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

Факультет инфокоммуникационных технологий

Образовательная программа Интеллектуальные системы в гуманитарной сфере

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

О Т Ч Е Т

по курсовой работе по дисциплине «Основы WEB-программирования»

Тема задания: Web-сервис администрирования медицинской клиники

Обучающийся Шипицына Дарья Вадимовна, гр. К3342

Преподаватель дисциплины: Говоров Антон Игоревич, ассистент факультета ИКТ университета ИТМО

Оценка за курсовую работу ____

Подписи членов комиссии:

_____ (Говоров А.И.)
(подпись)

_____ (Чунаев А.В.)
(подпись)

_____ (Антонов М.Б.)
(подпись)

Дата ____

Санкт-Петербург
20 20

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Курсовая работа посвящена реализации программной системы для администратора медицинской клиники. Медицинская клиника – это сложная система, нуждающаяся в упорядочивании данных и ведении учета о пациентах, врачах, приемах, а также финансовый учет.

Цель курсовой работы по дисциплине «Основы Web-программирования»: овладеть практическими навыками реализации Web-сервисов.

Задачи в рамках курсовой работы:

1. Анализ предметной области.
2. Выделение функциональных требований.
3. Проектирование модели данных в соответствии с функциональными требованиями.
4. Выбор средств проектирования.
5. Проектирование серверной части.
6. Проектирование клиентской части.
7. Реализация.

1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

1.1. Предметная область

Предметной областью курсовой работы является процесс управления и администрирования медицинской клиники. Информационная система для данной сферы должна предусматривать следующие возможности: добавление и удаление пациентов, просмотр подробной информации о пациентах, принятие на работу врачей, ведение расписание приемов клиники и ведение финансового учета.

1.2. Функциональные требования

Данные, хранимые в базе данных клиники, строго конфиденциальны, поэтому необходимо предусмотреть авторизацию администраторов клинику и обеспечить недоступность данных неавторизованным пользователям.

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 врачах: фамилия, имя, отчество, специальность, образование, пол, категория, телефон и email. Для каждого врача составляется график работы с указанием рабочих дней и часов.

Администратору клиники могут понадобиться следующие сведения и отчеты:

1. Общее количество приемов у каждого врача.
2. Количество врачей каждой специализации.
3. Стоимость всех приемов у каждого врача.
4. Количество приемов у выбранного врача по выбранной дате.
5. Отчет о суммарной выручке клиники за выбранный период.

Таким образом, в соответствии с вышеперечисленными требованиями, функционал web-сервиса должен включать в себя:

1. Авторизация в приложении.

2. Просмотр, добавление, редактирование и удаление записей пациентов.
3. Просмотр медицинской карты пациентов
4. Просмотр и добавление записей врачей.
5. Просмотр графика врача.
6. Просмотр и добавление записей о приемах.
7. Вывод информации по соответствующим запросам.

Описанный функционал схематически представлен на рисунке 1.

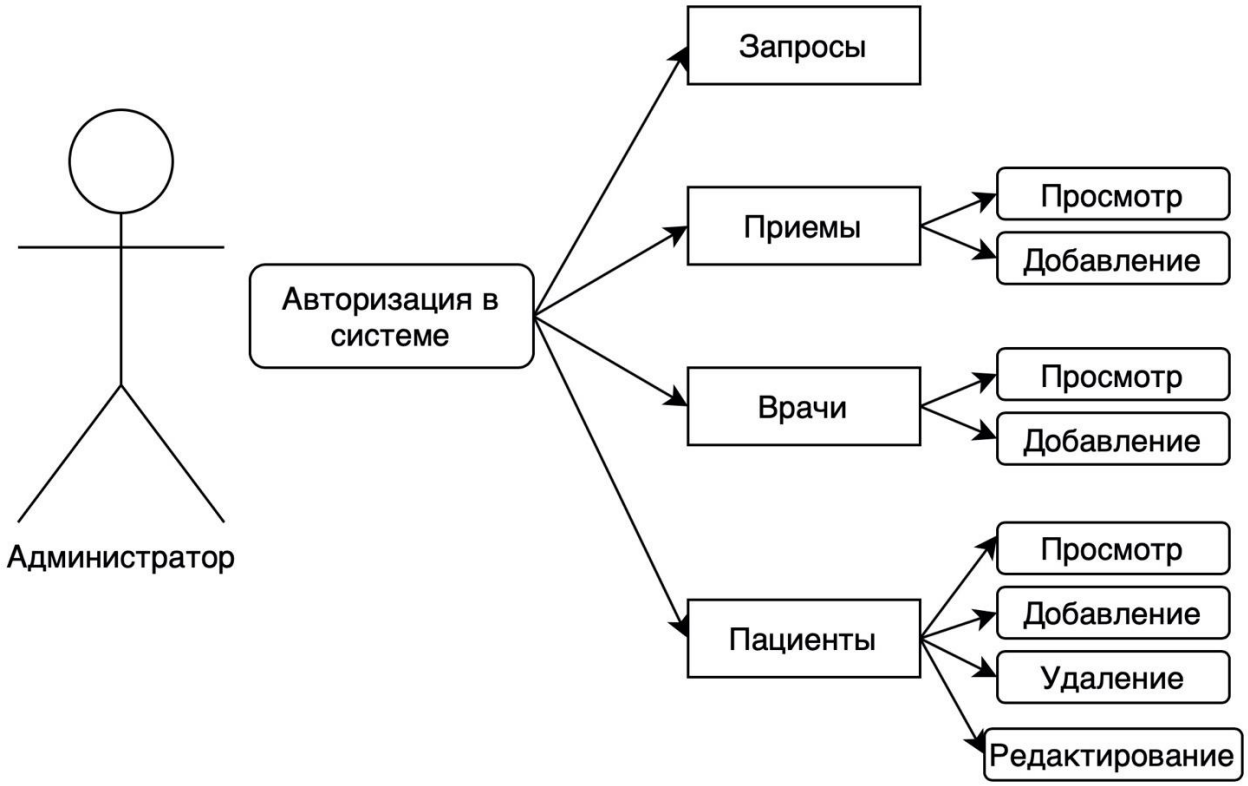

Рисунок 1 – функциональные возможности администратора клиники в реализуемом web-сервисе

2. ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ СЕРВИСА

2.1. Описание архитектуры сервиса

Для web-сервиса, реализуемого в рамках курсовой работы, была использована клиент-серверная архитектура (рис. 2). Архитектура «клиент-сервер» определяет общие принципы организации взаимодействия в сети, где имеются серверы, узлы-поставщики некоторых специфичных функций (сервисов) и клиенты (потребители этих функций) [4].

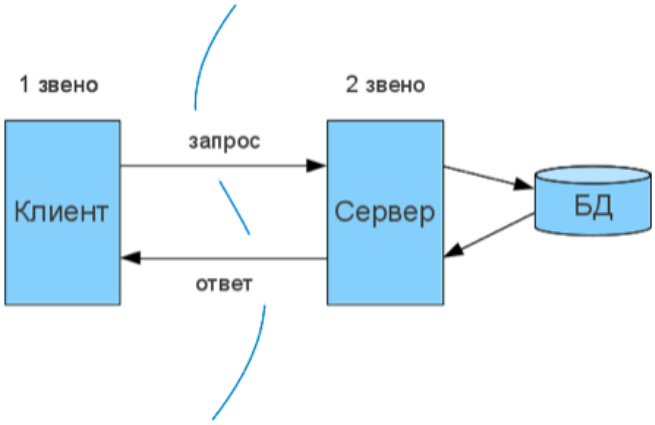

Рисунок 2 – архитектура «Клиент-сервер»

Клиент – это браузер в классической ситуации.

Веб-сервер – это сервер, принимающий HTTP-запросы от клиентов и выдающий им HTTP-ответы. Веб-сервером называют как программное обеспечение, выполняющее функции веб-сервера, так и непосредственно компьютер, на котором это программное обеспечение работает.

База данных – это информационная модель, позволяющая упорядоченно хранить данные об объекте или группе объектов, обладающих набором свойств, которые можно распределить по категориям.

2.2. Модель данных

Следующей задачей проектирования стала разработка модели данных, позволяющей при последующих шагах реализовать весь необходимый функционал.

Сущности разработанной модели:

1. Patient – Пациент (фамилия, имя, отчество, дата рождения, пол, телефон, email).
2. Doctor – Врач (фамилия, имя, отчество, образование, категория, специализация, пол, рабочие часы (M2M), телефон, email).

3. App_times – Рабочие часы (дата приема, время приема).

4. Ассоциативная сущность Schedule – Расписание. Создается при связи объектов из модели Doctor и App_times.

5. PriceList – Прайслист (услуга, цена).

6. Appointment – Прием (пациент, услуга, запись из расписания, диагноз и лечение, оплачено (T/F).

Модель данных представлена на рисунке 3.

