

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 Интеллектуальные системы в гуманитарной сфере

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

О Т Ч Е Т

по курсовой работе

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

Обучающийся: Суздальцева Маргарита Вячеславовна

Руководитель: Говоров А.И.

Оценка ____

Дата ____

Санкт-Петербург
2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	3
1. АРХИТЕКТУРА, ПРЕДМЕТНАЯ ОБЛАСТЬ, ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ	4
1.1. Описание предметной области и функциональные требования	4
1.2. Архитектура web-сервиса	4
1.3. Архитектура базы данных	6
2. СЕРВЕРНАЯ ЧАСТЬ СЕРВИСА	8
3. КЛИЕНТСКАЯ ЧАСТЬ СЕРВИСА	17
3.1. Введение	17
3.2. Главная страница	18
3.3. Личный кабинет сотрудника / клиента	19
3.4. Детали заявки	20
3.5. Кабинет руководителя	22
3.6. Форма заявки на рекламу	26
3.7. Регистрация клиента, авторизация	27
4. ЗАКЛЮЧЕНИЕ.....	30
СПИСОК ЛИТЕРАТУРЫ.....	31
Приложение 1. Dockerfile & docker-compose.yml.....	32
Приложение 2. models.py	33
Приложение 3. serializers.py	35
Приложение 4. views.py	37

ВВЕДЕНИЕ

В задании 8 варианта было предложено создать программную систему, предназначенную для отдела маркетинга рекламного агентства “ЛУЧ”. Данный отдел занимается учетом работы с клиентами, то есть, **основная цель** - организация учета поступивших и выполненных заявок рекламодателей.

Сокращенный текст задания: «Одной из задач, решаемых отделом маркетинга рекламного агентства «Луч», является учет работы с клиентами. Для этого необходимо организовать оперативный учет поступивших и выполненных заявок клиентов (рекламодателей). Рекламное агентство заключает трудовые соглашения с заказчиками на исполнение определенного вида рекламных услуг. Для оформления заявки рекламодатель должен указать контактное лицо, телефон и электронный адрес для связи. Рекламодатель оформляет заявку на рекламу, пользуясь прайс-листом, в котором указаны цены по наименованию рекламных услуг, предоставляемых агентством «Луч». Здесь же оговариваются исполнители изготовления рекламы (сотрудники агентства), стоимость и объем (количество) работ. Для выполнения работ необходимо знать единицы измерения и материалы. Заказчик должен иметь контактные данные исполнителя. Согласно заявке, выписывается Платежное Поручение Заказчику, которое он обязан оплатить. После оплаты счета агентство обязуется предоставить рекламные продукты. Заказ считается выполненным, если оплачено Платежное поручение.»

В рамках системы клиент должен иметь доступ к прайс-листу и возможность оставить заявку на услугу (с кратким описанием). Заявку в дальнейшем рассматривают сотрудники и связываются с ним по оставленным данным. Аналогично, клиент имеет контактные данные исполнителя. Требуется выписывать платежное поручение клиенту, а агентство обязано предоставить рекламные продукты по итогам работы.

Были поставлены такие **задачи**:

1. Спроектировать БД, соответствующую предметной области;
2. Реализовать API;
3. Реализовать интерфейсы с использованием API;
4. Выполнить контейнеризацию и оркестрацию приложения средствами docker и docker-compose.

Web-сервис был реализован средствами Python 3.6, Django (3.0.6), Django REST framework (3.11.0), Vue.js, Vuetify (аналог Muse-UI). Использована БД PostgreSQL.

1. АРХИТЕКТУРА, ПРЕДМЕТНАЯ ОБЛАСТЬ, ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

1.1. Описание предметной области и функциональные требования

Процесс работы организован примерно так: клиент просматривает прайс-лист услуг, регистрируется с контактными данными и информацией о компании, оставляет заявку на рекламу, выбирая услугу из списка, кратко описывая материалы и желаемый результат. С этого момента исполнитель может связаться с клиентом и наоборот. Предполагается, что и у сотрудников, и у клиентов есть доступ к своим заявкам, а руководителю доступны все. К каждой заявке прикрепляется платёжное поручение в момент создания, а по итогам выполнения – добавляются рекламные продукты.

В результате анализа текста задания были сформулированы следующие **функциональные требования**:

1. Отображение прайс-листа услуг и статистики о работе агентства
2. Регистрация клиента - оставить и просмотреть заявки можно только после авторизации (сотрудники также авторизуются, но добавляются администратором)
3. Добавление новой заявки на рекламу клиентом (может оставить более одной)
4. Отображение заявок клиента / исполнителя в личном кабинете, сортировка по дате создания
5. Кабинет руководителя с ограниченным доступом
6. Просмотр всех заявок руководителем с возможностью фильтровать их по всем полям (и информация о количестве найденных по запросу заявок)
7. Возможность для руководителя отметить любую заявку как выполненную
8. Отображение деталей выбранной заявки (“подробнее”), полной информации о ней, о соответствующем платёжном поручении и прикрепленных рекламных продуктах

1.2. Архитектура web-сервиса

Для реализации сервиса использовалось следующее ПО:

БД - PostgreSQL, база данных подключена в settings.py с помощью psycopg2-binary (рис. 1);

```

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'marketingdb',
        'USER': 'postgres',
        'PASSWORD': 'postgres',
        'HOST': 'marketing_db',
        'PORT': '5432'
    }
}

```

Рисунок 1 – Подключение БД

backend – Django (3.0.6) и конкретно Django REST framework (3.11.0) как основа сервиса, отвечает за все запросы, требуемые исходя из функциональных требований (отображение, фильтрация, добавление, изменение данных и т. д.).

Модули, использованные в проекте представлены на рис. 2:

```

asgiref==3.2.7
Django==3.0.6
django-cors-headers==3.3.0
django-templated-mail==1.1.1
djangorestframework==3.11.0
djoser==2.0.3
psycpg2-binary==2.8.5
pytz==2020.1
sqlparse==0.3.1

```

Рисунок 2 - requirements.txt

Запросы к серверу разрешены с помощью cors headers (рис. 3):

```

113  CORS_ORIGIN_WHITELIST = [
114      "http://localhost:8080",
115      "http://127.0.0.1:8080",
116      "http://localhost:8085",
117      "http://127.0.0.1:8085"
118  ]
119
120  CORS_ALLOW_METHODS = [
121      'DELETE',
122      'GET',
123      'OPTIONS',
124      'PATCH',
125      'POST',
126      'PUT',
127  ]

```

Рисунок 3 – разрешение запросов

Для выполнения тестовых запросов в процессе создания API дополнительно использовалось приложение Postman.

frontend - Vue.js в комбинации с плагином Vuetify (выбран вместо предложенного в задании Muse-UI) для реализации интерфейсов и базового дизайна.

docker - наконец, добавлен Dockerfile для бэкенда и для фронтенда, а также файл docker-compose.yml. Они служат для удобства и скорости сборки и запуска приложения. Docker-файлы приведены в Приложении 1.

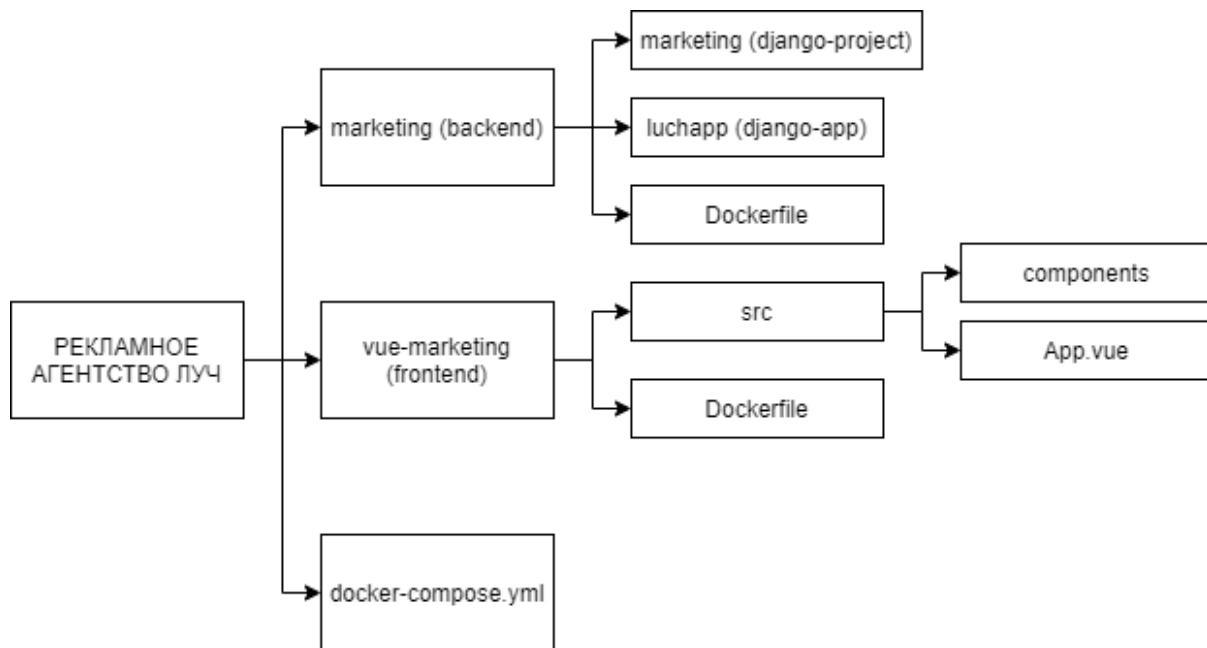


Рисунок 4 - Структура web-сервиса

1.3. Архитектура базы данных

При проектировании базы данных marketingdb были выбраны следующие сущности:

- Компания (Company) - компания клиента;
- Услуга (Service);
- Сотрудник (Employee) и Клиент (Client) - оба связаны с таблицей User по Foreign Key;
- Заявка (Request) - ассоциативная сущность, через которую связаны клиент, сотрудник, услуга;
- Платёж (Payment) - платёж (или платёжное поручение), относящийся к заявке;
- Продукт (Product) - результаты работы по заявке.

