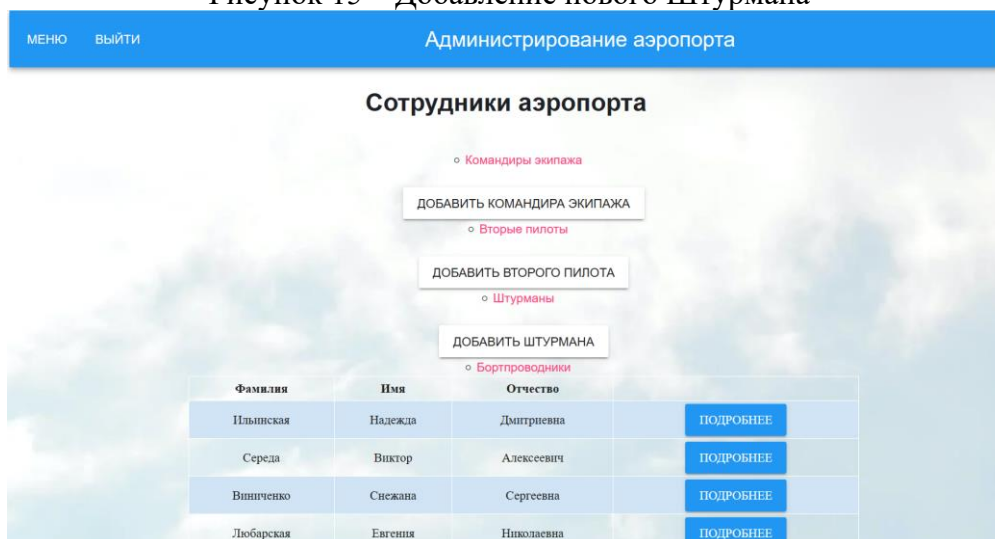


Рисунок 16 – Просмотр всех Бортпроводников



- h. Просмотр всей информации о выбранном бортпроводнике
- Страница с полной информацией о выбранном бортпроводнике. Имеется кнопка «Удалить» и «Изменить» с соответствующими функциями. Аналогичный интерфейс имеют страницы с подробной информацией о командире экипажа, втором пилоте и штурмане. Скриншот представлен на рисунке 17.

Фамилия	Внннченко
Имя	Снежана
Отчество	Сергеевна
Дата рождения	1993-06-13
Образование	ГУАП
Опыт работы в годах	1
Допуск на рейс	true
Экипаж	5

ИЗМЕНИТЬ УДАЛИТЬ

Рисунок 17 – Страница с подробной информацией о выбранном Бортпроводнике

- i. Изменение информации о выбранном бортпроводнике
- При нажатии на кнопку «Изменить» на рис.17 появляется форма с возможностью изменения данных (Фамилии, Имени, Отчества) у выбранного бортпроводника. Аналогичные интерфейсы созданы для командира экипажа, второго пилота и штурмана. Скриншот представлен на рисунке 18.

Фамилия	Внннченко
Имя	Снежана
Отчество	Сергеевна
Дата рождения	1993-06-13
Образование	ГУАП
Опыт работы в годах	1
Допуск на рейс	true
Экипаж	5

