**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 Мобильные и сетевые технологии

Направление подготовки (специальность) Мобильные и сетевые технологии

О Т Ч Е Т

По курсовой работе

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

Обучающийся: Воронов Алексей Сергеевич гр. К3340

Руководитель*:* Говоров А.И.

Оценка \_\_\_\_

Дата \_\_\_\_

Санкт-Петербург

20 \_\_\_\_

#### СОДЕРЖАНИЕ

**ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ, СОКРАЩЕНИЯ**  3

**ВВЕДЕНИЕ** ?

**1. НАЗВАНИЕ ПЕРВОЙ ГЛАВЫ** ?

**1.1. Название раздела** ?

**1.2. Название раздела** ?

**1.3. Название раздела** ?

…

**Выводы по первой главе** ?

**2. НАЗВАНИЕ ВТОРОЙ ГЛАВЫ**  ?

**2.1. Название раздела** ?

**2.2. Название раздела** ?

**2.3. Название раздела** ?

…

**Выводы по второй главе**  ?

**…**

**ЗАКЛЮЧЕНИЕ** ?

**СПИСОК ЛИТЕРАТУРЫ** ?

**ПРИЛОЖЕНИЕ.**

**1.** Название приложения ?

**2**. Название приложения ?

…

#### ВВЕДЕНИЕ

Разработка web-приложений является большой частью современной IT-индустрии. Сегодня практически все компьютеры (настольные и мобильные) подключены к сети Интернет и в основном используют один из ее сервисов. Первоначально данный сервис (web-сеть) использовался только для связывания и предоставления статической информации. Однако в настоящее время он стал платформой для удаленного использования специальных прикладных программ – web-приложений. Если ранее, прежде чем использовать приложение на локальном компьютере, требовалось его устанавливать, то сейчас приложение можно запускать с помощью web- браузера, просто указав его адрес. При этом само приложение выполняется на удаленном компьютере (сервере), а пользователь может работать с ним на своем компьютере с помощью web-браузера.

В связи с активным развитием сети Интернет, web-сети и изменением подхода к использованию приложений специалисты в области информационных технологий должны уметь разрабатывать такие web- приложения.

Создание web-приложений требует от специалистов по информационным технологиям, помимо умения программировать на каком-либо универсальном языке (С#, Java, Python, Ruby и т. п.), знать основные стандарты сети Интернет, такие как: URL, HTTP, HTML, CSS и JavaScript.

Целью данной курсовой работы является создание web-приложения согласно выбранному варианту с использованием современных средств web-разработки, таких как Django REST framework, Vue.js и Muse-ui.

Задачи, которые должны быть выполнены в рамках курсовой работы:

* Изучить предметную область
* Выявить функциональные требования
* Разработать web-приложение с использованием предложенных средств
* Провести контейнеризацию и оркестрацию web-приложения средствами Docker

# ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

## Описание варианта

Реализовать сайт используя вышеуказанные технологии, в соответствии с практическим заданием.

Задание 5

Создать программную систему, предназначенную для работников приемной комиссии колледжа. Она должна обеспечивать хранение, просмотр и изменение сведений об абитуриентах. Результатом работы приемной комиссии должен быть список абитуриентов, зачисленных в колледж.

Секретарь приемной комиссии регистрирует абитуриентов. Для каждого абитуриента в базу данных заносятся следующие сведения: фамилия, имя, отчество, паспортные данные, какое учебное заведение, где и когда окончил, наличие золотой или серебряной медали, название специальности, на которые поступает абитуриент. При подаче заявления абитуриент указывает форму обучения (очная, очно-заочная (вечерняя, заочная), поступление на бюджет или контракт. Абитуриент может поступать вне конкурса (инвалиды, сироты). Также существуют абитуриенты-целевики, которые поступают по договорам с направляющими организациями, и обучаются на коммерческой основе.

Абитуриенты, поступающие на базе 9 классов, участвуют в конкурсе аттестатов. Для них указывается информация по 4-м профильным дисциплинам и средний балл по всем остальным дисциплинам аттестата. На основе этих данных строится рейтинг абитуриентов.

Абитуриенты, поступающие на базе 11 классов, предоставляют сертификаты ЕГЭ по 2 дисциплинам, на основе чего строится рейтинг абитуриентов. Конкурс для абитуриентов на базе 9 и 11 классов раздельный, т.к. они поступают на разные курсы.