Таким образом, БД включает таблицы:

- Клиент (id, Компания (FK), ФИО, Телефон, Email)
- Услуга (id, Вид, Наименование, Цена)
- Сотрудник (id, ФИО, Должность, Телефон, Email)
- Заявка (id, Услуга (FK), Клиент (FK), Сотрудник (FK), Краткое описание, Материалы, Дата, Статус)

- Платёж (id, Заявка (FK), Сумма, Дата запроса, Дата оплаты, Статус)
- Компания (id, Название, Тип)
- Продукт (id, Заявка (FK), Название, Описание, Изображение)

Схема представлена на рис. 5:

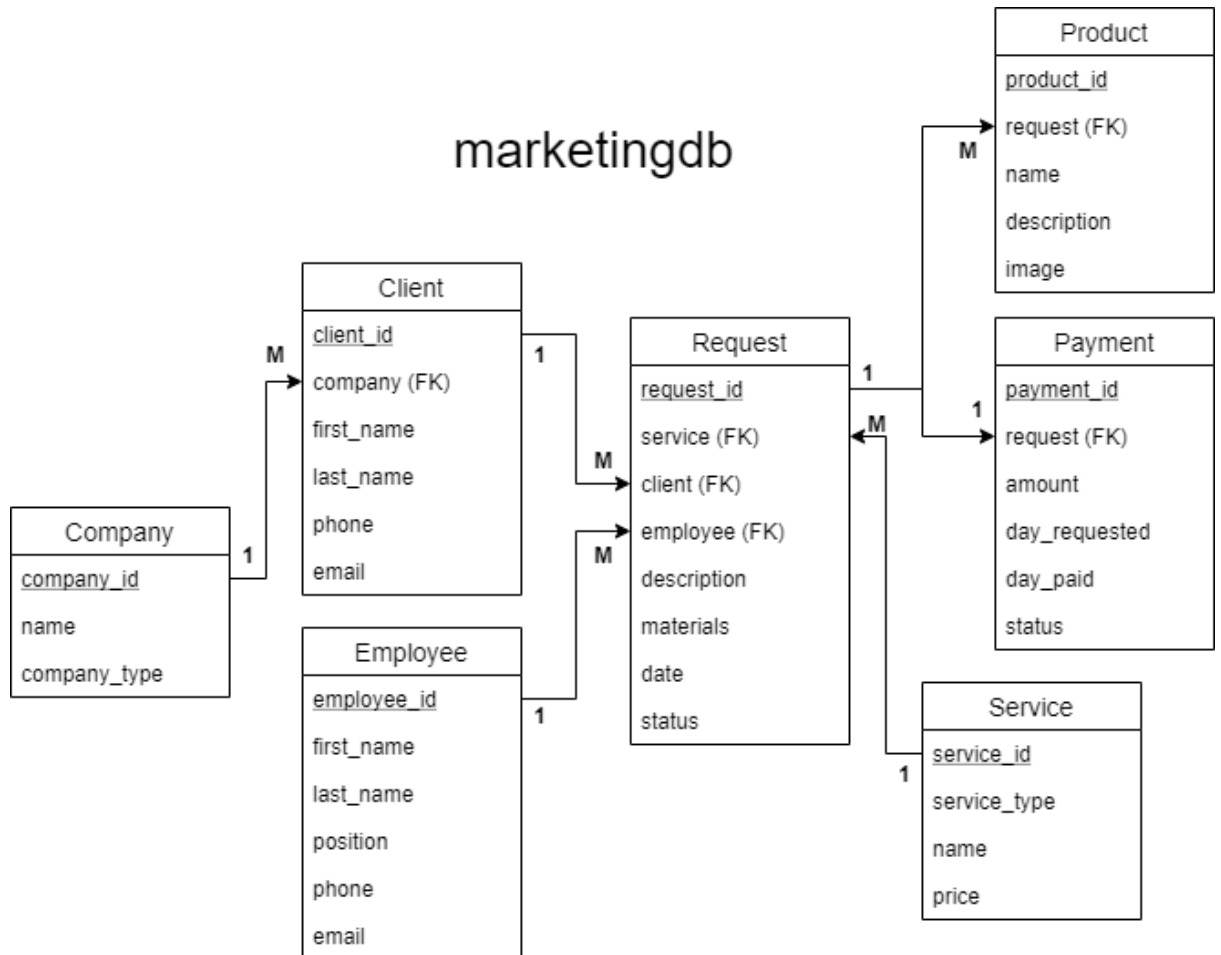


Рисунок 5 - Схема базы данных web-сервиса

2. СЕРВЕРНАЯ ЧАСТЬ СЕРВИСА

Бэкенд приложения был реализован с помощью Django REST framework (3.0.6).

Модели таблиц БД из прошлой главы описаны в файле models.py (см. Приложение 2).

Далее они были зарегистрированы в админ-панели (рис. 6, 7, 8).

```
1 from django.contrib import admin
2
3 from .models import *
4
5 admin.site.register(Company)
6
7 admin.site.register(Service)
8
9 admin.site.register(Employee)
10
11 admin.site.register(Client)
12
13 admin.site.register(Request)
14
15 admin.site.register(Payment)
16
17 admin.site.register(Product)
```

Рисунок 6 – admin.py

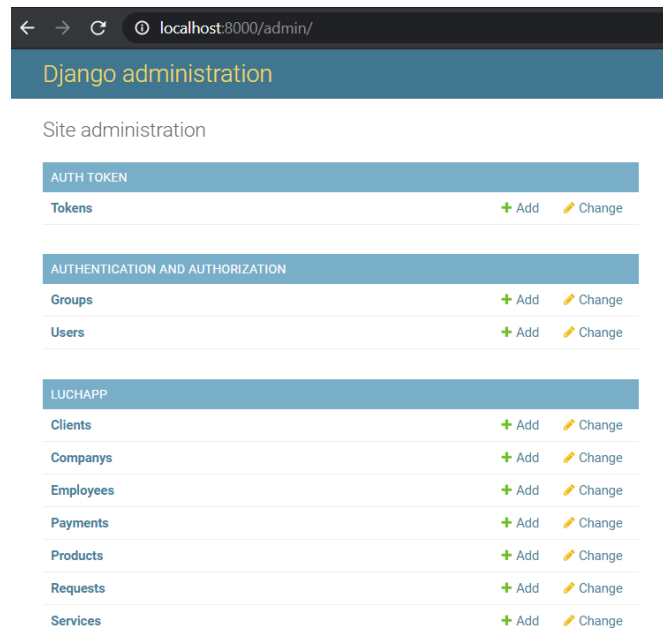


Рисунок 7 – админ-панель Django

The screenshot shows the Django administration interface for a 'request' model. The page title is 'Django administration' and the breadcrumb trail is 'Home > Luchapp > Requests > Request object (1)'. The main heading is 'Change request'. The form contains the following fields:

- Service:** A dropdown menu showing 'Service object (3)' with a plus icon.
- Client:** A dropdown menu showing 'Client object (1)' with a plus icon.
- Employee:** A dropdown menu showing 'Employee object (1)' with a plus icon.
- Description:** A text input field containing 'собираюсь продавать hand-made шопперы'.
- Materials:** A text input field containing 'фото сумок, макет'.
- Status:** A dropdown menu showing 'completed'.

At the bottom of the form, there are four buttons: 'Delete' (red), 'Save and add another' (blue), 'Save and continue editing' (blue), and 'SAVE' (blue).

Рисунок 8 – Пример заявки

В `serializers.py` (см. Приложение 3) прописаны сериализаторы для каждой модели с помощью `serializers.ModelSerializer` из `rest_framework`, представляющие их в формате json. В некоторых случаях понадобилось несколько для одной (например, различные поля выбраны или `read_only=True`). Также у полей с множественным выбором указан `source` для корректного отображения.

В файле `views.py` с помощью `generics views` из `rest_framework` описаны представления, основанные на классах. Были использованы `generics.CreateAPIView`, `generics.RetrieveUpdateDestroyAPIView`, `generics.ListAPIView`, `generics.RetrieveAPIView`, в зависимости от того, какие функции из CRUD (create, read, update, delete) были необходимы в каждой ситуации. Скриншоты `views.py` – в Приложении 4.