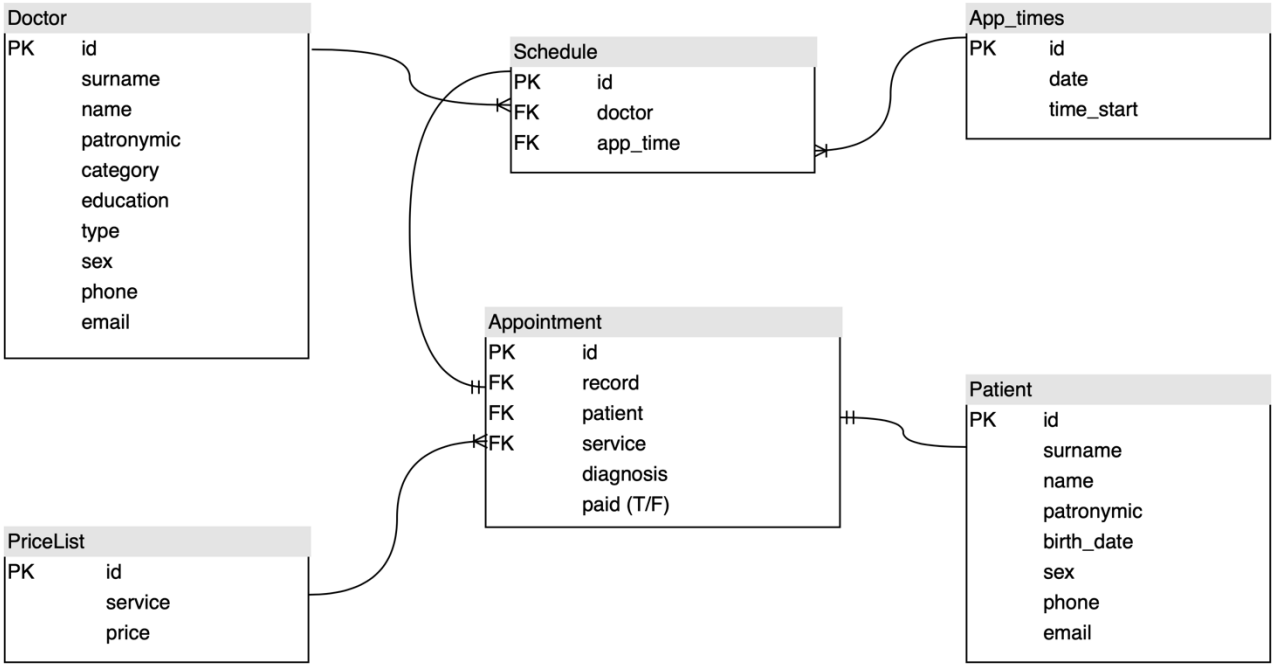

Рисунок 3 – Модель данных

3. ОПИСАНИЕ СЕРВЕРНОЙ ЧАСТИ

3.1. Средства разработки

Для разработки серверной части проекта в соответствии с заданием были использованы следующие средства разработки: среда разработки PyCharm, язык программирования Python, фреймворк Django REST Framework. Django Rest Framework (DRF) — это библиотека, которая работает со стандартными моделями Django для создания гибкого и мощного API для проекта [1].

API DRF состоит из 3-х слоев: сериализатора, вида и маршрутизатора [2].

Сериализатор: преобразует информацию, хранящуюся в базе данных и определенную с помощью моделей Django, в формат, который легко и эффективно передается через API.

Вид (ViewSet): определяет функции (чтение, создание, обновление, удаление), которые будут доступны через API.

Маршрутизатор: определяет URL-адреса, которые будут предоставлять доступ к каждому виду.

3.2. Разработка интерфейсов

Проект был начат в среде разработки PyCharm, после чего к нему были подключены все необходимые библиотеки и модули для начала работы.

После этого был создан файл, содержащий описание таблиц базы данных, соответствующее схеме базы данных (рис. 3) и представленное в виде класса Python – Model. Ниже представлен пример кода, создающего модель Пациент.

```
class Patient(models.Model):
    surname = models.CharField("Фамилия", max_length=50)
    name = models.CharField("Имя", max_length=50)
    patronymic = models.CharField("Отчество", max_length=50)
    birth_date = models.DateField("Дата рождения", blank=False, default='2000-01-01')
    SEX = (
        ('мужской', 'мужской'),
        ('женский', 'женский'),
    )
    sex = models.CharField("Пол", max_length=7, choices=SEX)
    phone = models.CharField("Номер телефона", max_length=11, blank=False, default="не указан")
    email = models.CharField("Адрес электронной почты", max_length=50, blank=False, default="не указана")

    def __str__(self):
        return self.surname + ' ' + self.name
```

Рисунок 4 – Модель Пациент

Аналогичным способом были созданы и другие модели.

Далее был реализован файл views.py, который отвечает за вывод всей необходимой информации. Методы этого файлы реализованы при помощи rest_framework.views.APIview (рис. 5), а также rest_framework.generics [5] (рис. 6).

```
class Patients(APIView):
    permission_classes = [permissions.IsAuthenticated, ]
    # permission_classes = [permissions.AllowAny, ]

    def get(self, request):
        patients = Patient.objects.all()
        serializer = PatientSerializer(patients, many=True)
        return Response({"data": serializer.data})

    def post(self, request):
        patients = PatientPostSerializer(data=request.data)
        print(patients)
        if patients.is_valid():
            patients.save()
            return Response({"status": "Add"})
        else:
            return Response({"status": "Error"})
```

Рисунок 5 – Реализация получения и добавления информации о пациентах посредством rest_framework.views.APIview

```
class PatientUpdate(generics.UpdateAPIView):
    permission_classes = [permissions.IsAuthenticated, ]
    queryset = Patient.objects.all()
    serializer_class = PatientPostSerializer
```

Рисунок 6 – Реализация обновления информации о пациенте посредством rest_framework.generics

Также в одном из видов был использован модуль DjangoFilterBackend для реализации фильтрации записей в модели Приемов по полям «Фамилия доктора», «Фамилия пациента», «Специализация доктора», «Дата приема». Код представлен на рисунке 7.

```
class Appointments(generics.ListAPIView):
    permission_classes = [permissions.IsAuthenticated, ]
    serializer_class = AppointmentFilterSerializer
    queryset = Appointment.objects.all()
    filter_backends = (filters.SearchFilter, DjangoFilterBackend)
    filter_fields = ('record__doctor__surname', 'patient__surname', 'record__doctor__type', 'record__app_time__date')
```

Рисунок 7 – использование DjangoFilterBackend

Следующим шагом стало создание сериализаторов в файле serializers.py (рис. 8).

```python
class AppointmentSerializer(serializers.ModelSerializer):
    patient = PatientSerializer()
    record = ScheduleSerializer()
    service = PriceSerializer(many=True)

    class Meta:
        model = Appointment
        fields = ('id', 'patient', 'record', 'service', 'paid', 'diagnosis')
```

Рисунок 8 – сериализация информации о приеме

Помимо прочего были прописаны пути доступа к страницам, на которых будет отображена необходимая информация в файле urls.py (рис. 9).

```python
urlpatterns = [
    path('patients/', Patients.as_view()),
    path('patient/<int:pk>/', PatientDetail.as_view()),
    path('app/<int:pk>/', AppointmentDetail.as_view()),
    path('patient/<int:pk>/edit/', PatientUpdate.as_view()),
    path('patient/<int:pk>/delete/', PatientDelete.as_view()),
    path('med_card/', MedCard.as_view()),
    path('doctors/', Doctors.as_view()),
    path('schedule/', ScheduleView.as_view()),
    path('add_doc/', CreateDoctor.as_view()),
    path('app_times/', AppTimesList.as_view()),
    path('apps/', Apps.as_view()),
    path('appointments/', Appointments.as_view()),
    path('app_detail/', AppDetail.as_view()),
    path('add_app/', CreateApp.as_view()),
    path('app/<int:pk>/edit/', AppUpdate.as_view()),
    path('doc_free_time/', DocFreeTime.as_view()),
    path('services/', Services.as_view()),
    path('query_1/', QuerySet1.as_view()),
    path('query_2/', QuerySet2.as_view()),
    path('query_3/', QuerySet3.as_view()),
    path('query_4/', QuerySet4.as_view()),
    path('report/', Report.as_view())
]
```

Рисунок 9 – файл urls.py

3.3. Разработанные интерфейсы в Django REST

Ниже (рис. 10 – рис. 15) представлены некоторые из полученных интерфейсов.