Абитуриент может не только подать, но и забрать документы, а также перевести их на другую специальность.

Известно количество мест на каждый факультет. Приемная комиссия по результатам экзаменов должна сформировать списки абитуриентов, зачисленных в колледж. Секретарю приемной комиссии могут потребоваться следующие сведения:

* Список абитуриентов, подавших заявление на заданную специальность.
* Количество абитуриентов, подавших заявления на каждую специальность покаждой форме обучения на бюджет (или контракт).
* Количество абитуриентов на базе 9 и 11 классов, поступающих на бюджет(или контракт).
* Общее количество поданных заявлений ежедневно.
* Конкурс на каждую специальность по каждой форме бучения на бюджет.

Необходимо предусмотреть возможность получения документа, представляющего собой сгруппированный по заданной специальности список абитуриентов по заданной форме обучения, зачисленных в колледж, с указанием набранных ими баллов по аттестату. Отчет должен содержать проходной балл по специальности в целом, а также количество абитуриентов, поступающих на специальность.

## Описание предметной области и функциональные требования

Согласно варианту, в качестве предметной области выбран абстрактный университет. Информационная система для данной сферы занимается управлением списка абитуриентов, добавление новых абитуриентов, составление списков по определенным параметрам. Данные об абитуриентах содержат личную информацию, которая не должна быть доступна любому пользователю web-приложения, следовательно, необходимо реализовать авторизацию и в системе. Таким образом, нужно было разработать web- приложение, которое должно обладать следующим функционалом:

Для администратора:

1. Авторизация в web-приложении.
2. Просмотр информации по конкретному абитуриенту.
3. Возможность добавления /удаления абитуриентов.
4. Получение списка абитуриентов по определенным параметрам.
5. Получение документа со списком по выбранным параметрам.
6. Возможность перевода абитуриентов на другую специальность.

# ОПИСАНИЕ СЕРВЕРНОЙ ЧАСТИ

## Модель данных

Согласно варианту и выявленным функциональным требованиям была создана модель данных, представленная на рисунке 1 (Приложение 1)