Во многих view требовалась фильтрация объектов запроса, она реализуется в функции `get_queryset()`. Функция сначала получает все объекты модели, затем параметры запроса, после этого - фильтрует (и если нужно - сортирует с помощью `order_by`) записи, возвращая `queryset`:

```

37 class AdminFilterView(generics.ListAPIView):
38     serializer_class = RequestSerializer
39
40     def get_queryset(self):
41         queryset = Request.objects.all()
42         params = self.request.query_params
43
44         service = params.get('service', None)
45         client = params.get('client', None)
46         employee = params.get('employee', None)
47         status = params.get('status', None)
48         after = params.get('after', None)
49         before = params.get('before', None)
50
51         if service:
52             queryset = queryset.filter(service__id=service)
53
54         if client:
55             queryset = queryset.filter(client__id=client)
56
57         if employee:
58             queryset = queryset.filter(employee__id=employee)
59
60         if status:
61             queryset = queryset.filter(status=status)
62
63         if after:
64             queryset = queryset.filter(date__date__gte=after)
65
66         if before:
67             queryset = queryset.filter(date__date__lte=before)
68
69         queryset = queryset.order_by('-date')
70     return queryset
71

```

Рисунок 9 – пример представления с фильтрацией

В urls.py (рис. 10) содержатся адреса, соответствующие каждому из представлений:

```

1  from django.urls import path, include
2  from .views import *
3  from rest_framework.auth_token.views import obtain_auth_token
4
5  app_name="luchapp"
6
7  urlpatterns = [
8      path('auth/', include('djoser.urls')),
9      path('auth/token', obtain_auth_token, name='token'),
10     path('request/new/', CreateRequestView.as_view()),
11     path('request/<int:pk>', GetRequestView.as_view()),
12     path('request/all', GetRequestsView.as_view()),
13     path('client/new', CreateClientView.as_view()),
14     path('client/all', GetClientsView.as_view()),
15     path('company/new', CreateCompanyView.as_view()),
16     path('payment/new', CreatePaymentView.as_view()),
17     path('payment/<int:pk>', GetPaymentView.as_view()),
18     path('payment/all', GetPaymentsView.as_view()),
19     path('service/<int:pk>', GetServiceView.as_view()),
20     path('service/all', GetServicesView.as_view()),
21     path('gethead/', GetHeadView.as_view()),
22     path('getclient/', GetClientView.as_view()),
23     path('request/adminfilter', AdminFilterView.as_view()),
24     path('employee/all', GetEmployeesView.as_view()),
25     path('product/all', GetProductsView.as_view()),
26 ]

```

Рисунок 10 – luchapp/urls.py

```

16 from django.contrib import admin
17 from django.urls import path, include
18
19 urlpatterns = [
20     path('admin/', admin.site.urls),
21     path('api/', include('luchapp.urls')),
22 ]

```

Рисунок 11 – marketing/urls.py

Теперь, если перейти по адресу localhost:8000/api/ отображается:

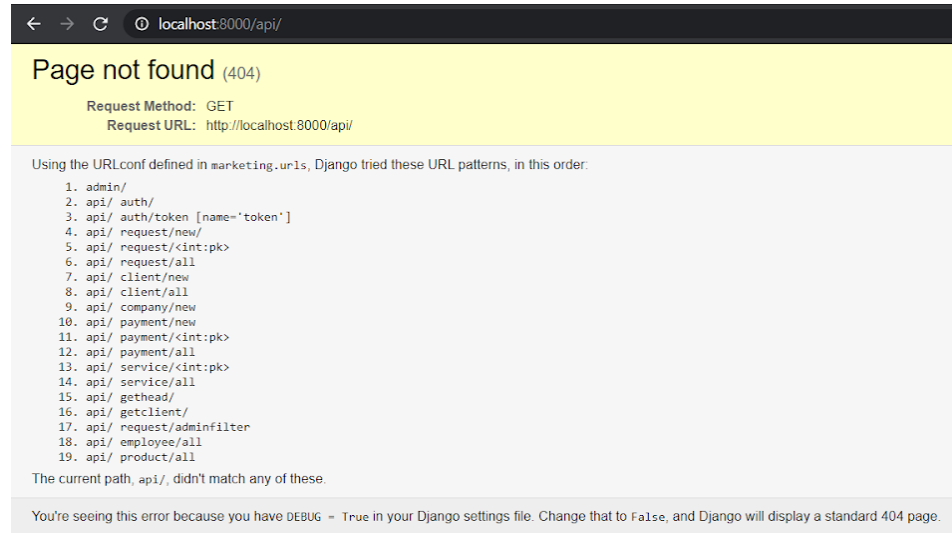


Рисунок 12 - Адреса, относящиеся к API

Здесь api/auth/ и api/auth/token - авторизация по токenu djoser.

Получившиеся эндпоинты сведены в таблицу 1.

Таблица 1 – сводная таблица информации о созданном API

№	urls	запросы	назначение	соответствие с views.py
4	api/request/new/	POST, OPTIONS	создает новую заявку (запись в таблице Request, параметры -- её поля)	class CreateRequestView(generics.CreateAPIView)
5	api/request/<int:pk>	GET, PUT, PATCH, DELETE, HEAD, OPTIONS	работа (получение, обновление, удаление...) с заявкой с определенным id (primary key)	class GetRequestView(generics.RetrieveUpdateDestroyAPIView)
6	api/request/all	GET, HEAD, OPTIONS	вывод всех заявок, возможна фильтрация по пользователю (клиенту/сотруднику), параметр ?user=user__username	class GetRequestsView(generics.ListAPIView)
7	api/client/new	POST, OPTIONS	создает нового клиента (запись в таблице Client, параметры -- её поля)	class CreateClientView(generics.CreateAPIView)

8	api/client/all	GET, HEAD, OPTIONS	вывод всех клиентов	class GetClientsView(generics.ListAPIView)
9	api/company/new	POST, OPTIONS	создает новую компанию (запись в таблице Company, параметры -- её поля)	class CreateCompanyView(generics.CreateAPIView)
10	api/payment/<int:pk>	GET, PUT, PATCH, DELETE, HEAD, OPTIONS	работа с платежным поручением с определенным id	class GetPaymentView(generics.RetrieveUpdateDestroyAPIView)
11	api/payment/new	POST, OPTIONS	создает новое платёжное поручение (запись в таблице Payment, параметры -- её поля)	class CreatePaymentView(generics.CreateAPIView)
12	api/payment/all	GET, HEAD, OPTIONS	вывод платежки, относящейся к заявке, параметр ?req=request__id	class GetPaymentsView(generics.ListAPIView)
13	api/service/<int:pk>	GET, HEAD, OPTIONS	вывод услуги с определенным id	class GetServiceView(generics.RetrieveAPIView)
14	api/service/all	GET, HEAD, OPTIONS	вывод всех услуг	class GetServicesView(generics.ListAPIView)
15	api/gethead/	GET, HEAD, OPTIONS	определяет, является ли пользователь сотрудником-руководителем (отдаёт сотрудника либо ничего), параметр ?user=user_username	class GetHeadView(generics.ListAPIView)
16	api/getclient/	GET, HEAD, OPTIONS	определяет, является ли пользователь клиента (отдаёт клиента либо ничего), если задан параметр ?user=user_username	class GetClientsView(generics.ListAPIView)
17	api/request/adminfilter	GET, HEAD, OPTIONS	вывод заявок с использованием следующих фильтров: service, client, status, after & before (промежуток дат)	class AdminFilterView(generics.ListAPIView)
18	api/employee/all	GET, HEAD, OPTIONS	вывод всех сотрудников	class GetEmployeesView(generics.ListAPIView)
19	api/product/all	GET, HEAD, OPTIONS	вывод продуктов, относящихся к заявке, параметр ?req=request__id	class GetProductsView(generics.ListAPIView)

Так как многие эндпоинты аналогичны, рассмотрим только некоторые из них подробно.

api/request/new - служит для создания новой заявки, принимает несколько параметров (ниже на скриншоте), а дата создания добавляется автоматически:

Рисунок 13 – api/request/new

api/request/all - служит в основном для отображения заявок конкретного клиента / исполнителя, хотя без заданных параметров выводит все;

username=martgorit (исполнитель):

Рисунок 14 – api/request/all?user=martgorit

api/request/<int:pk> - служит для работы с заявкой, имеющей заданный id (primary key);

Заявка с id=3:

```
GET /api/request/3

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

{
  "id": 3,
  "service": {
    "id": 6,
    "service_type": "другое",
    "name": "рекламный ролик под ключ",
    "price": 100000
  },
  "client": {
    "id": 3,
    "user": {
      "id": 6,
      "username": "oleoleg",
      "email": "oleoleg@yandex.ru"
    },
    "company": {
      "id": 3,
      "company_type": "ООО",
      "name": "Oleg Inc."
    },
    "phone": "666666",
    "first_name": "Oleg",
    "last_name": "Olegov"
  },
  "employee": {
    "id": 3,
    "user": {
      "id": 3,
      "username": "narugarita",
      "email": "rzdaltseva@yandex.ru"
    },
    "position": "executor",
    "phone": "333333",
    "first_name": "Pera",
    "last_name": "Cy"
  },
  "description": "реклама корма для кошек",
  "materials": "корм, привады дрессированных кошек, остальное -- обдум",
  "date": "2020-06-26T16:35:41.330912Z",
  "status": "1"
}
```

Рисунок 15 – api/request/3

api/request/adminfilter - служит для фильтрации заявок по статусу выполнения, периоду создания, клиенту, исполнителю, услуге, т.е. по всем полям. Выполненные заявки (status=1):

```
GET /api/request/adminfilter/status=1

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

{
  "id": 6,
  "service": {
    "id": 6,
    "service_type": "другое",
    "name": "рекламный ролик под ключ",
    "price": 100000
  },
  "client": {
    "id": 6,
    "user": {
      "id": 9,
      "username": "sasha",
      "email": "sasha@gmail.com"
    },
    "company": {
      "id": 6,
      "company_type": "ООО",
      "name": "Счастливые Радости Реклама"
    },
    "phone": "000000",
    "first_name": "Cass",
    "last_name": "Cassix"
  },
  "employee": {
    "id": 1,
    "user": {
      "id": 1,
      "username": "rita",
      "email": "rzdaltseva@gmail.com"
    },
    "position": "head",
    "phone": "111111",
    "first_name": "Маггарита",
    "last_name": "Суздальцева"
  },
  "description": "самый крутой рекламный ролик чипсов",
  "materials": "чипсы, а что еще - я не знаю",
  "date": "2020-06-29T17:50:41.912136Z",
  "status": "1"
}
```