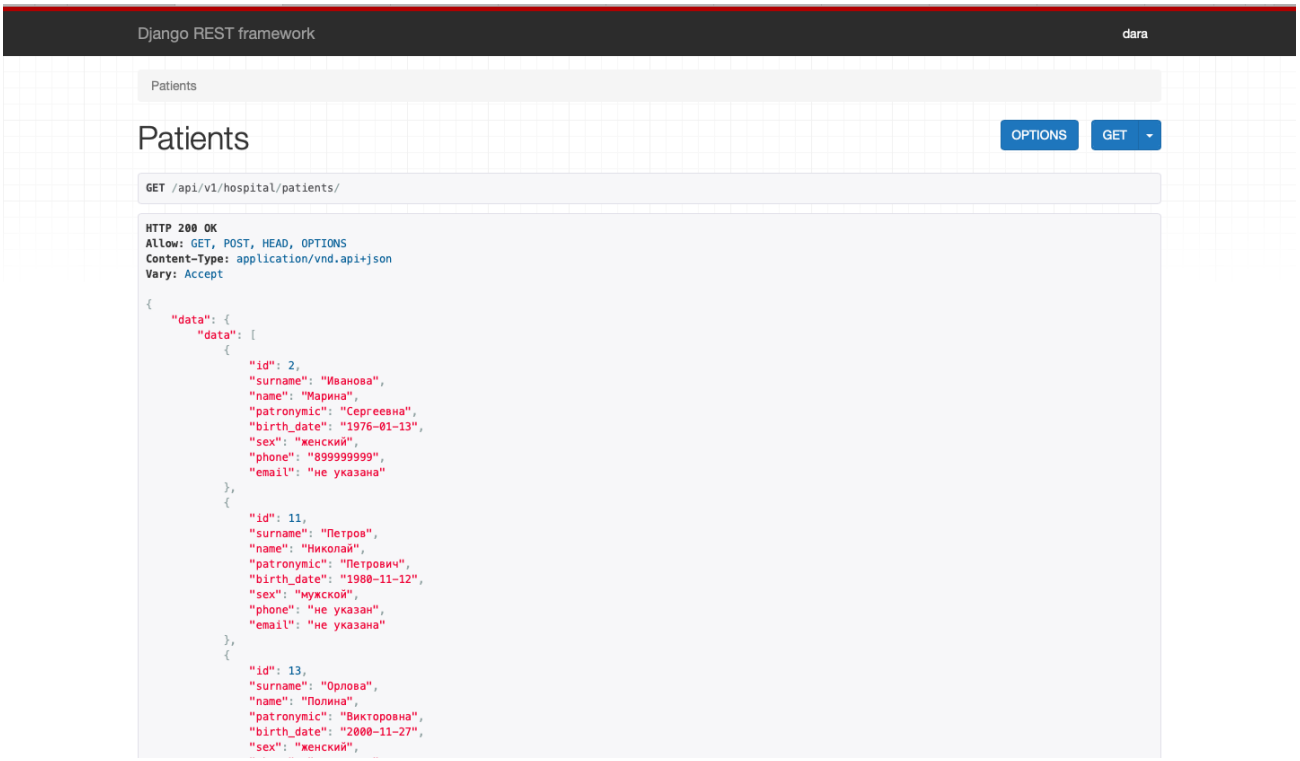

Рисунок 10 – вывод списка всех пациентов

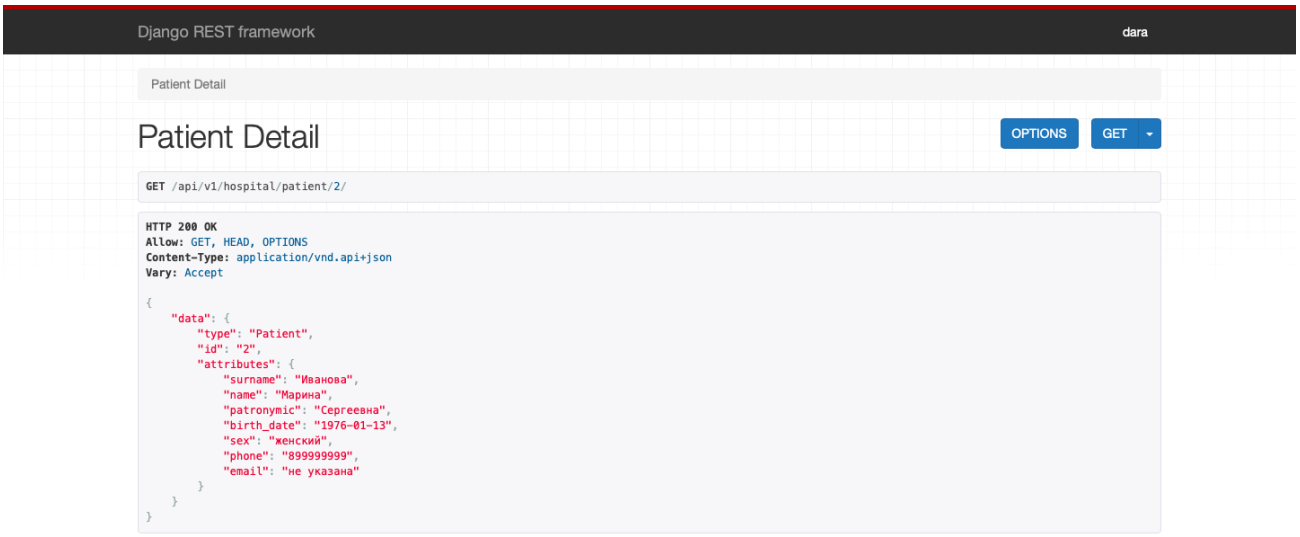

Рисунок 11 – вывод подробной информации о пациенте с id = 2

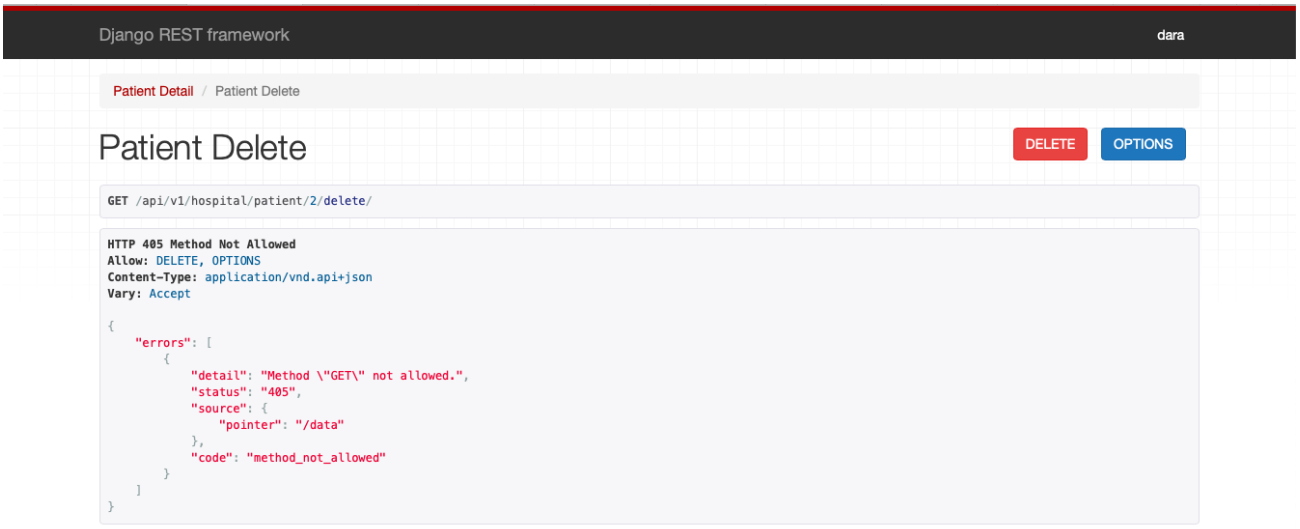

Рисунок 12 – удаление пациента

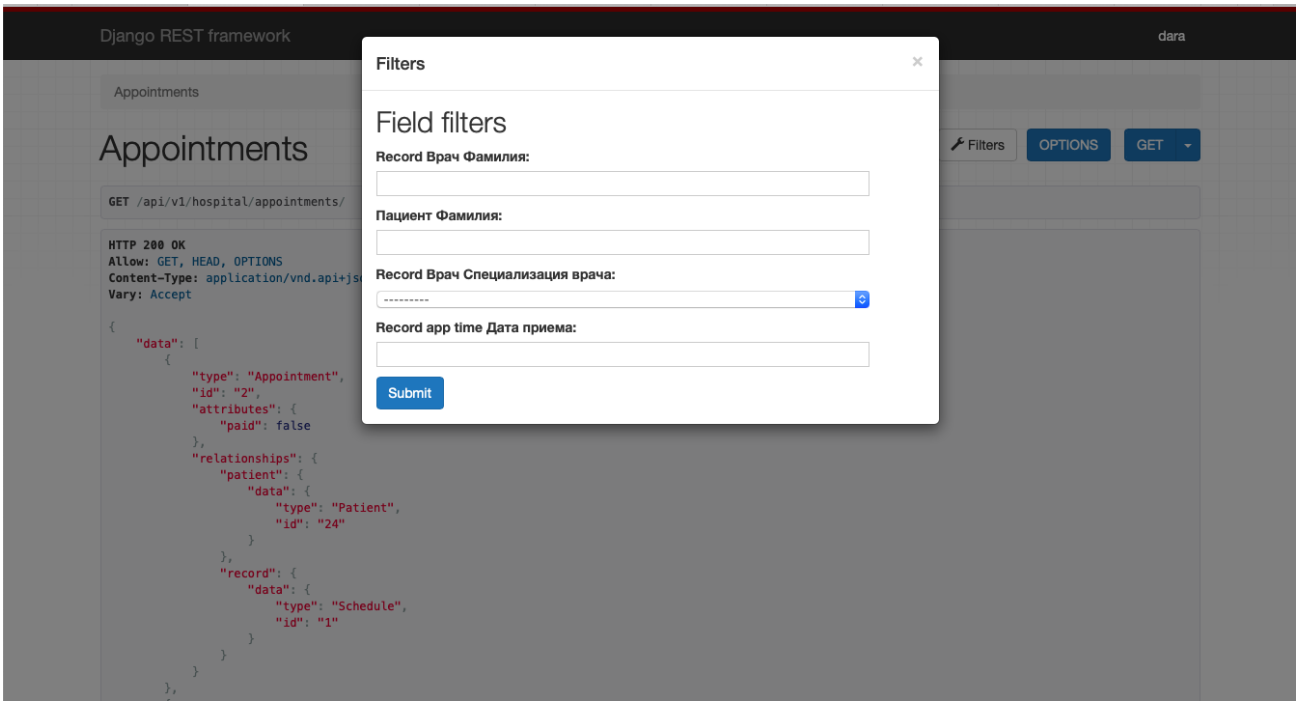

Рисунок 13 – фильтрация записей приемов

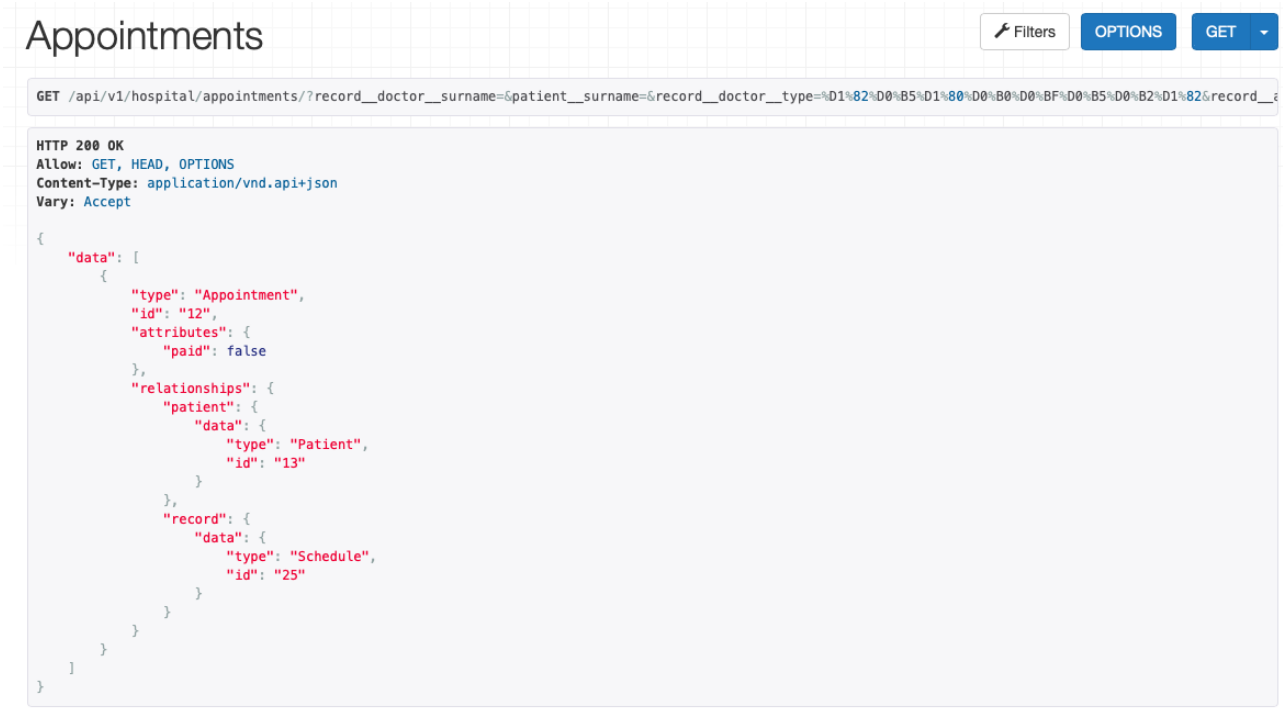

Рисунок 14 – результат фильтрации записей приемов

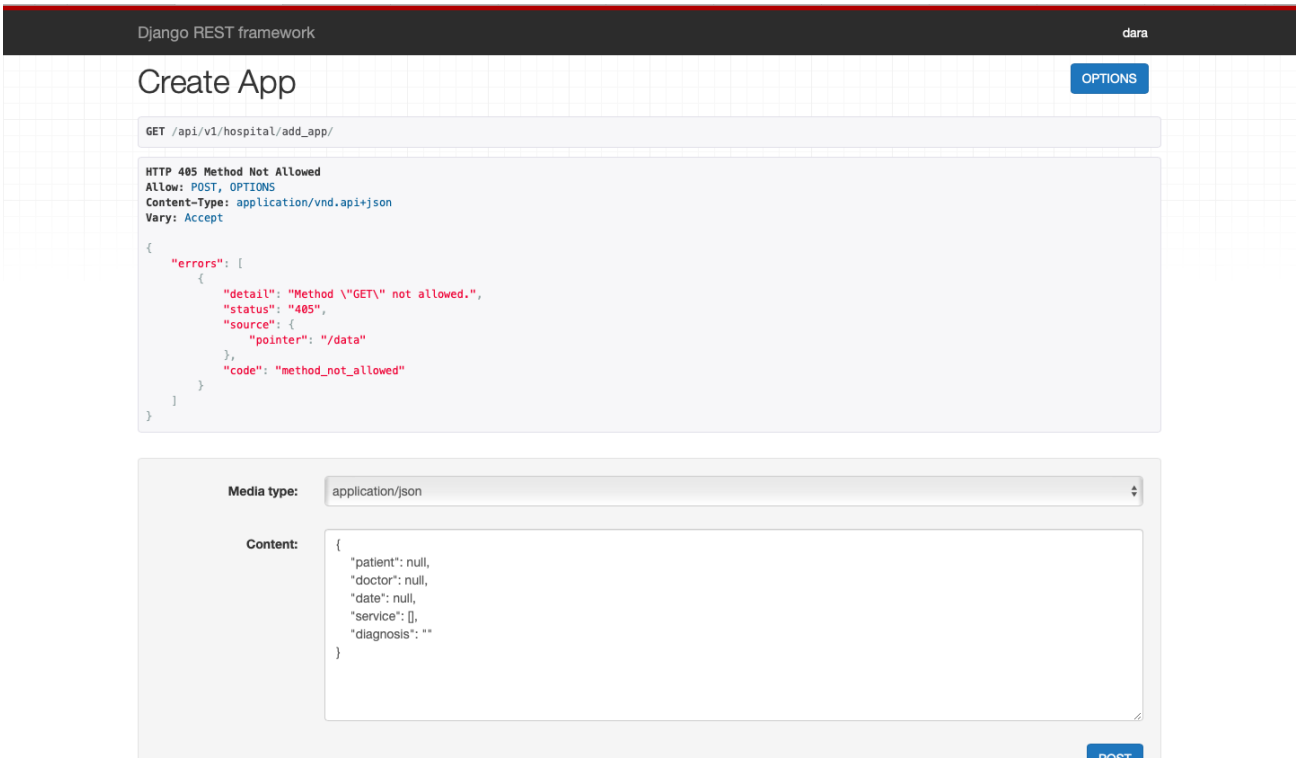

Рисунок 15 – создание записи о приеме

4. ОПИСАНИЕ КЛИЕНТСКОЙ ЧАСТИ

4.1. Средства разработки

Для реализации клиентский части, или части фронтэнда, были использованы такие средства разработки, как Vue.JS и MUSE-UI. Vue.js — это JavaScript библиотека для создания веб-интерфейсов с использованием шаблона архитектуры MVVM (Model-View-View-Model) [3].

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.

На рисунке 16 представлен один из компонентов Vue.

```
<template>
  <mu-container>
      <mu-appbar style="width: 100%;" color="primary">
          Система администрирования клиники
          <mu-button flat slot="right" v-if="!auth" @click="goLogin">Вход</mu-button>
          <mu-button flat slot="right" v-else @click="logout">Выход</mu-button>
          <mu-menu slot="left">
              <mu-button flat>Меню</mu-button>
              <mu-list slot="content">
                  <mu-list-item button @click="goPatient">
                      <mu-list-item-content>
                          <mu-list-item-title>Пациенты</mu-list-item-title>
                      </mu-list-item-content>
                  </mu-list-item>
                  <mu-list-item button @click="goDoctor">
                      <mu-list-item-content>
                          <mu-list-item-title>Врачи</mu-list-item-title>
                      </mu-list-item-content>
                  </mu-list-item>