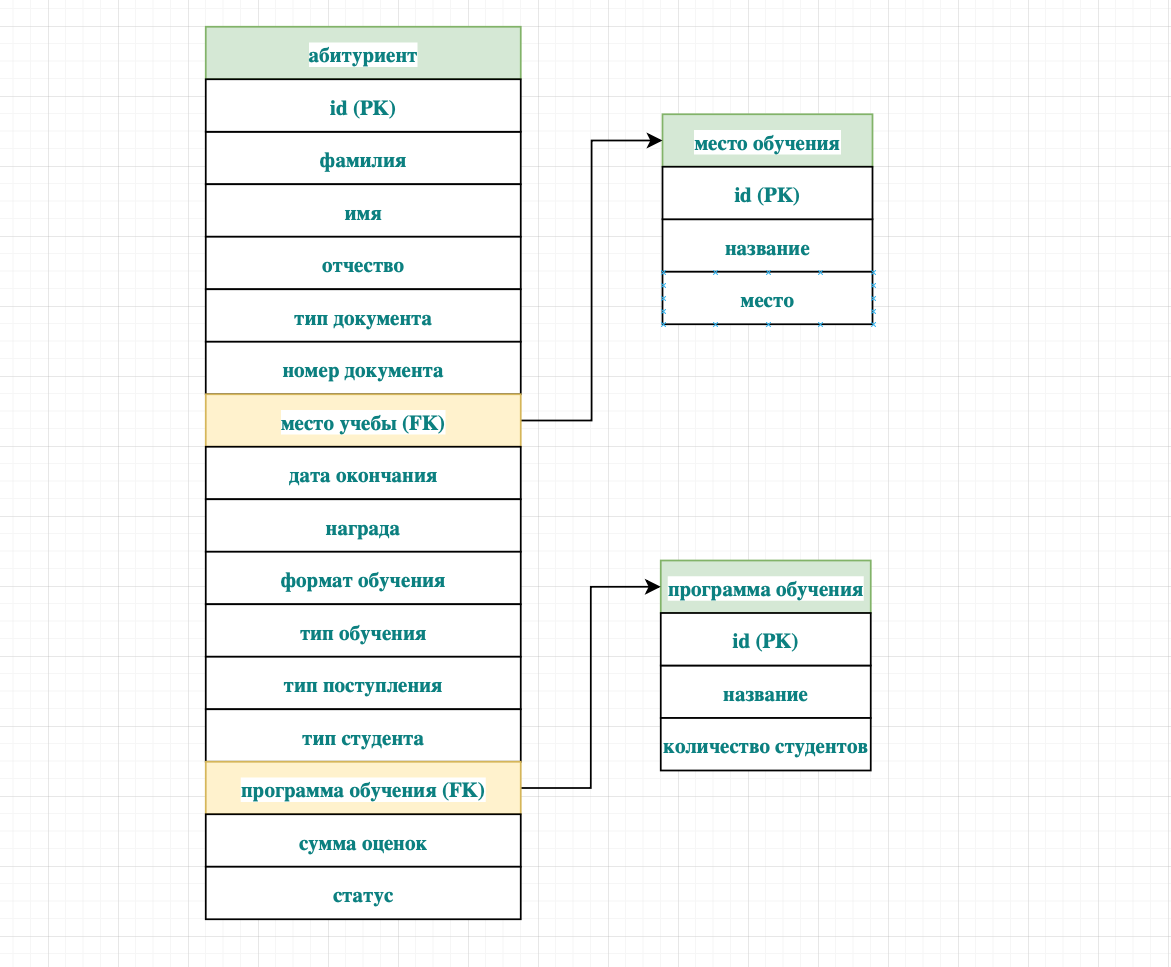


Рисунок 1

Для управления базой данных была выбрана PostgreSQL – объектно-реляционная система управления базами данных (ОРСУБД, ORDBMS), основанная на POSTGRES

После подключения созданной на сервере Postgre базы данных к бэкенду Django были созадны следующие модели в файле models.py:

* Study\_Program – Программа обучения – данная модель содержит информацию о названии программы и количество мест.
* Study\_Place – Место обучения – данная модель содержит информацию об месте обучения абитуриента до поступления: место и тип учебного заведения
* Abiturient – Абитуриент – данная модель содержит сведения об абитуриенте, такие как: программа обучения, место обучения, фамилия, имя, отчество, тип документа, номер документа, место учебы, дата окончания, награда, формат обучения, тип обучения, тип поступления, тип студента, программа обучения, сумма оценок, статус.

## Сериализация и создание отображений

Среда сериализации Django предоставляет механизм для «перевода» моделей Django в другие форматы. Обычно эти другие форматы основаны на тексте и используются для отправки данных Django, но сериализатор может обрабатывать любой форма. Использование сериализаторов в данной работе необходимо для того, чтобы серверная часть, написанная на Django REST, могла без проблем обмениваться данными с клиентской частью, написанной на Vue.js. Листинг кода всех сериализаторов приведен в приложении

Для создания отображений, отвечающих за выполнение запросов был выбран класс APIView. Использование класса APIView во многом аналогично использованию обычного класса View, как обычно, входящий запрос отправляется соответствующему методу-обработчику, например .get() или .post(). Листинг кода с созданием отображений приведен в приложении.

## Разработанные интерфейсы в Django REST

Для создания отображений использовался класс APIView (Приложение 2)

, которыйобладает встроенными атрибутами для последующего создания функций CRUD для модели. Согласно варианту, были созданы следующие отображения:

* class Study\_Programs – отображение для моделей «Программа обучения»
* class Study\_Places – отображение для моделей «Место обучения»
* class Abiturients – отображение для модели «Абитуриент»
* class AbiturientSetStatus – отображение для изменения статуса модели «Абитуриента»
* class AbiturientSetProgram – отображение для изменения программы обучения модели «Абитуриента»

Абитуриенты

Изображение выглядит как снимок экрана

Автоматически созданное описание

Рисунок 2 – Абитуриенты

Программы обучения Изображение выглядит как снимок экрана

Автоматически созданное описание

Рисунок 3 – Программы обучения

Места обучения Изображение выглядит как снимок экрана

Автоматически созданное описание

Рисунок 4 – Места обучения

Изменение статуса абитуриента Изображение выглядит как снимок экрана

Автоматически созданное описание

Рисунок 5 – Изменение статуса абитуриента

Изменение программы обучения абитуриента Изображение выглядит как снимок экрана

Автоматически созданное описание

Рисунок 6 – Изменение статуса абитуриента

Авторизация и получение токена Изображение выглядит как снимок экрана

Автоматически созданное описание

Рисунок 7 – Авторизация и получение токена

# ОПИСАНИЕ КЛИЕНТСКОЙ ЧАСТИ

## Описание средств разработки клиентской части

Для создания клиентской части был использован JavaScript-фреймворк Vue.js и его библиотека Muse-ui для создание интерфейса. 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.