Рисунок 16 – api/request/adminfilter?status=1 (часть 1)

localhost:8000/api/request/adminfilter?status=1	localhost:8000/api/request/adminfilter?status=1
Django REST framework	Django REST framework
<pre> "id": 3, "username": "marugarita", "email": "rzdaltseva@yandex.ru" }, "position": "executor", "phone": "333333", "first_name": "Рита", "last_name": "Су" }, "description": "реклама корма для кошек", "materials": "корм, приведём дрессированных кошек, остальное -- обсудим", "date": "2020-06-26T16:36:41.336912Z", "status": "1" }, { "id": 2, "service": { "id": 2, "service_type": "реклама на транспорте", "name": "брендированный автобус", "price": 30000 }, "client": { "id": 2, "user": { "id": 5, "username": "polinka", "email": "polinka@mail.ru" }, "company": { "id": 2, "company_type": "ООО", "name": "Весёлая Логистика" }, "phone": "555555", "first_name": "Полина", "last_name": "Анкна" }, "employee": { "id": 2, "user": { </pre>	<pre> "id": 2, "username": "martgorit", "email": "ritusyonok@gmail.com" }, "position": "executor", "phone": "222222", "first_name": "Март", "last_name": "Горитовна" }, "description": "реклама услуг компании на автобусе", "materials": "нужна консультация", "date": "2020-06-26T16:29:06.947084Z", "status": "1" }, { "id": 1, "service": { "id": 3, "service_type": "реклама в интернете", "name": "настройка таргета в instagram", "price": 15000 }, "client": { "id": 1, "user": { "id": 4, "username": "maryshan", "email": "maryshan@gmail.com" }, "company": { "id": 1, "company_type": "ИП", "name": "Мари Шан" }, "phone": "444444", "first_name": "Мари", "last_name": "Шан" }, "employee": { "id": 1, "user": { </pre>

Рисунок 17 – api/request/adminfilter?status=1 (часть 2)

<pre> "user": { "id": 1, "username": "rita", "email": "rzdaltseva@gmail.com" }, "position": "head", "phone": "111111", "first_name": "Маргарита", "last_name": "Суздальцева" }, "description": "собираюсь продавать hand-made шопперы", "materials": "фото сумок, макет", "date": "2020-06-26T16:02:25.837351Z", "status": "1" }] </pre>

Рисунок 18 – api/request/adminfilter?status=1 (часть 3)

api/getclient/ - служит для определения клиента по username пользователя (или отдаёт пустой массив, если пользователь - не клиент);

username=sasha - клиент, username=rita - не клиент:

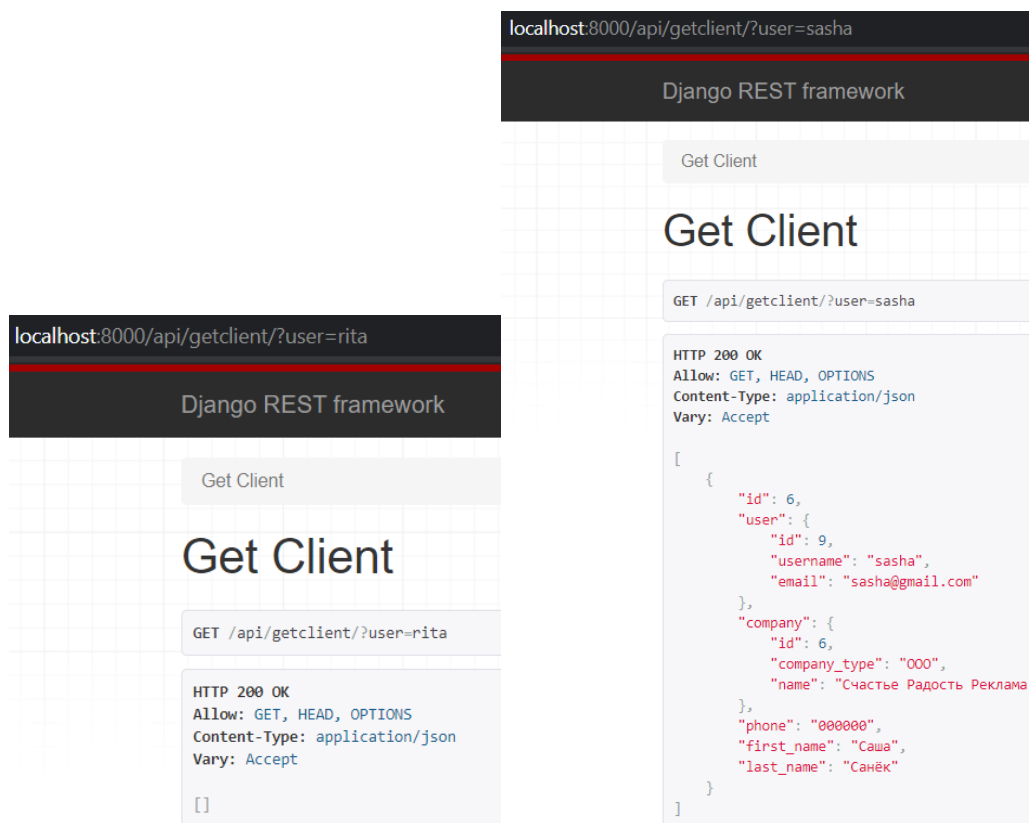


Рисунок 19 – api/getclient/?user=rita, api/getclient/?user=sasha

api/gethead/ - служит для определения того, руководитель ли авторизован, по username пользователя;

username=rita - руководитель, username=martgorit - простой сотрудник:

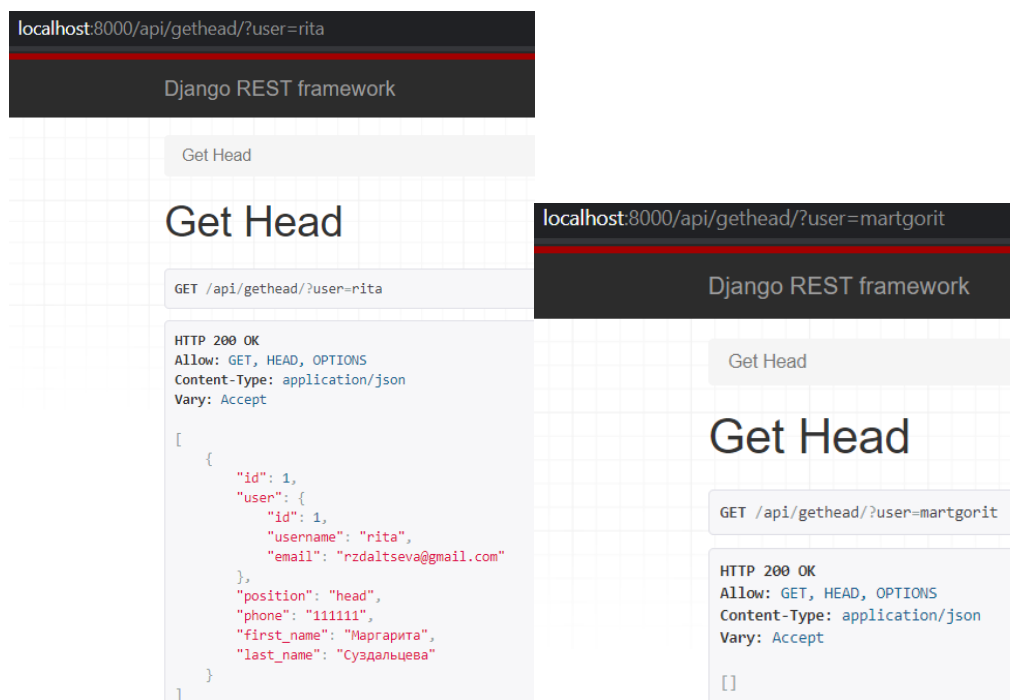


Рисунок 20 – api/gethead/?user=rita, api/gethead/?user=martgorit

3. КЛИЕНТСКАЯ ЧАСТЬ СЕРВИСА

3.1. Введение

Для создания интерфейсов приложения использовалось описанное в предыдущей главе API, Vue.js, Vuetify (простейший дизайн), axios (запросы).