                   <mu-list-item button @click="goApp">
                      <mu-list-item-content>
                          <mu-list-item-title>Приемы</mu-list-item-title>
                      </mu-list-item-content>
                  </mu-list-item>
                  <mu-list-item button @click="goQueries">
                      <mu-list-item-content>
                          <mu-list-item-title>Запросы</mu-list-item-title>
                      </mu-list-item-content>
                  </mu-list-item>
              </mu-list>
          </mu-menu>
      </mu-appbar>
  </mu-container>

</template>
```

```
<script>

    export default {
        name: 'Home',
        computed: {
              auth() {
                  if (sessionStorage.getItem("auth_token")) {
                      return true
                  }
              }
        },
        methods: {
            goLogin() {
                  this.$router.push({name: "login"})
            },
            goPatient() {
                  this.$router.push({name: "patient"})
            },
            goDoctor() {
                  this.$router.push({name: "doctor"})
            },
            goApp() {
                  this.$router.push({name: "apps"})
            },
            goQueries() {
                  this.$router.push({name: "queries"})
            },
            logout() {
                  sessionStorage.removeItem("auth_token")
                  window.location = '/'
            }
        }
    }
</script>
```

Рисунок 16 – компонент Home.vue

4.2. Реализованные интерфейсы

В соответствии с функциональными требованиями были реализованы следующие интерфейсы:

1. Авторизация в приложении.
2. Просмотр, добавление, редактирование и удаление записей пациентов.
3. Просмотр медицинской карты пациентов
4. Просмотр и добавление записей врачей.