## Разработанные клиентские интерфейсы Vue

Согласно заданию и созданной архитектуре web-приложения, были созданы следующие интерфейсы Vue.

Отображение абитуриентов, есть возможность выбрать параметры и отфильровать по ним абитуриентов, так же отображается количество абитуриентов подходящих под выбранные параметры Изображение выглядит как снимок экрана

Автоматически созданное описание

Рисунок 8 – Отображение абитуриентов

Подробное отображение информации по абитуриенту: отображает все поля доступные для модели абитуриента, так же есть возможность изментить учебную программу или зачислить абитуриента в университет.Изображение выглядит как снимок экрана

Автоматически созданное описание

Рисунок 9 – Подробное отображение информации по абитуриенту

Добавление абитуриента форма для добавления нового абитуриента в список: Изображение выглядит как снимок экрана

Автоматически созданное описание

Рисунок 10 – Добавление абитуриента

Добавление места обучения: форма для добавления нового места обучения Изображение выглядит как снимок экрана

Автоматически созданное описание

Рисунок 11 – Добавление места обучения

Формирование списка абитуриентов: предоставляет возможность сформировать список по выбранным параметрам и сохранить его в формате PDF Изображение выглядит как снимок экрана

Автоматически созданное описание

Рисунок 12 – Список всех абитуриентов Изображение выглядит как снимок экрана

Автоматически созданное описание

Рисунок 13 – Сформированный документ со списком абитуриентов

# КОНТЕЙНЕРИЗАЦИЯ И ОРКЕСТРАЦИЯ

Контейнеризация — это подход к разработке программного обеспечения, при котором приложение или служба, их зависимости и конфигурация (абстрактные файлы манифеста развертывания) упаковываются вместе в образ контейнера. Контейнеризованное приложение может быть протестировано как модуль и развернуто в виде экземпляра контейнера в операционной системе (ОС) текущего узла. Docker — это проект с открытым исходным кодом для автоматизации развертывания приложений в виде переносимых, самодостаточных контейнеров, которые могут работать в облаке или локально.

Оркестрация — это координация взаимодействия нескольких контейнеров. Конечно, можно работать и без оркестрации — никто не запрещает создать контейнер, в котором будут запущены все необходимые процессы. Однако в этом случае приложение лишается гибкости, масштабируемости, а также возникнут вопросы безопасности, поскольку запущенные в одном контейнере процессы не будут изолированы и смогут влиять друг на друга.

Оркестрация позволяет создавать информационные системы из множества контейнеров, каждый из которых отвечает только за одну определенную задачу, а общение осуществляется через сетевые порты и общие каталоги. При необходимости каждый такой контейнер можно заменить другим, что позволяет, например, быстро перейти на другую версию базы данных при необходимости. Конфигурации контейнеров backend (приложение 4), и frontend (приложение 5) можно найти в приложении, а так же файл конфигурации docker-compose (приложение 3)

#### Заключение

Вывод: ходе выполнения лабораторной работы были получены навыки создания web-приложения с помощью web-фреймворка Django REST языка программирования Python, web-фреймворка Vue языка программирования JavaScript и JS Muse-UI.

#### СПИСОК ЛИТЕРАТУРЫ

1. Документация PostgreSQL [Электронный ресурс]. – URL: https://postgrespro.ru/docs/postgresql/9.6/intro-whatis (дата обращения: 30.06.2020)

2. Документация Django [Электронный ресурс]. – URL: https://django.fun/docs/django/ru/3.0/ (дата обращения: 30.06.2020)

3. Vue JS Официальная документация [Электронный ресурс]. – URL: https://ru.vuejs.org/v2/guide/index.html (дата обращения: 30.06.2020)

#### ПРИЛОЖЕНИЯ

Приложение 1. Файл models.py

**from** django.db **import** models  
**from** django.contrib.auth.models **import** User  
**import** json  
*# Create your models here.***class** Study\_Program(models.Model):  
 name = models.TextField(max\_length=100, verbose\_name=**'название'**)  
 count = models.IntegerField(verbose\_name=**'количество студентов'**)  
  
