

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Web-программирование

Отчет

Лабораторная работа №3:  
Реализация серверной части приложения средствами  
Django и Django Rest Framework

Выполнил:  
Кривошапкина Айталинка

Группа:  
К33402

Проверил:  
Говоров А. И.

Санкт-Петербург

2021 г.

## **Цель**

Овладеть практическими навыками и умениями реализации web-сервисов средствами Django.

## **Практическое задание**

Реализовать сайт, используя фреймворк Django 3, Django REST Framework, Djoser и СУБД PostgreSQL \*, в соответствии с вариантом задания лабораторной работы - программная система, предназначенная для администратора лечебной клиники.

### **Описание:**

Прием пациентов ведут несколько врачей различных специализаций. На каждого пациента клиники заводится медицинская карта, в которой отражается вся информация по личным данным больного и истории его заболеваний (диагнозы). При очередном посещении врача в карте отражается дата и время приема, диагноз, текущее состояние больного, рекомендации по лечению. Так как прием ведется только на коммерческой основе, после очередного посещения пациент должен оплатить медицинские услуги (каждый прием оплачивается отдельно). Расчет стоимости посещения определяется врачом согласно прейскуранту по клинике.

Для ведения внутренней отчетности необходима следующая информация о врач: фамилия, имя, отчество, специальность, образование, пол, дата рождения и дата начала и окончания работы в клинике, данные по трудовому договору. Для каждого врача составляется график работы с указанием рабочих и выходных дней.

Прием пациентов врачи могут вести в разных кабинетах. Каждый кабинет имеет определенный режим работы, ответственного и внутренний телефон.

## **Ход работы**

1. Реализовать модель базы данных. Всего имеется 10 сущностей.