5. Просмотр графика врача.
6. Просмотр и добавление записей о приемах.
7. Вывод информации по соответствующим запросам.

Все они представлены на рисунках 17 – 34.

При переходе на сайт неавторизованного пользователя он попадает на пустую страницу (рис. 17).

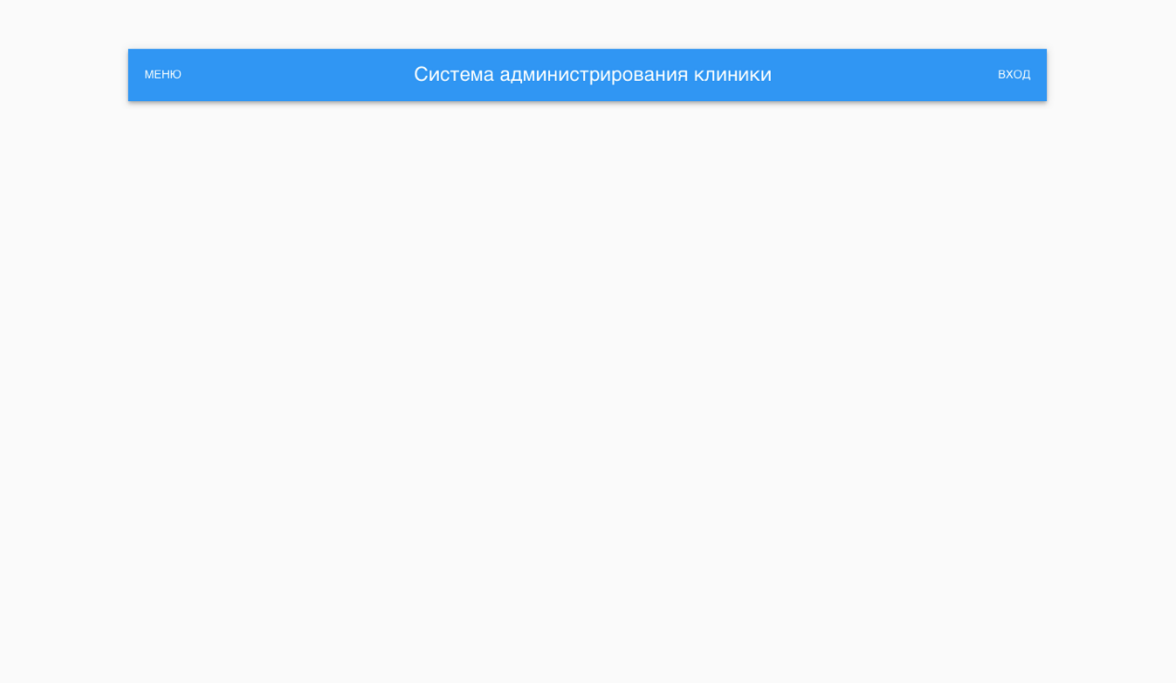

Рисунок 17 – пустая страница при переходе на сайт неавторизованного пользователя

При попытке перейти по любому из пунктов меню неавторизованный пользователь получает ошибку (рис. 18).

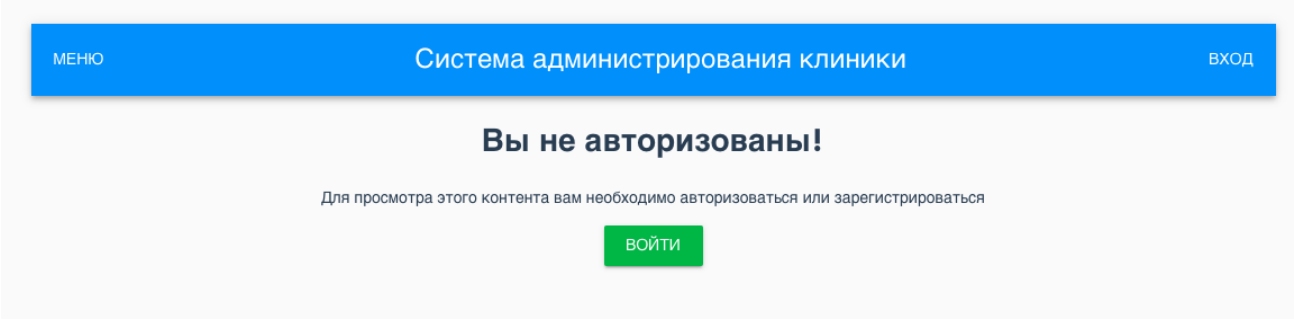

Рисунок 18 – ошибка для неавторизованного пользователя

Далее пользователь может авторизоваться на странице входа (рис. 19).

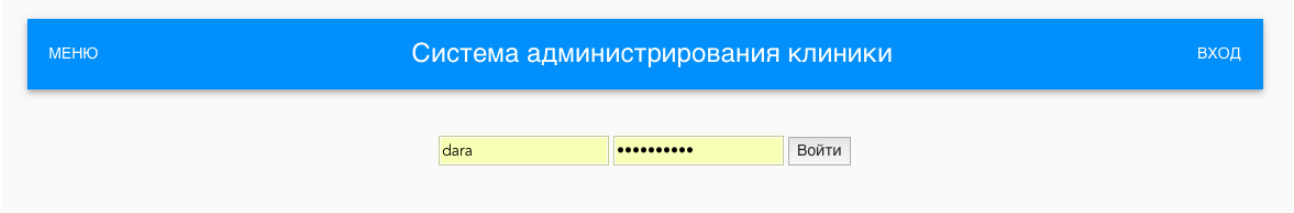

Рисунок 19 – страница входа

Далее авторизованный пользователь попадает на страницу со списком пациентов (рис. 20).

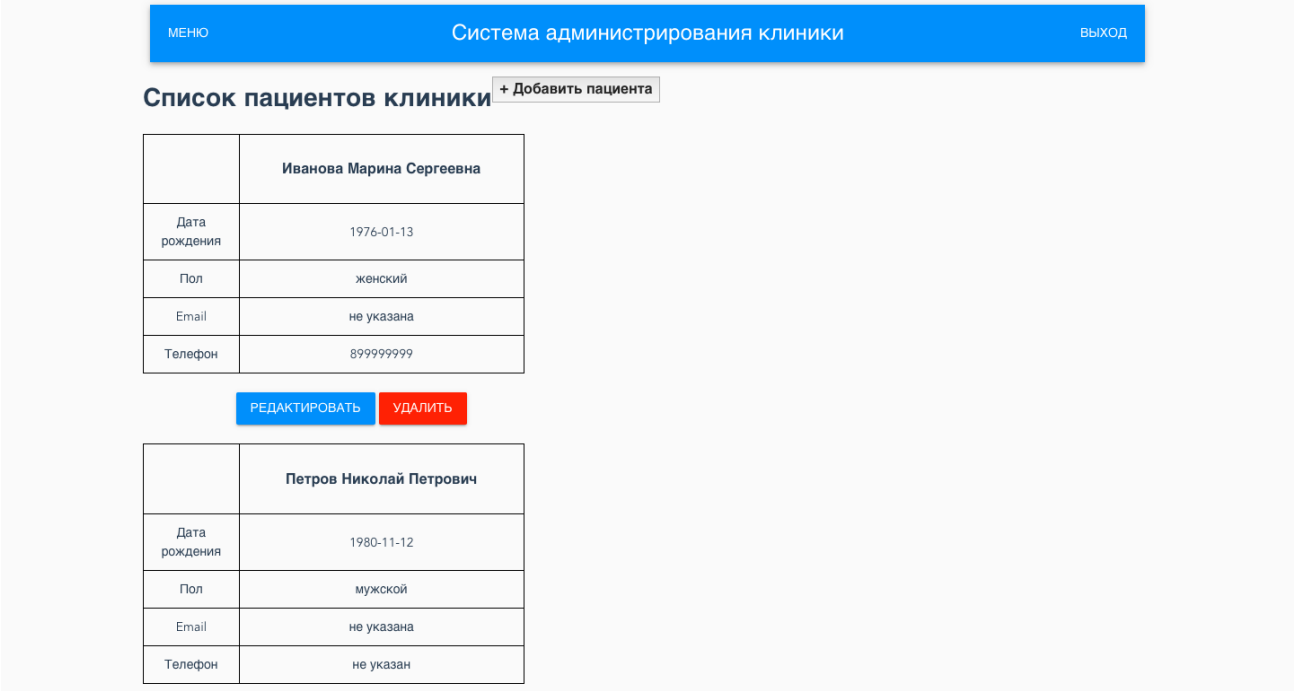

Рисунок 20 – список пациентов

Информация о пациентах представлена в табличном виде, где отображены все атрибуты соответствующей модели Patient. При нажатии на имя пациента будет отображена его медицинская карта с медицинскими записями (рис. 21).