 **class** Meta:  
 verbose\_name = **'программа обучения'** verbose\_name\_plural = **'программы обучения'  
  
class** Study\_Place(models.Model):  
 name = models.TextField(max\_length=100, verbose\_name=**'название'**)  
 place = models.TextField(max\_length=100, verbose\_name=**'место'**)  
  
 TYPE = (  
 (1, **'Школа'**),  
 (2, **'Колледж'**),  
 )  
  
 type = models.PositiveSmallIntegerField(choices=TYPE, default=1)  
  
 **class** Meta:  
 verbose\_name = **'место обучения'** verbose\_name\_plural = **'места обучения'  
  
class** Abiturient(models.Model): *#Модель абитуриента* surname = models.TextField(max\_length=100, verbose\_name=**'фамилия'**)  
 name = models.TextField(max\_length=100, verbose\_name=**'имя'**)  
 secondname = models.TextField(max\_length=100, verbose\_name=**'отчество'**)  
  
 DOC\_TYPE = (  
 (1, **'Паспорт'**),  
 (2, **'Свидетельство о рождении'**),  
 )  
  
 document\_type = models.PositiveSmallIntegerField(choices=DOC\_TYPE, default=1, verbose\_name=**'тип документа'**)  
 document\_number = models.TextField(max\_length=100, verbose\_name=**'номер документа'**)  
  
 study\_place = models.ForeignKey(Study\_Place, verbose\_name=**'место учебы'**, on\_delete=models.CASCADE)  
 study\_date = models.DateField(verbose\_name=**'дата окончания'**)  
  
 AWARD\_TYPE = (  
 (1, **'Нет'**),  
 (2, **'Серебрянная медаль'**),  
 (3, **'Золотая медаль'**),  
 )  
  
 award\_type = models.PositiveSmallIntegerField(choices=AWARD\_TYPE, default=1, verbose\_name=**'награда'**)  
  
 STUDY\_TYPE = (  
 (1, **'очная'**),  
 (2, **'очно-заочная'**),  
 (3, **'заочная'**),  
 )  
  
 study\_type = models.PositiveSmallIntegerField(choices=STUDY\_TYPE, default=1, verbose\_name=**'формат обучения'**)  
  
 CONTRACT\_TYPE = (  
 (1, **'бюджет'**),  
 (2, **'платный'**),  
 )  
  
 contract\_type = models.PositiveSmallIntegerField(choices=CONTRACT\_TYPE, default=1, verbose\_name=**'тип обучения'**)  
  
 ABIT\_TYPE = (  
 (1, **'после 9'**),  
 (2, **'после 11'**),  
 )  
  
 abit\_type = models.PositiveSmallIntegerField(choices=ABIT\_TYPE, default=2, verbose\_name=**'тип поступления'**)  
  
 STUDENT\_TYPE = (  
 (1, **'нет'**),  
 (2, **'целевик'**),  
 (3, **'инвалид'**),  
 (4, **'сирота'**),  
 )  
  
 student\_type = models.PositiveSmallIntegerField(choices=STUDENT\_TYPE, default=1, verbose\_name=**'тип студента'**)  
  
 study\_program = models.ForeignKey(Study\_Program, verbose\_name=**'программа обучения'**, on\_delete=models.CASCADE)  
  
 marks = models.PositiveSmallIntegerField(default=0, verbose\_name=**"сумма оценок"**)  
  
 accepted = models.BooleanField(verbose\_name=**'зачислен'**, default=**False**)  
  
 **class** Meta:  
 verbose\_name = **'студент'** verbose\_name\_plural = **'студенты'**

Приложение 2. Файл views.py

**from** rest\_framework.views **import** APIView  
**from** rest\_framework.response **import** Response  
**from** rest\_framework **import** permissions  
  
**from** abit **import** models  
**from** abit **import** serializers  
  
  
*# Create your views here.***class** Study\_Programs(APIView):  
 permission\_classes = [permissions.IsAuthenticated]  
  
 **def** get(self, request):  
 programs = models.Study\_Program.objects.all()  
 serializer = serializers.Study\_Program\_Serialzer(programs, many=**True**)  
 **return** Response({**'values'**: serializer.data})  
  
 **def** post(self, request):  
 program = serializers.Study\_Program\_Serialzer(data=request.data)  
 **if** program.is\_valid():  
 program.save()  
 **return** Response({**'status'**: **'succes'**})  
 **else**:  
 **return** Response({**'status'**: **'error'**})  
  