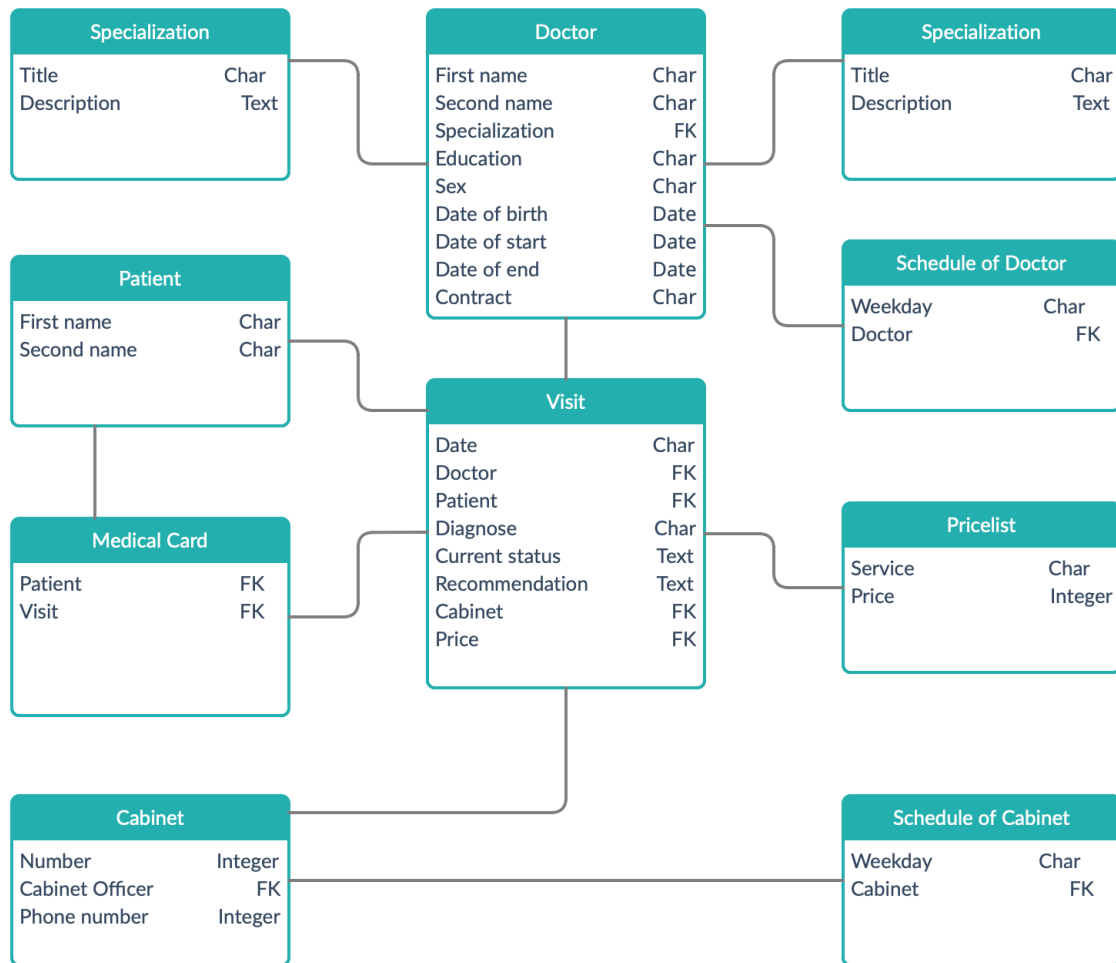


Рисунок 1 - Модель базы данных

Реализация модели Visit в models.py:

```

class Visit(models.Model):
    date = models.DateField(verbose_name='Дата приема',
default=datetime.date.today)
    doctor = models.ForeignKey('Doctor', on_delete=models.CASCADE,
verbose_name='Доктор')
    patient = models.ForeignKey('Patient', on_delete=models.CASCADE,
verbose_name='Пациент')
    diagnose = models.CharField(max_length=120, verbose_name='Диагноз')
    current_status = models.TextField(verbose_name='Текущее состояние
БОЛЬНОГО')
    recommendation = models.TextField(verbose_name='Рекомендации по
лечению')
  
```

```

cabinet = models.ForeignKey('Cabinet', on_delete=models.CASCADE,
verbose_name='Кабинет')

price = models.ForeignKey('PriceList', on_delete=models.CASCADE,
verbose_name='Цена')

def __str__(self):
    return str(self.doctor) + " / " + str(self.date)

```

## 2. Реализовать логику работы API средствами Django REST Framework (используя методы сериализации).

### Реализация VisitSerializer:

```

class VisitSerializer(serializers.ModelSerializer):
    doctor = DoctorSerializer()
    patient = PatientSerializer()
    cabinet = CabinetSerializer()
    price = PriceListSerializer()

    class Meta:
        model = Visit
        fields = "__all__"

```

### Реализация VisitCreateSerializer:

```

class VisitCreateSerializer(serializers.ModelSerializer):
    class Meta:
        model = Visit
        fields = "__all__"

    def validate(self, data):
        if data['doctor'].date_of_start > data['date'] and
        (data['doctor'].date_of_end is None or data['doctor'].date_of_end <
        data['date']):
            raise serializers.ValidationError("Выберите другого доктора или
            дату")

        weekdays_of_doctor = []
        schedule_of_doctors =
        ScheduleOfDoctor.objects.filter(doctor=data['doctor'])
        for item in schedule_of_doctors:
            weekdays_of_doctor.append(item.weekday)
            weekday = weekdays[data['date'].weekday()][0]

            if weekday not in weekdays_of_doctor:
                raise serializers.ValidationError("В этот день недели доктор не
                работает")