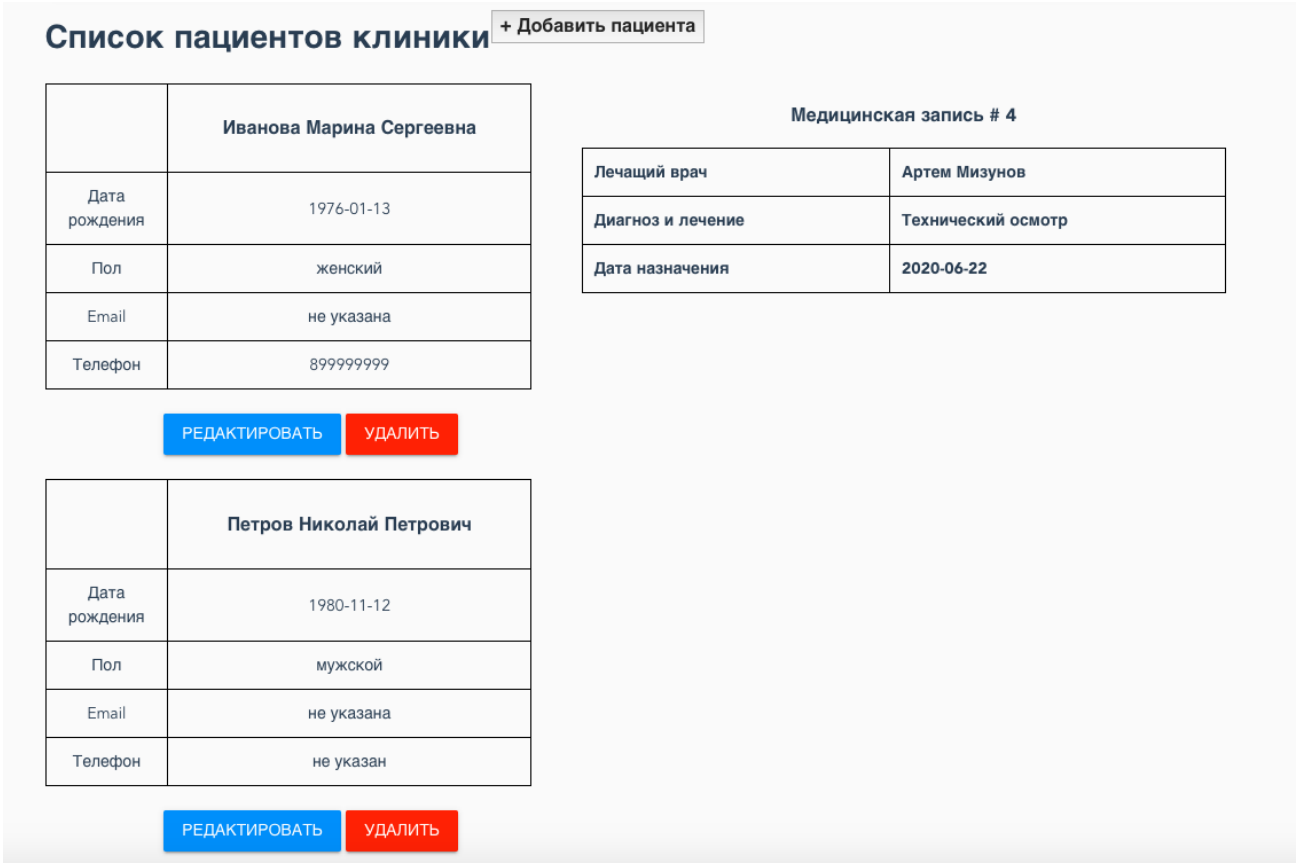

Рисунок 21 – медицинская карточка пациента Петров Николай Петрович

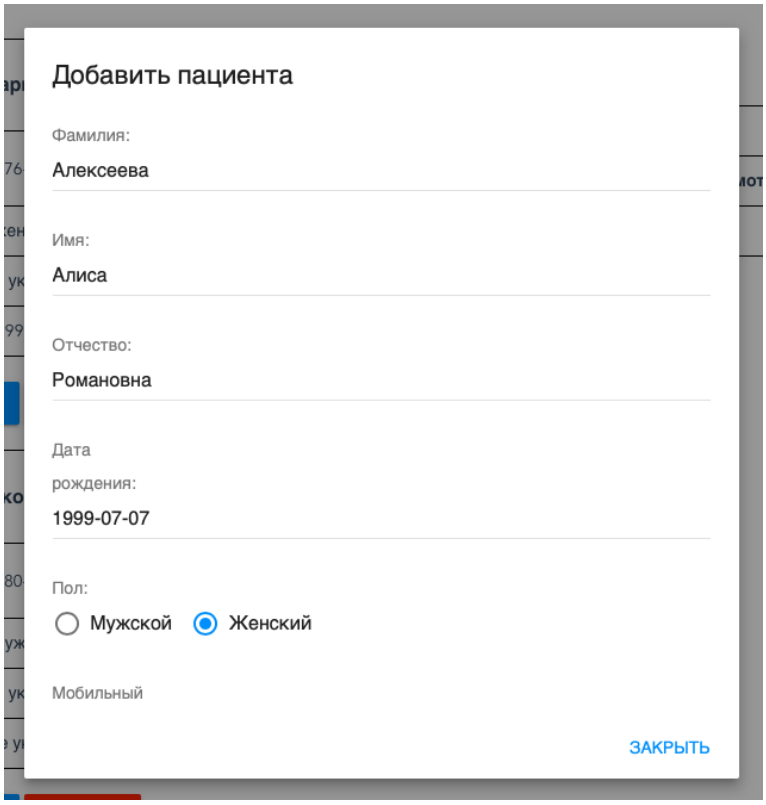

Рисунок 22 – добавление пациента

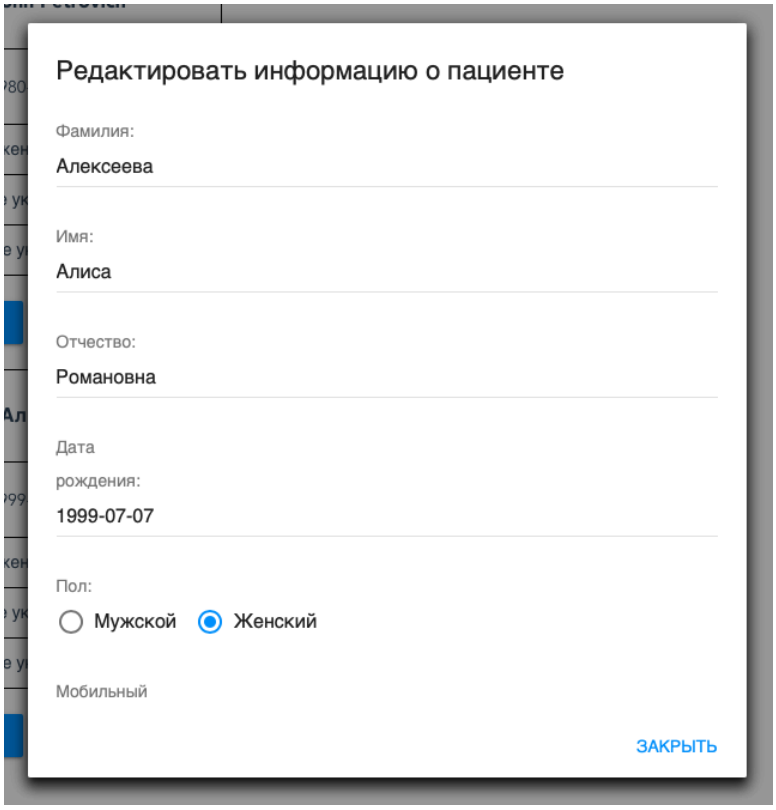

Рисунок 23 – редактирование информации о пациенте

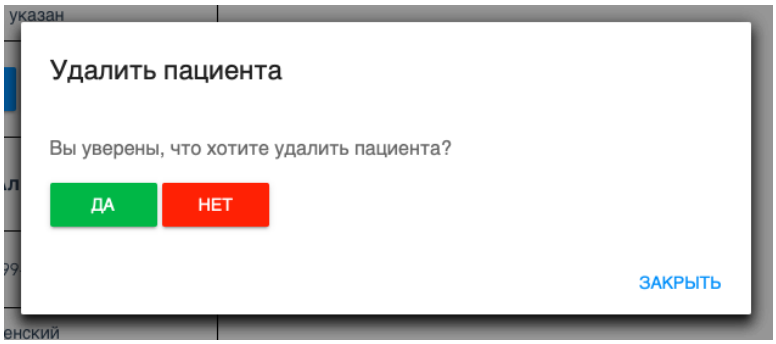

Рисунок 24 – удаление пациента

Информация о врачах аналогично хранится в табличном виде (рис. 25).