Список интерфейсов:

1. Главная страница
2. Личный кабинет сотрудника / клиента
3. Детали заявки
4. Кабинет руководителя
5. Форма заявки на рекламу
6. Форма регистрации клиента, авторизация

Боковое меню (navigation drawer) из 3 вкладок («главная», «личный кабинет», «для админа») описано в App.vue. Адресация представлена в файле router/index.js (рис. 21):

```
1 import Vue from 'vue'
2 import Router from 'vue-router'
3 import Index from '@/components/Index'
4 import Cabinet from '@/components/Cabinet'
5 import AdminCab from '@/components/AdminCab'
6 import Details from '@/components/Details'
7 import ReqForm from '@/components/ReqForm'
8 import RegClient from '@/components/RegClient'
9 import Auth from '@/components/Auth'
10
11 Vue.use(Router)
12
13 export default new Router({
14   routes: [
15     {
16       path: '/',
17       name: 'Index',
18       component: Index
19     },
20     {
21       path: '/Cabinet',
22       name: 'Cabinet',
23       component: Cabinet
24     },
25     {
26       path: '/auth',
27       name: 'Auth',
28       component: Auth
29     },
30     {
31       path: '/adminCab',
32       name: 'AdminCab',
33       component: AdminCab
34     },
35     {
36       path: '/details/:id',
37       name: 'Details',
38       component: Details,
39       props: true
40     },
41     {
42       path: '/newReq',
43       name: 'ReqForm',
44       component: ReqForm
45     },
46     {
47       path: '/newClient',
48       name: 'RegClient',
49       component: RegClient,
50       props: true
51     }
52   ]
53 })
```

Рисунок 21 – router/index.js

3.2. Главная страница

Главная страница приложения выглядит следующим образом:

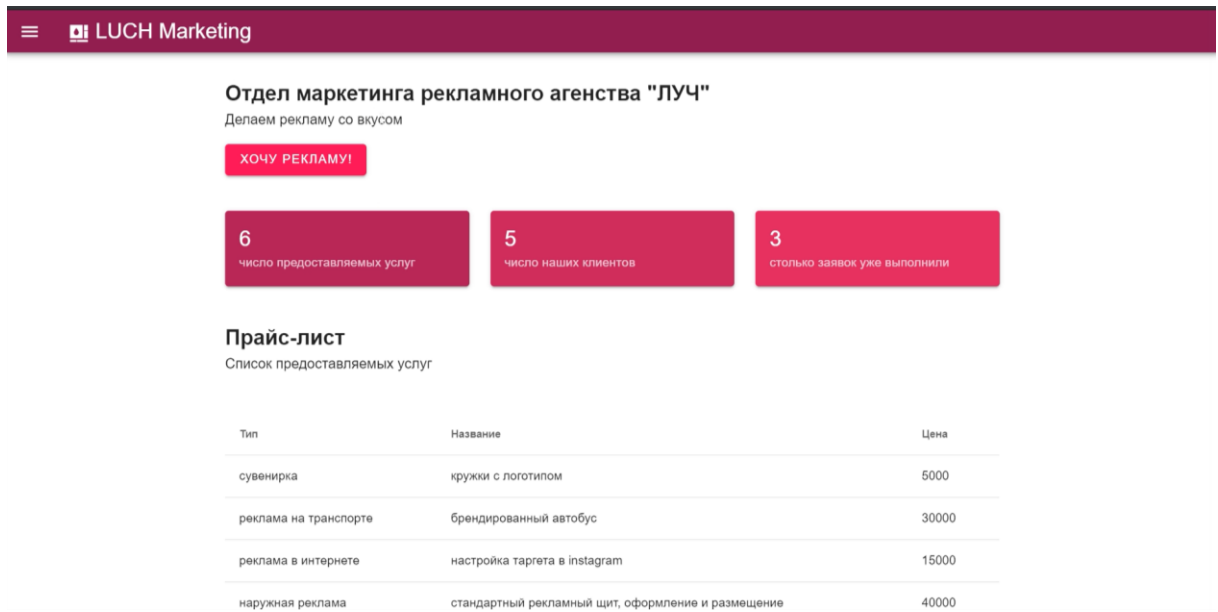


Рисунок 22 – Интерфейс «Главная страница»

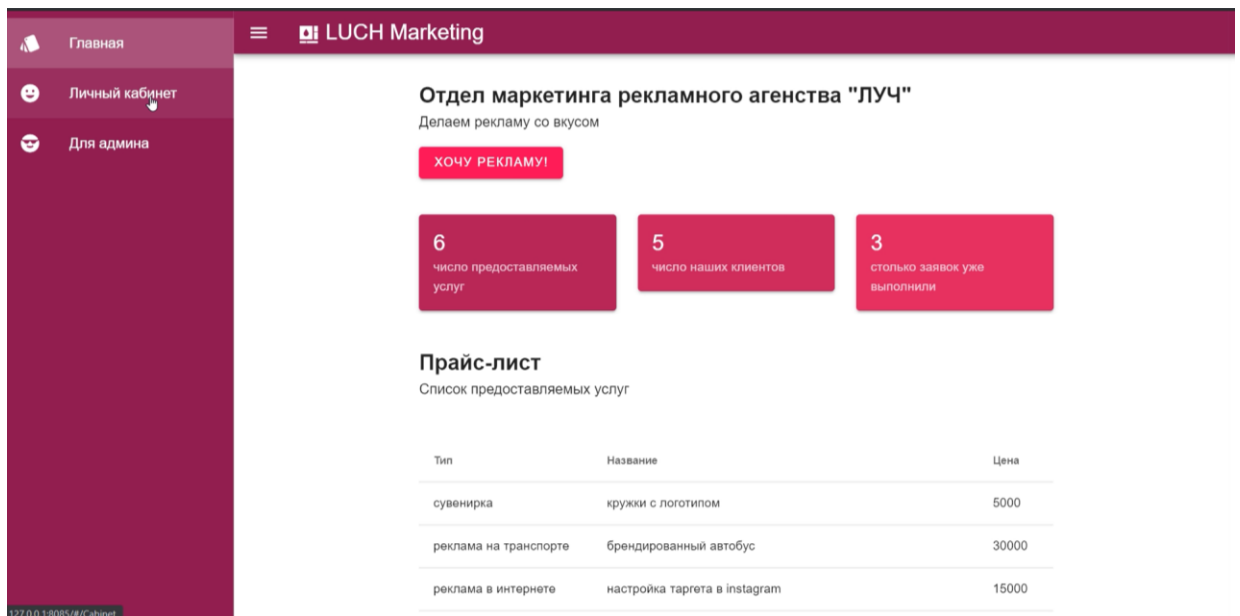


Рисунок 23 – Интерфейс «Главная страница» с открытым боковым меню

Название компоненты: Index.vue

Элементы: кнопка-ссылка на форму для заявок, карточки со статистикой, список «прайс-лист услуг»

Доступ: для всех

Выполненные запросы:

Таблица 2 – Запросы в Index.vue

POSTMAN	axios	Назначение
GET http://127.0.0.1:8000/ api/service/all	.get('http://\${window.location.hostname}: 8000/api/service/all')	Получение списка всех услуг из БД, чтобы составить прайс-лист и посчитать количество
GET http://127.0.0.1:8000/ api/client/all	.get('http://\${window.location.hostname}: 8000/api/client/all')	Получение списка всех клиентов, чтобы затем их посчитать
GET http://127.0.0.1:8000/ api/request/all	.get('http://\${window.location.hostname}: 8000/api/request/all')	Получение списка всех заявок, чтобы посчитать количество выполненных

3.3. Личный кабинет сотрудника / клиента

Личный кабинет универсален для сотрудника и клиента: клиент видит в нём оставленные заявки, а сотрудник – те, на который назначен исполнителем.

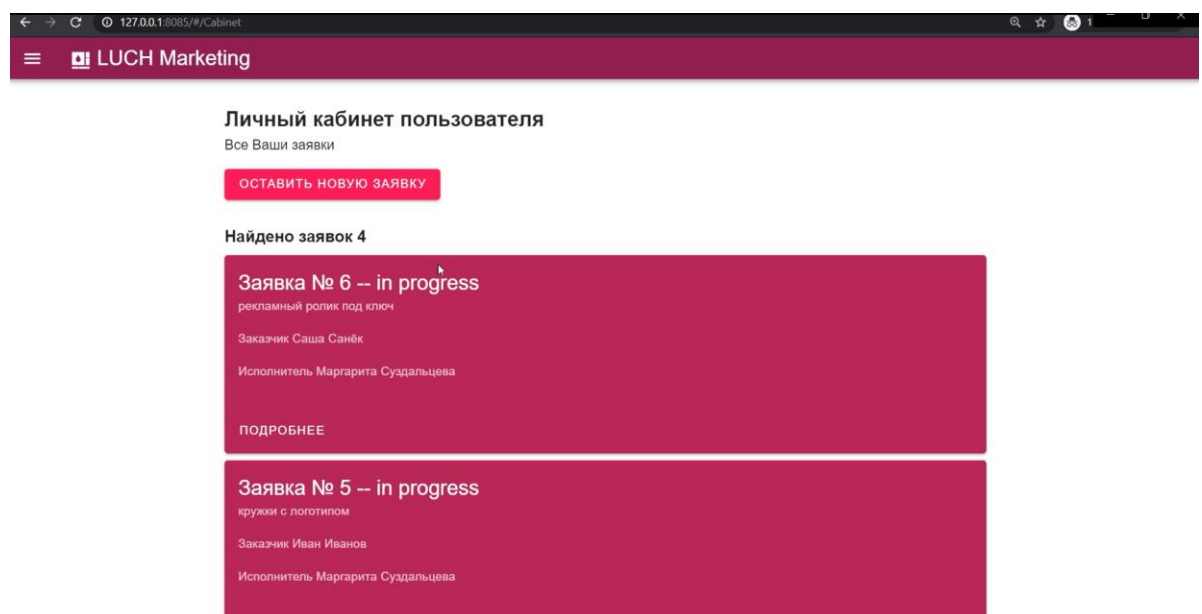


Рисунок 24 – Интерфейс «Личный кабинет»