```

```
return data
```

## Реализация VisitAPIView:

```
class VisitAPIView(viewsets.ModelViewSet):
    serializer_class = VisitSerializer
    queryset = Visit.objects.all()

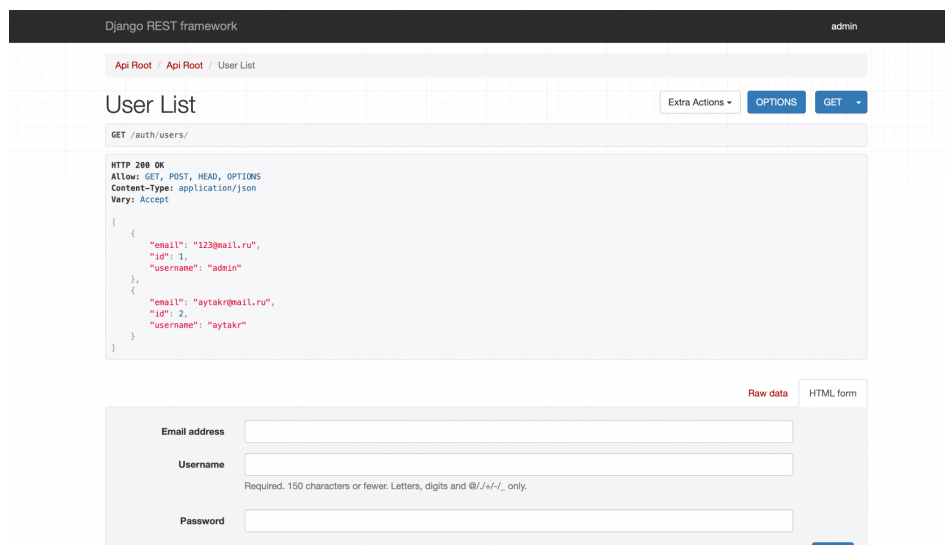
    def get_serializer_class(self):
        if self.action == 'create':
            return VisitCreateSerializer
        else:
            return VisitSerializer
```

3. Подключить регистрацию / авторизацию по токенам / вывод информации о текущем пользователе средствами Djoser.

В `hospital_project/urls.py`:

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('hospital_app.urls')),
    path('', include(router.urls)),
    path('auth/', include('djoser.urls')),
    re_path('auth/', include('djoser.urls.auth_token')),
    path(r'docs/', include('django_mkddocs.urls',
        namespace='documentation')),
]
```

В результате получаем:

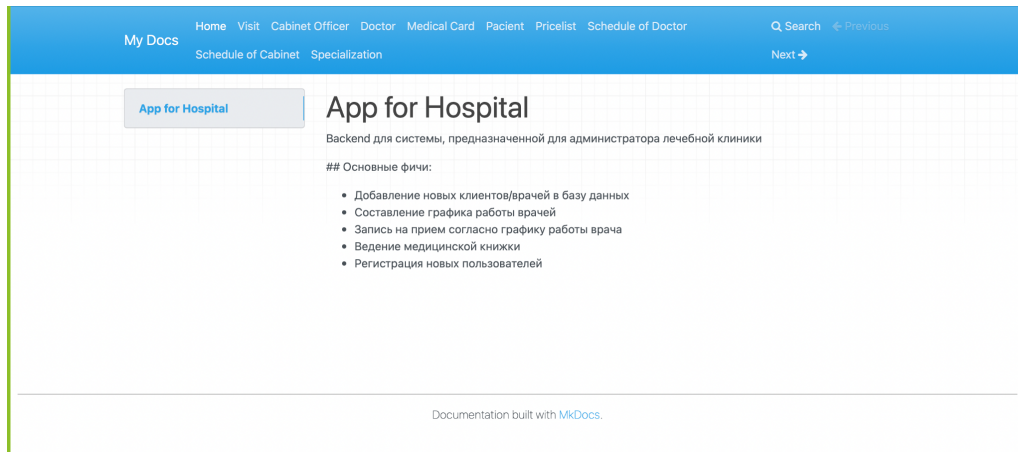


The screenshot shows the Django REST framework admin interface. At the top, there's a header with 'Django REST framework' and 'admin'. Below it, a breadcrumb trail shows 'Api Root / Api Root / User List'. The main content area is titled 'User List' and shows the details of a GET request to '/auth/users/'. The response is a JSON array of two users: 'admin' and 'aytakr'. Below the response, there's a form for creating a new user with fields for 'Email address', 'Username', and 'Password'. The 'Username' field has a note: 'Required. 150 characters or fewer. Letters, digits and @/./+/-/\_ only.' There are also buttons for 'Extra Actions', 'OPTIONS', and 'GET'.

Рисунок 2 - Регистрация

4. Реализовать документацию, описывающую работу всех используемых endpoint-ов из пункта 3 и 4 средствами Read the Docs или MkDocs.

Документация представлена для каждого эндпоинта



*Рисунок 3 - Документация*

## Вывод

В результате данной работы были изучены основы реализации серверной части приложения средствами Django и Django Rest Framework.

]