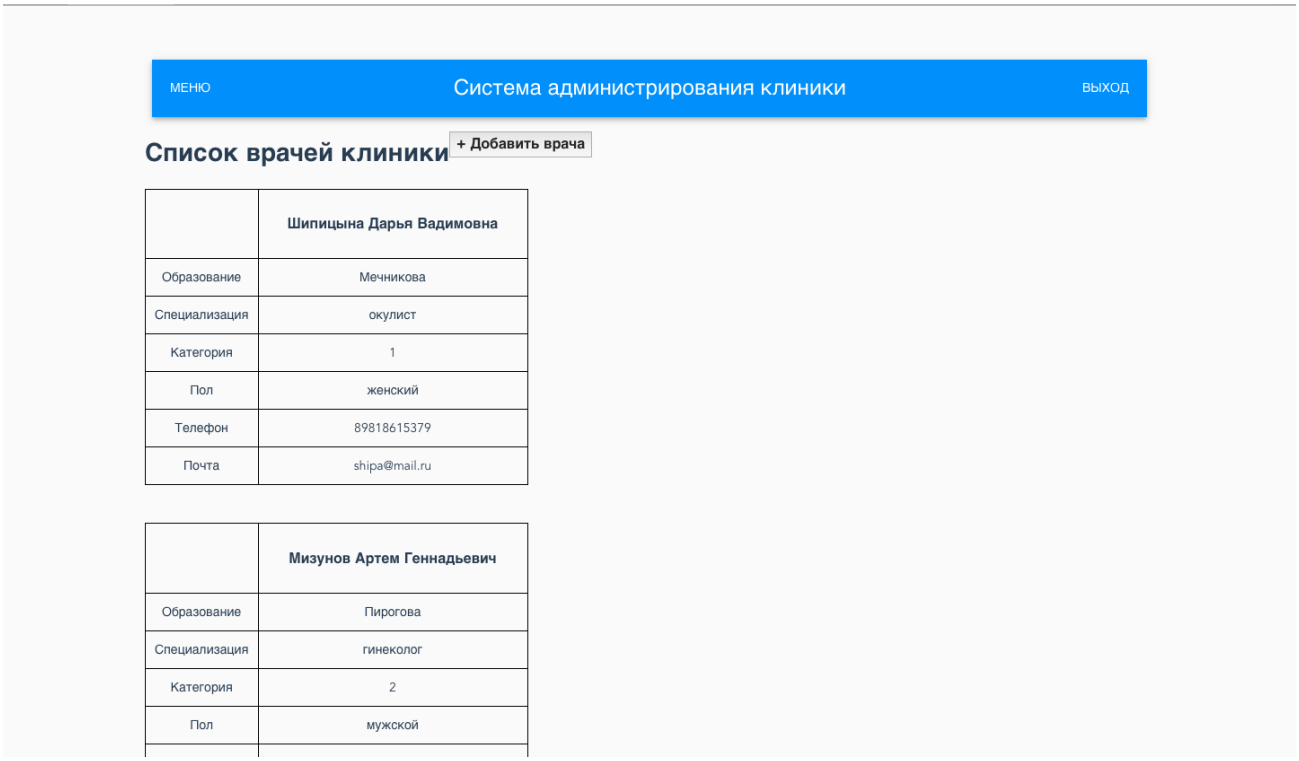

Рисунок 25 – список врачей клиники

При нажатии на имя врача выводится его актуальный график (рис. 26). Актуальный график – это только время будущих приемов, так как на бэкенде стоит фильтр по дате текущего дня.

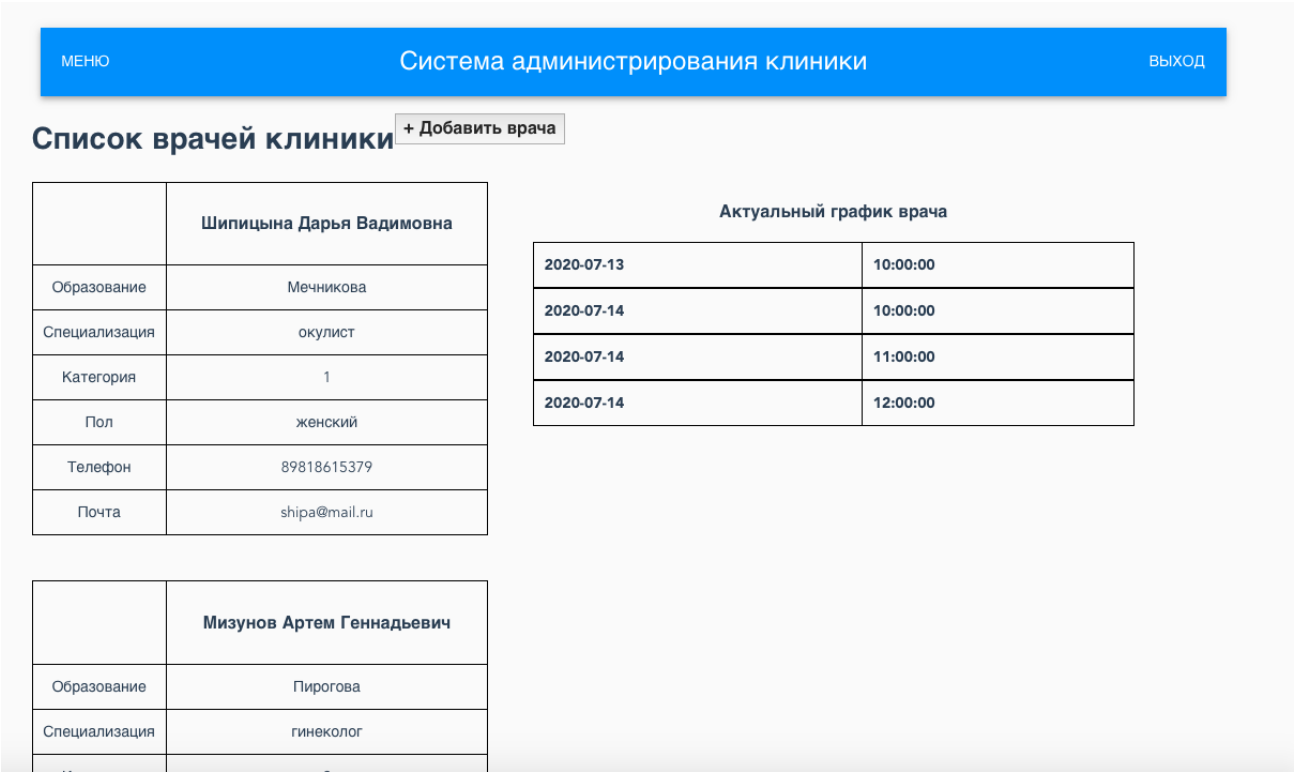

Рисунок 26 – актуальный график врача Мизунов Артем Геннадьевич

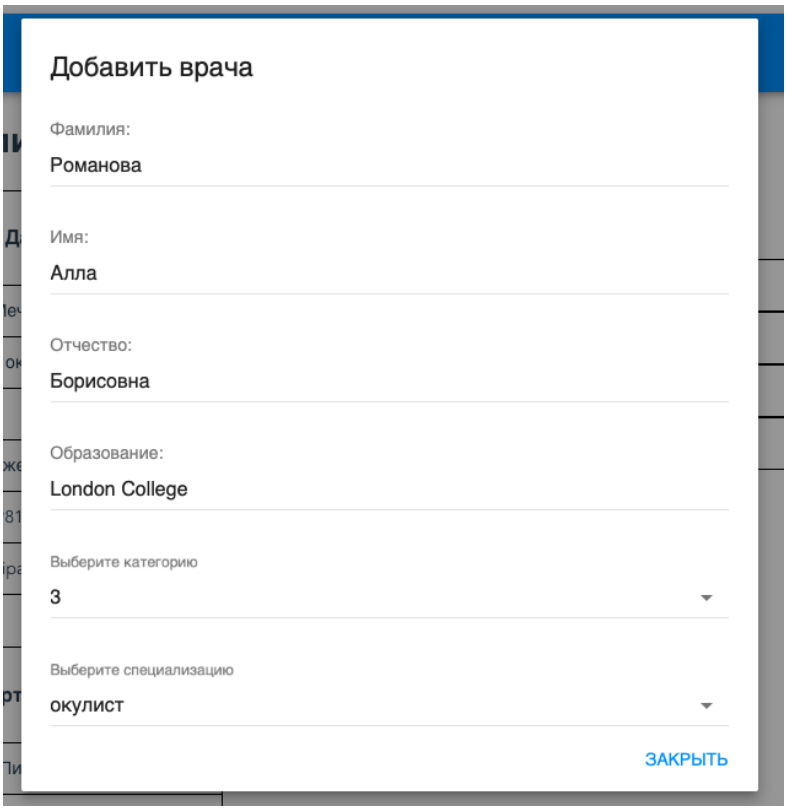

Рисунок 27 – форма добавления врача

На странице приемов реализован табличный список всех приемов (рис. 28), а также добавление приема (рис. 29). При добавлении приема, когда пользователь выбирает врача, то отображаются только незанятые окошки именно этого врача. При добавлении записи выбранное время становится недоступным для следующей записи на прием.