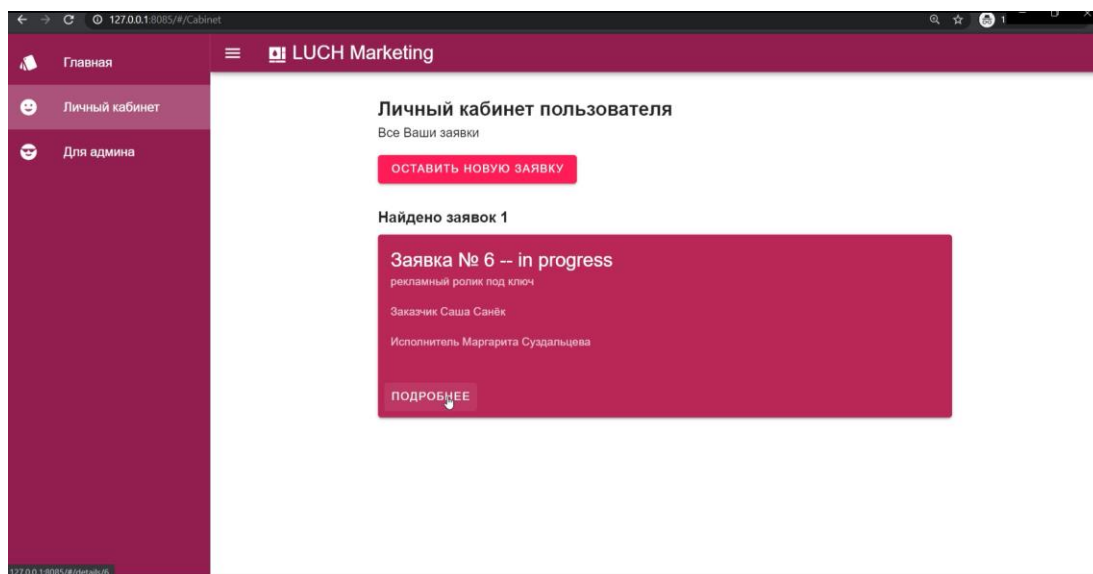


Рисунок 25 – Интерфейс «Личный кабинет» (2)

Название компоненты: Cabinet.vue

Элементы: кнопка-ссылка на форму для заявок, число найденных заявок, карточки заявок с краткой информацией о заявке (статус выполнения, услуга, исполнитель, заказчик) и кнопкой «подробнее» ведущей на страницу «детали» и отдающей id в компоненту

Доступ: для авторизованных пользователей, переадресация на авторизацию

Выполненные запросы:

Таблица 3 – Запросы в Cabinet.vue

POSTMAN	axios	Назначение
GET http://127.0.0.1:8000/api/service/all?user=<username>	.get('http://\${window.location.hostname}:8000/api/request/all?user=\${sessionStorage.getItem('user')}')	Получение заявок авторизованного пользователя

3.4. Детали заявки

Данный интерфейс содержит полную информацию о выбранной заявке, прикрепленные к ней платёжное поручение (с кнопкой оплаты, которая доступна, если заявка не оплачена – `if="pay_status === 'не оплачено'"`) и продукты.

Название компоненты: Details.vue

Элементы: таблица со всей информацией о заявке (номер, услуга, клиент, сотрудник, краткое описание, материалы, дата и время создания, статус), карточка платежа (номер, сумма, когда запрошен, статус оплаты, если оплачен – дата и время), карточки рекламных продуктов (можно просмотреть, но добавляются из админки)

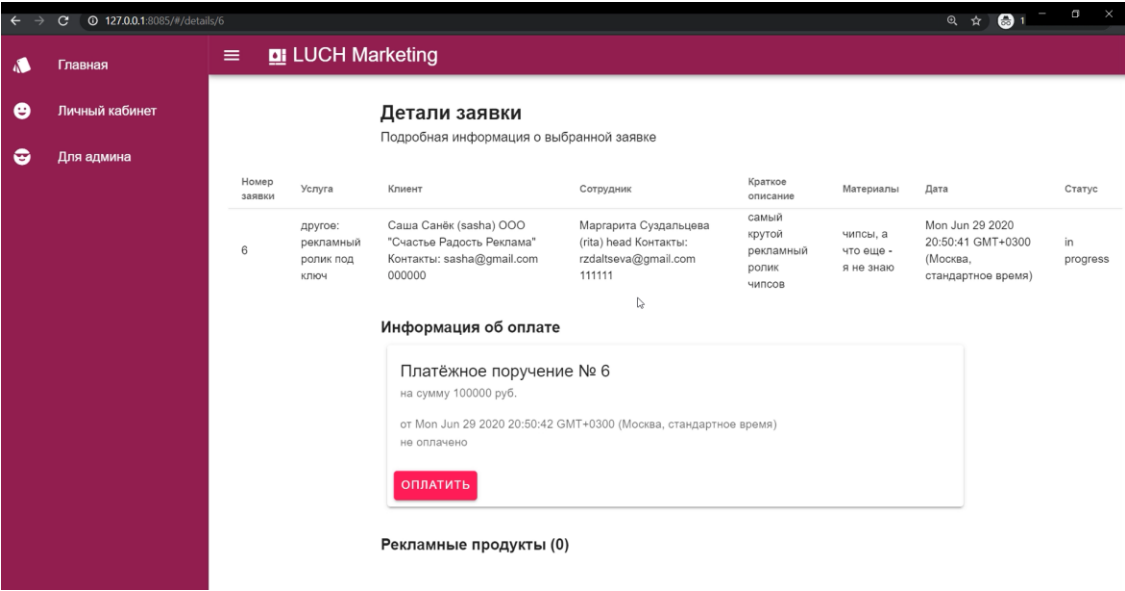


Рисунок 26 – Интерфейс «Детали заявки»

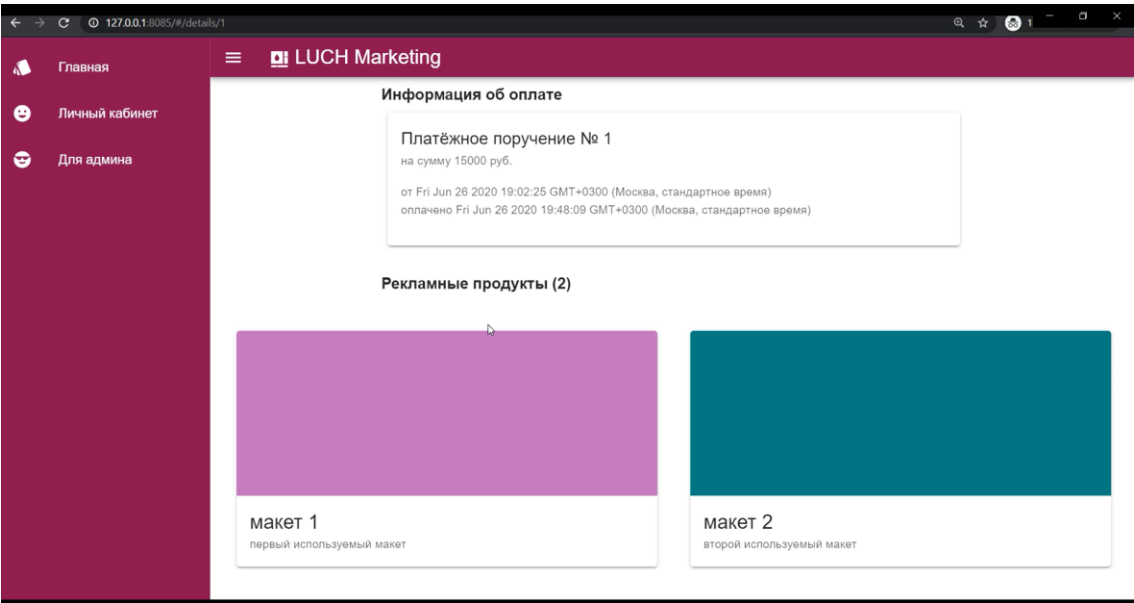


Рисунок 27 – Интерфейс «Детали заявки» (2)

Доступ: для авторизованных пользователей, переадресация на авторизацию

Выполненные запросы:

Таблица 4 – Запросы в Details.vue

POSTMAN	axios	Назначение
GET http://127.0.0.1:8000/ api/request/<int:pk>	.get(`http://\${window.location.hostname}: 8000/api/request/\${this.id}`)	Получение заявки с id, который получен при нажатии на кнопку «подробнее» в кабинете, заполнение таблицы
GET http://127.0.0.1:8000/	.get(`http://\${window.location.hostname}: 8000/api/product/all?req=\${this.id}`)	Получение рекламных продуктов заявки, заполнение карточек

api/product/all?req=<request__id>		
GET http://127.0.0.1:8000/ api/payment/all?req= <request__id>	.get('http://\${ window.location.hostname }: 8000/api/payment/	Получение платёжного поручения заявки, заполнение карточки

3.5. Кабинет руководителя

Кабинет руководителя (рис. 28-33) позволяет просматривать все существующие заявки, фильтровать их по всем полям (при этом указывается число найденных заявок), отмечать заявки как выполненные нажатием на кнопку (доступна для заявок in progress, аналогично v-if). Также есть кнопка перехода к деталям на карточках. Кнопка «очистить» сбрасывает фильтры.