  
**class** Study\_Places(APIView):  
 permission\_classes = [permissions.IsAuthenticated]  
  
 **def** get(self, request):  
 places = models.Study\_Place.objects.all()  
 serializer = serializers.Study\_Place\_Serialzer(places, many=**True**)  
 **return** Response({**'values'**: serializer.data})  
  
 **def** post(self, request):  
 place = serializers.Study\_Place\_Serialzer(data=request.data)  
 **if** place.is\_valid():  
 place.save()  
 **return** Response({**'status'**: **'succes'**})  
 **else**:  
 **return** Response({**'status'**: **'error'**})  
  
**class** Abiturients(APIView):  
 permission\_classes = [permissions.IsAuthenticated]  
  
 **def** get(self, request):  
 abiturients = models.Abiturient.objects.all()  
 serializer = serializers.Abitutient\_Serialzer\_Get(abiturients, many=**True**)  
 **return** Response({**'values'**: serializer.data})  
  
 **def** post(self, request):  
 abit = serializers.Abitutient\_Serialzer\_Post(data=request.data)  
 **if** abit.is\_valid():  
 abit.save()  
 **return** Response({**'status'**: **'succes'**})  
 **else**:  
 print(abit.errors)  
 **return** Response({**'status'**: **'error'**})  
  
**class** AbiturientSetStatus(APIView):  
 permission\_classes = [permissions.IsAuthenticated]  
  
 **def** post(self, request, id):  
 dict\_status = {**'true'**: **True**, **'false'**: **False**}  
 abit = models.Abiturient.objects.get(id=id)  
 abit.accepted = dict\_status[request.data[**'status'**]]  
 abit.save()  
 print(abit.accepted)  
 **return** Response({**'status'**: **'succes'**})  
  
**class** AbiturientSetProgram(APIView):  
 permission\_classes = [permissions.IsAuthenticated]  
  
 **def** post(self, request, id):  
 print(request.data[**'id\_program'**])  
 abit = models.Abiturient.objects.get(id=id)  
 abit.study\_program = models.Study\_Program.objects.get(id=request.data[**'id\_program'**])  
 abit.save()  
 print(abit.study\_program)  
 **return** Response({**'status'**: **'succes'**})  
  
  
**class** Succeed\_Abiturients(APIView):  
 permission\_classes = [permissions.IsAuthenticated]  
  
 **def** get(self, request):  
 abiturients = models.Abiturient.objects.get(accepted=**True**)  
 serializer = serializers.Abitutient\_Serialzer(abiturients, many=**True**)  
 **return** Response({**'values'**: serializer.data})

Приложение 3. Файл docker-compose

**version**: **'3'  
  
services**:  
 **db**:  
 **image**: postgres  
 **ports**:  
 - **"5432:5432"  
 environment**:  
 - POSTGRES\_USER=admin  
 - POSTGRES\_PASSWORD=28032803  
 - POSTGRES\_DB=abit  
  
 **backend**:  
 **container\_name**: abit\_backend  
 **build**:  
 **context**: ./project  
 **dockerfile**: Dockerfile  
  
 **command**: python ./manage.py runserver 0.0.0.0:8000  
 **volumes**:  
 - ./project:/project  
 **ports**:  
 - **"8005:8005"  
 depends\_on**:  
 - db  
 **frontend**:  
 **container\_name**: abit\_frontend  
 **build**:  
 **context**: ./abit-vue  
 **dockerfile**: Dockerfile  
  
 **volumes**:  
 - ./project/abit-vue:/project/abit-vue  
 - /abit-vue/node\_modules  
 **ports**:  
 - **"8085:8080"  
 depends\_on**:  
 - backend

Приложение 4. Файл docker для backend

**FROM** python:3.6.9  
  
**ENV *PYTHONUNBUFFERED*** 1  
  
**RUN** mkdir **/**project  
  
**WORKDIR /**project  
  
**COPY** . **/**project  
  
**RUN** pip3 install **-**r requirements.txt

Приложение 5. Файл docker для frontend

**FROM** node:12  
  
**WORKDIR /**abit-vue  
  
**COPY** package**\***.json .**/  
RUN** npm install **--**silent  
**CMD** [**"npm"**, **"start"**]  
  
**COPY** . .