УДАЛИТЬ

Новая фамилия	Новое имя	Новое отчество
Мильковская	Снежана	Сергеевна

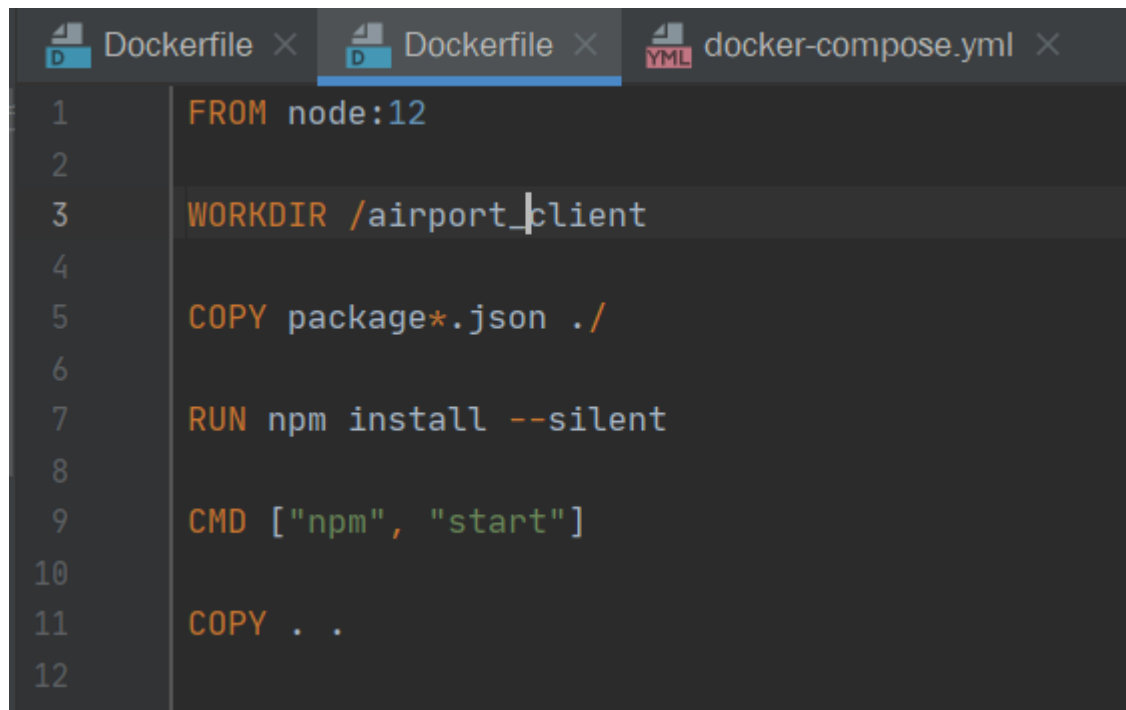
ВНЕСТИ ИЗМЕНЕНИЯ

Рисунок 18 – Изменение информации о выбранном Бортпроводнике

## 4. РАБОТА С DOCKER

Docker — это открытая платформа для разработки, доставки и эксплуатации приложений. С помощью технологии Docker (контейнеризации) можно разделить исходное приложение на несколько компонентов, которые взаимодействие между которыми возможно реализовать. Подобный подход может привести разные методы реализации компонентов, тем самым предоставляя разработчику широкий спектр возможностей. Благодаря этому разработчику предоставляется возможность создания микро-сервисных архитектур.

Dockerfile – это сценарий, который состоит из последовательности команд и аргументов, необходимых для создания образа. Такие сценарии упрощают развёртывание и процесс подготовки приложения к запуску. Создается виртуальное окружение, внутри которого будет собираться/исполняться программа. Dockerfile для контейнеризации клиентской части разработанного web-приложения представлен на рисунке 19.

A screenshot of a code editor with three tabs: 'Dockerfile', 'Dockerfile', and 'docker-compose.yml'. The first 'Dockerfile' tab is active and shows a Dockerfile with 12 lines of code. The code is as follows:

```
1 FROM node:12
2
3 WORKDIR /airport_client
4
5 COPY package*.json ./
6
7 RUN npm install --silent
8
9 CMD ["npm", "start"]
10
11 COPY . .
12
```

Рисунок 19 –Dockerfile

Docker Compose - инструмент для создания и запуска многоконтейнерных Docker приложений. В Compose используется специальный файл для конфигурирования сервисов приложения. Затем используется простая команда для создания и запуска всех сервисов из конфигурационного файла. Это самая тривиальная реализация микросервисной архитектуры.

## ЗАКЛЮЧЕНИЕ

Данная курсовая работа по дисциплине «Основы web-программирования» показывает полученные в течение семестра навыки разработки и создания web-приложений с помощью стека технологий:

- PostgreSQL – свободная объектно-реляционная система управления базами 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
- Docker – программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации.

Реализованное в рамках курсовой работы web-приложение полностью отвечает потребностям предметной области и выполняет все выявленные функциональные требования, а именно:

- CRUD для моделей «Командир экипажа», «Второй пилот», «Штурман» и «Бортпроводник»
- Просмотр информации обо всех созданных моделях
- Авторизация и регистрация в web-приложении
- Предоставление результатов выполнения запросов
- Предоставление отчетов по авиакомпаниям

## СПИСОК ЛИТЕРАТУРЫ

1. Введение — Vue.js [Электронный ресурс]. – URL: <https://ru.vuejs.org/v2/guide/> (дата обращения 03.06.2020)
2. Django REST framework [Электронный ресурс]. – URL: <https://www.django-rest-framework.org/> (дата обращения 03.06.2020)
3. Современный учебник JavaScript [Электронный ресурс]. – URL: <https://learn.javascript.ru/> (дата обращения 03.06.2020)
4. Документация PostgreSQL – <https://postgrespro.ru/docs/postgresql/9.6/intro-what-is> (дата обращения: 27.06.2020)
5. Serializing Django objects – <https://docs.djangoproject.com/en/3.0/topics/serialization/> (дата обращения: 27.06.2020)
6. ViewSets – <https://www.django-rest-framework.org/api-guide/viewsets/> (дата обращения: 27.06.2020)
7. Class-based Views - <https://www.django-rest-framework.org/api-guide/views/> (дата посещения 28.06.2020)