Название компоненты: AdminCab.vue

Элементы: фильтры (статус, услуга, с, по, сотрудник, клиент), кнопки «отфильтровать» и «очистить», карточки с краткой информацией о заявке и двумя кнопками – «подробнее» и «отметить как завершённую»

Доступ: только для сотрудника с позицией руководитель

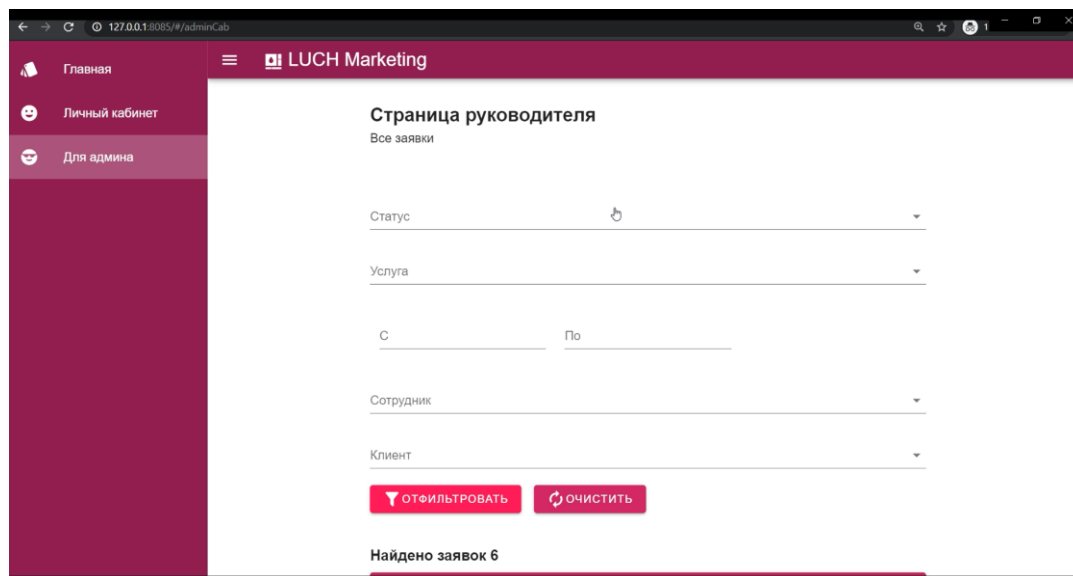


Рисунок 28 – Интерфейс «Кабинет руководителя»

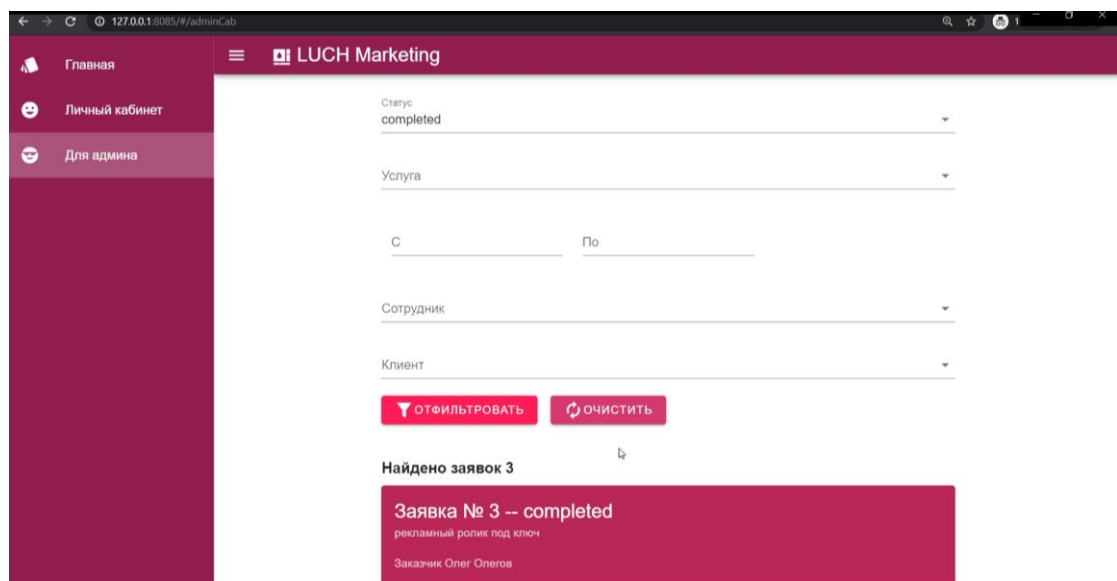


Рисунок 29 – Интерфейс «Кабинет руководителя» (2)

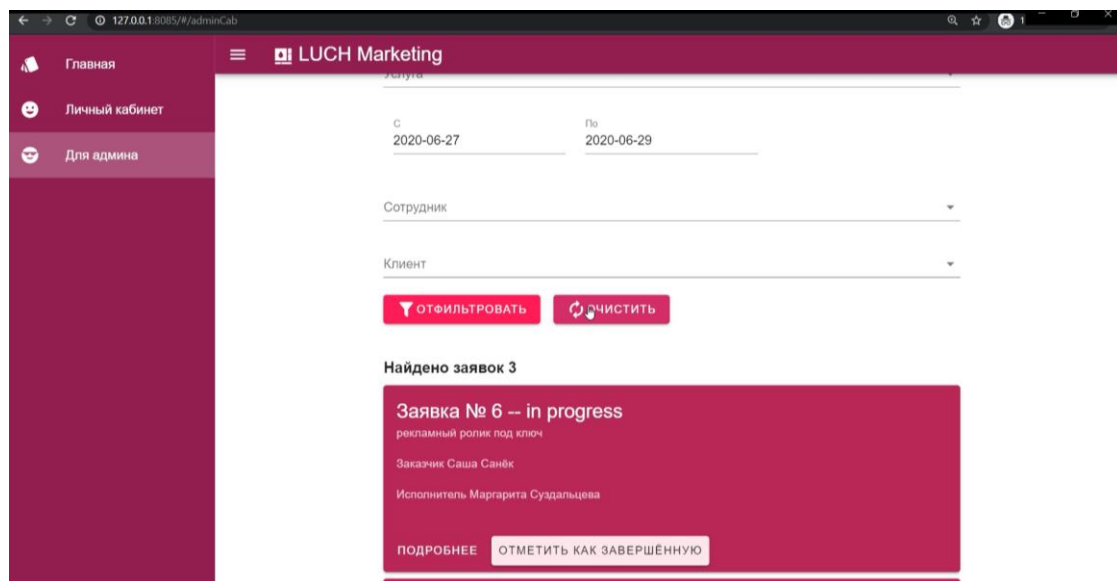


Рисунок 30 – Интерфейс «Кабинет руководителя» (3)

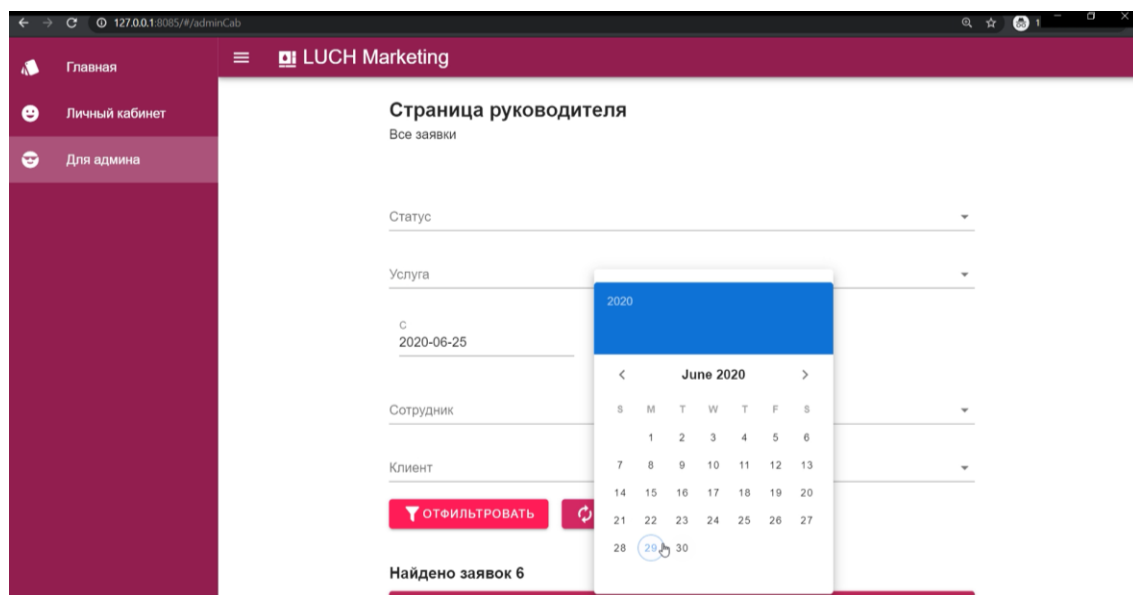


Рисунок 31 – Интерфейс «Кабинет руководителя» (3)

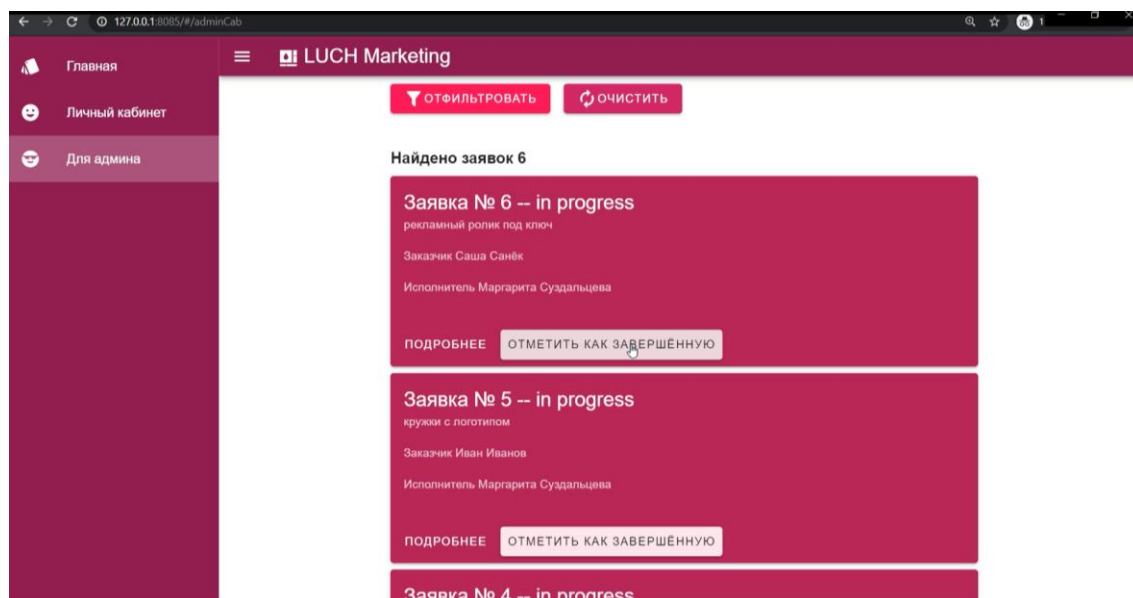


Рисунок 32 – Интерфейс «Кабинет руководителя» (4)