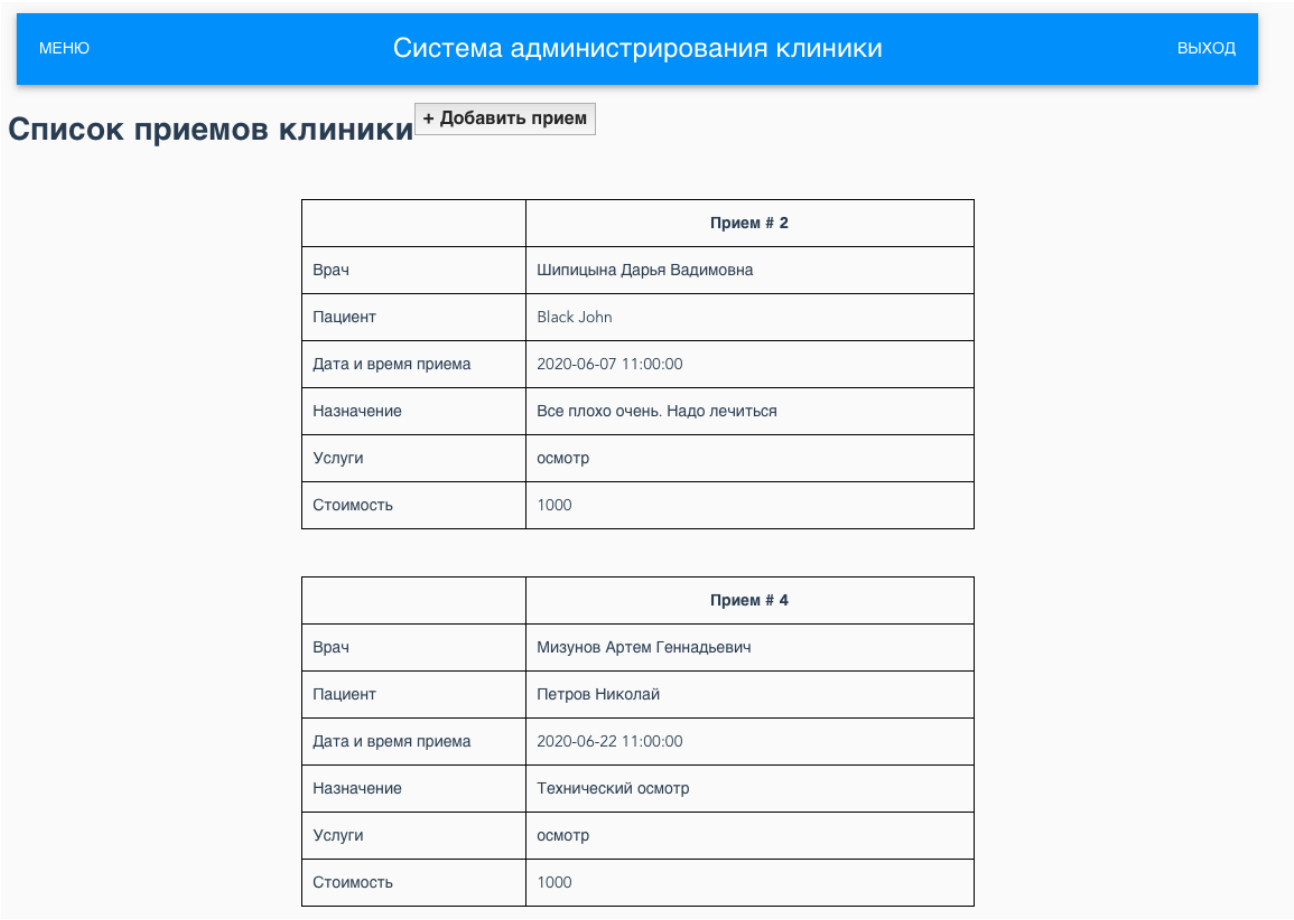

Рисунок 28 – список приемов клиники

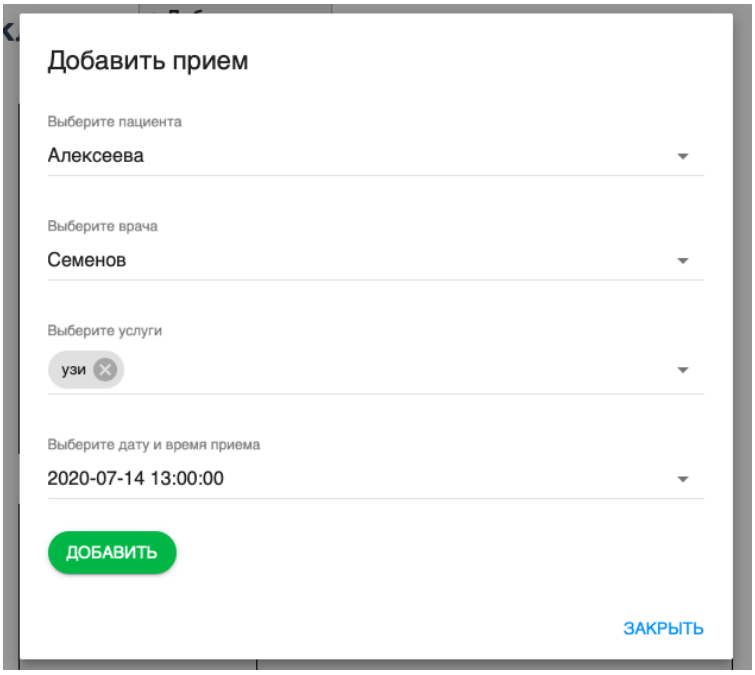

Рисунок 29 – добавление записи на прием

Ниже представлено отображение информации по запросам, которые могут возникнуть у администратора клиники.

Количество приемов у каждого врача

Фамилия врача	Количество
Мизунов	2
Семенов	1
Синий	1
Шипицына	1

Рисунок 30 – количество приемов у каждого врача

Запрос 2

Количество врачей каждой специализации

Специализация	Количество
гинеколог	2
окулист	1
терапевт	1

Рисунок 31 – количество врачей каждой специализации

Запрос 3

Сумма стоимостей всех приемов у каждого врача

Фамилия врача	Сумма
Мизунов	26000
Шипицына	1000
Семенов	700
Синий	700

Рисунок 32 – сумма стоимостей всех приемов у каждого врача

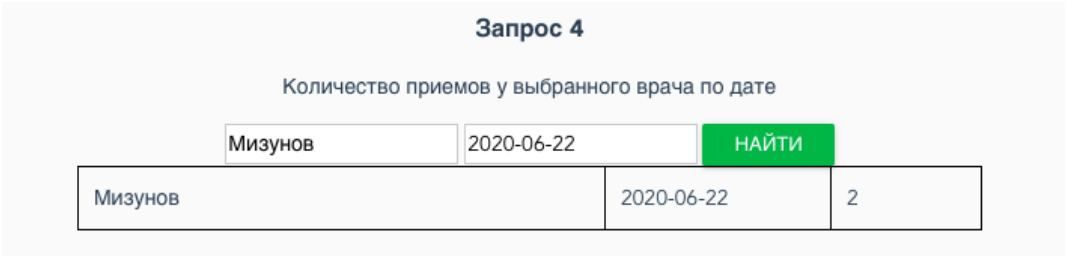

Рисунок 33 – количество приемов у выбранного врача в выбранную дату

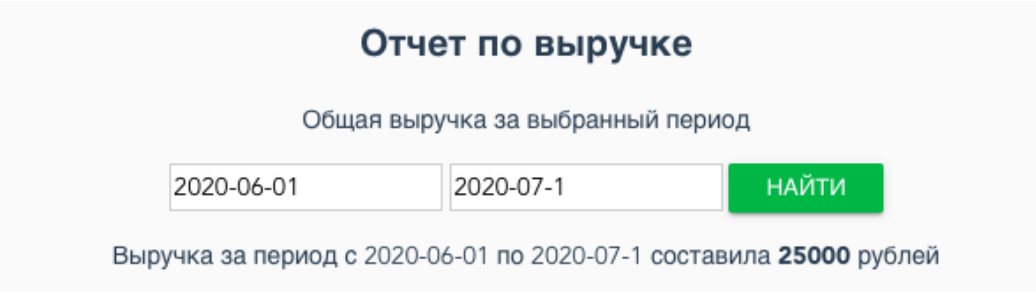

Рисунок 34 – отчет по выручке за выбранный период

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы были закреплены навыки анализа предметной области и создания модели данных, а также были получены навыки реализации Web-сервисов с выполнением всех необходимых этапов разработки, перечисленных среди задач данной курсовой работы.

В ходе выполнения работы были достигнуты все поставленные цели и задачи. Для этого были использованы навыки, приобретенные в ходе выполнения практических и лабораторных работ по предмету «Основы Web-программирования»: навыки использования Django Rest Framework, фреймворка Vue.JS и библиотеки MUSE-UI.

Результатом практической части курсовой работы является программная система для администрирования медицинской клиники.

СПИСОК ЛИТЕРАТУРЫ

1. Django.fun. Кратко о Django Rest Framework. [ЭИ] URL: https://django.fun/tutorials/kratko-o-django-rest-framework/ (дата обращения 10.06.2020)
2. Официальная документация Django Rest Framework. [ЭИ] URL: https://www.django-rest-framework.org/topics/documenting-your-api/ (дата обращения 15.06.2020)
3. Официальная документация Vue.JS. [ЭИ] URL: https://ru.vuejs.org/v2/guide/index.html (дата обращения 26.06.2020)
4. Курс лекций «Тестирование программного обеспечения». Архитектура «клиент-сервер». [ЭИ] URL: https://sergeygavaga.gitbooks.io/kurs-lektsii-testirovanie-programnogo-obespecheni/lektsiya-6-ch1-arhitektura-klient-server.html (дата обращения 01.07.2020)
5. Medium. Introduction to GenericAPIView in DRF. [ЭИ] URL: https://medium.com/@arrosid/introduction-to-genericapiview-in-django-rest-framework-eb557bf986b2 (дата обращения 26.06.2020)