# **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 Flight, AirCarrier, Crew, TransitLanding, Route, Plane, CrewCommander, SecondPilot,
Steward, Navigator
from .serializers import FlightSerializer, AirCarrierSerializer, CrewSerializer, CrewCommanderSerializer, \
    CrewFIOCommanderSerializer, TransitLandingSerializer, RouteSerializer, PlaneSerializer, \
    SecondPilotSerializer, SecondPilotFIOSerializer, StewardSerializer, StewardFIOSerializer, \
    NavigatorSerializer, NavigatorFIOSerializer

class FlightViewSet(viewsets.ModelViewSet):
    """Отображение для модели Рейс"""
    queryset = Flight.objects.all()
    serializer_class = FlightSerializer

class AirCarrierViewSet(viewsets.ModelViewSet):
    """Отображение для модели Авиаперевозчик"""
    queryset = AirCarrier.objects.all()
    serializer_class = AirCarrierSerializer

class CrewViewSet(viewsets.ModelViewSet):
    """Отображение для модели Экипаж"""
    queryset = Crew.objects.all()
    serializer_class = CrewSerializer

class RouteViewSet(viewsets.ModelViewSet):
    """Отображение для модели Маршрут"""
    queryset = Route.objects.all()
    serializer_class = RouteSerializer

class TransitViewSet(viewsets.ModelViewSet):
    """Отображение для модели Транзитная посадка"""
    queryset = TransitLanding.objects.all()
    serializer_class = TransitLandingSerializer

class PlaneViewSet(viewsets.ModelViewSet):
    """Отображение для модели Самолет"""
    queryset = Plane.objects.all()
    serializer_class = PlaneSerializer

class CrewCommanderViewSet(viewsets.ModelViewSet):
    """Отображение для модели Командир экипажа"""
    queryset = CrewCommander.objects.all()

    def get_serializer_class(self):
        if self.action == 'update':

```

```

        return CrewFIOCommanderSerializer
    elif self.action != 'update':
        return CrewCommanderSerializer

class SecondPilotViewSet(viewsets.ModelViewSet):
    """Отображение для модели Второй пилот"""
    queryset = SecondPilot.objects.all()

    def get_serializer_class(self):
        if self.action == 'update':
            return SecondPilotFIOSerializer
        elif self.action != 'update':
            return SecondPilotSerializer

class StewardViewSet(viewsets.ModelViewSet):
    """Отображение для модели Бортпроводник"""
    queryset = Steward.objects.all()

    def get_serializer_class(self):
        if self.action == 'update':
            return StewardFIOSerializer
        elif self.action != 'update':
            return StewardSerializer

class NavigatorViewSet(viewsets.ModelViewSet):
    """Отображение для модели Штурман"""
    queryset = Navigator.objects.all()

    def get_serializer_class(self):
        if self.action == 'update':
            return NavigatorFIOSerializer
        elif self.action != 'update':
            return NavigatorSerializer

"""Запросы к курсовой работе"""

class Query3(APIView):
    """Определить наличие свободных мест на заданный рейс."""

    def get(self, request):
        number = request.GET.get('number')
        plane = str(Flight.objects.filter(id=number)[0].plane)
        number_of_seats = Plane.objects.filter(id=plane)[0].number_of_seats
        number_of_tickets = Flight.objects.filter(id=2)[0].number_of_tickets
        results = number_of_seats - number_of_tickets
        return Response({'result': results})

class Query4(APIView):
    """Определить количество самолетов, находящихся в ремонте."""

    def get(self, request):

```

```

    results = Plane.objects.filter(on_repairing=True).count()
    return Response({'result': results})

class Query5(APIView):
    """Определить количество работников аэропорта"""

    def get(self, request):

        crew_commander = CrewCommander.objects.count()
        second_pilot = SecondPilot.objects.count()
        navigator = Navigator.objects.count()
        stewards = Steward.objects.count()
        all_employees = crew_commander+second_pilot+navigator+stewards
        return Response({'result': all_employees})

class AirCarrierReport(APIView):
    """отчет о бортах компании-владельца по маркам с характеристикой марки"""

    def get(self, request):

        aircarrier = request.GET.get('aircarrier')
        planes = Plane.objects.values("number", 'number_of_seats', 'on_repairing', 'speed',
'type').filter(air_carrier=aircarrier)
        all_planes = planes.count()
        planes_by_type =
Plane.objects.values('type').filter(air_carrier=aircarrier).order_by('type').annotate(Count('type'))
        results = {
            'planes': planes,
            'all_planes': all_planes,
            'planes_by_type': planes_by_type,
        }
        return Response({'result': results})

# Create your views here.

```