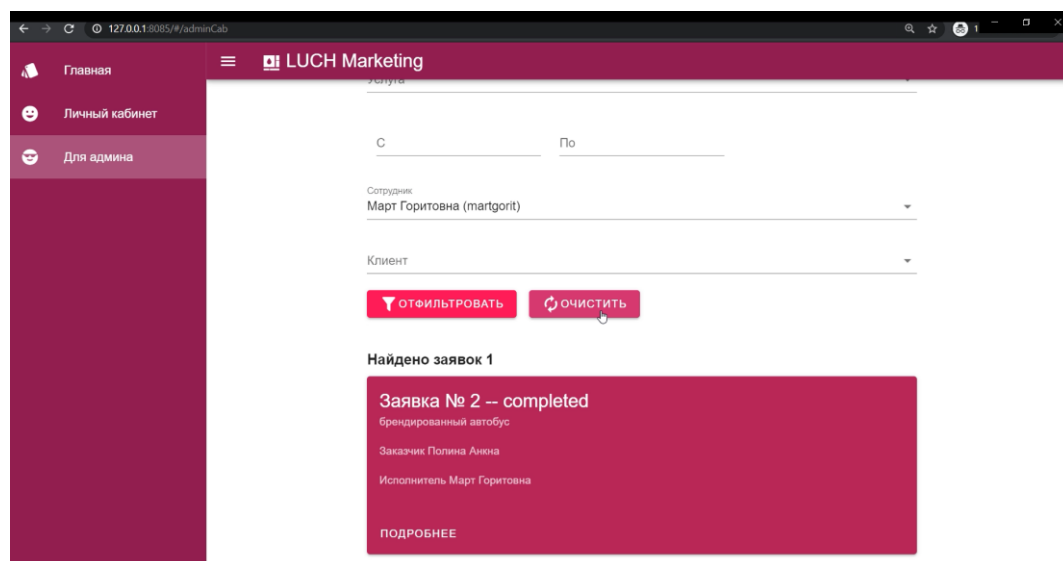


Рисунок 33 – Интерфейс «Кабинет руководителя» (5)

Выполненные запросы:

Таблица 5 – Запросы в AdminCab.vue

POSTMAN	axios	Назначение
GET http://127.0.0.1:8000/ api/gethead?user=<user_username>	.get(`http://\${window.location.hostname}:8000/api/gethead?user=\${sessionStorage.getItem('user')}`)	Узнать, руководитель ли пользователь, иначе – переадресация на главную
GET http://127.0.0.1:8000/ api/service/all	.get(`http://\${window.location.hostname}:8000/api/service/all`)	Получение всех услуг для выпадающего списка в фильтре
GET http://127.0.0.1:8000/ api/employee/all	.get(`http://\${window.location.hostname}:8000/api/employee/all`)	Получение всех сотрудников для выпадающего списка в фильтре
GET http://127.0.0.1:8000/ api/client/all	.get(`http://\${window.location.hostname}:8000/api/client/all`)	Получение всех клиентов для выпадающего списка в фильтре
GET http://127.0.0.1:8000/ api/request/all	.get(`http://\${window.location.hostname}:8000/api/request/all`)	Изначальное получение всех заявок, заполнение карточек
GET http://127.0.0.1:8000/ api/request/adminfilter<params>	.get(`http://\${window.location.hostname}:8000/api/request/adminfilter?status=\${this.req_status}&service=\${this.req_service}&after=\${this.req_after}&before=\${this.req_before}&employee=\${this.req_employee}&client=\${this.req_client}`)	Основной фильтр эндпоинта adminfilter

PATCH http://127.0.0.1:8000/ api/request/<int:pk>? status=1	.patch(`http://\${window.location.hostname}:8000/api/request/\${id}`, data)	Редактирование статуса выбранной заявки (in progress заменяется на completed)
--	---	--

3.6. Форма заявки на рекламу

Данный интерфейс (рис. 34-35) позволяет клиенту оставить новую заявку на рекламу. Вместе с созданием заявки автоматически создаётся платёжное поручение.

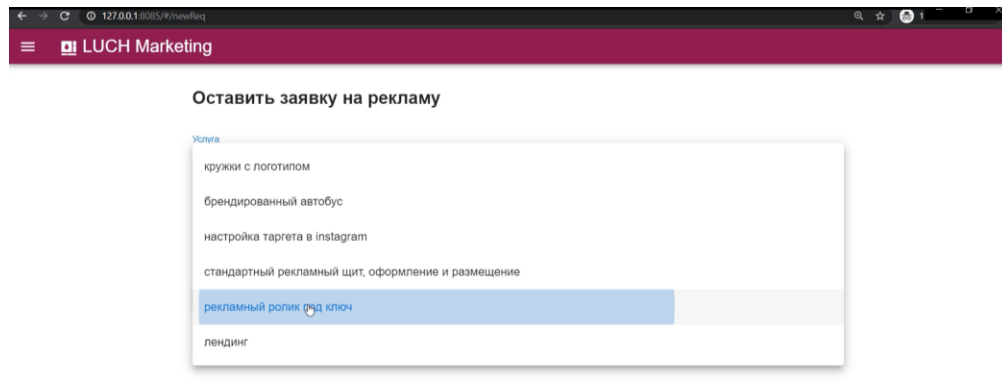


Рисунок 35 – Интерфейс «Форма заявки на рекламу»

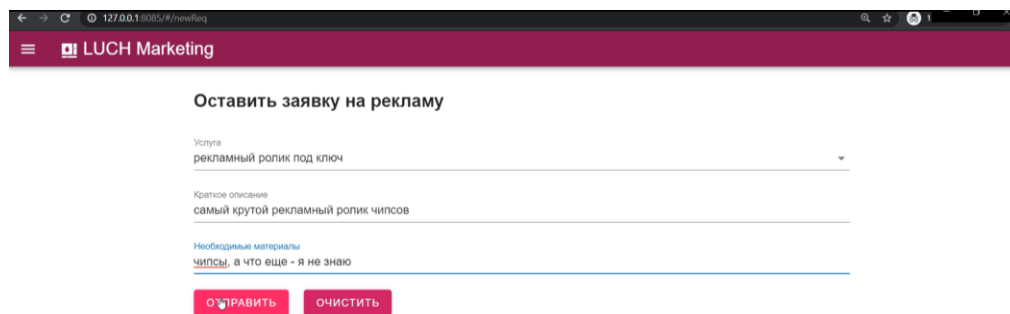


Рисунок 36 – Интерфейс «Форма заявки на рекламу» (2)

Название компоненты: ReqForm.vue

Элементы: форма с выпадающим списком услуг и полями «краткое описание» и «необходимые материалы», две кнопки – «отправить», «очистить»

Доступ: для авторизованных пользователей, переадресация на авторизацию

Выполненные запросы:

Таблица 6 – Запросы в ReqForm.vue

POSTMAN	axios	Назначение
GET http://127.0.0.1:8000/ api/getclient?user=<u ser__username>	.get(`http://\${window.location.hostname}: 8000/api/getclient?user=\${sessionStorage. getItem('user')}`)	Получение данных клиента по username пользователя
GET http://127.0.0.1:8000/ api/request/<int:pk>	.get(`http://\${window.location.hostname}: 8000/api/request/\${this.id}`)	Получение заявки с id, который получен при нажатии на кнопку «подробнее» в кабинете, заполнение таблицы
POST http://127.0.0.1:8000/ api/request/new/?<pa rams>	.post(`http://\${window.location.hostname}: 8000/api/request/new/`, data)	Создание заявки с параметрами из формы (data)
GET http://127.0.0.1:8000/ api/request/<int:pk>	.get(`http://\${window.location.hostname}: 8000/api/request/\${this.req_id}`)	Получение по id заявки (только что созданной) и затем цены услуги, чтобы создать платёжку
GET http://127.0.0.1:8000/ api/service/all	.get(`http://\${window.location.hostname}: 8000/api/service/all`)	Получение всех услуг для выпадающего списка в форме
POST http://127.0.0.1:8000/ api/payment/new/?<p arams>	.post(`http://\${window.location.hostname}: 8000/api/payment/new`, paydata)	Создание платёжки с нужными данными (paydata)

3.7. Регистрация клиента, авторизация

Данный интерфейс (рис. 37-38) нужен для регистрации пользователя (по стандартным адресам Djoser), но также добавляется клиент в таблицу Client и компания в Company. После - переадресация на авторизацию. Сотрудников, в свою очередь, регистрируют через админку, но им доступна та же авторизация.

Название компоненты: RegClient.vue

Элементы: форма с полями для таблицы User (логин, почта, пароль) и полями для таблицы Client (имя, фамилия, выпадающий список типов компаний, название компании, телефон), кнопка «зарегистрироваться»

Доступ: для всех

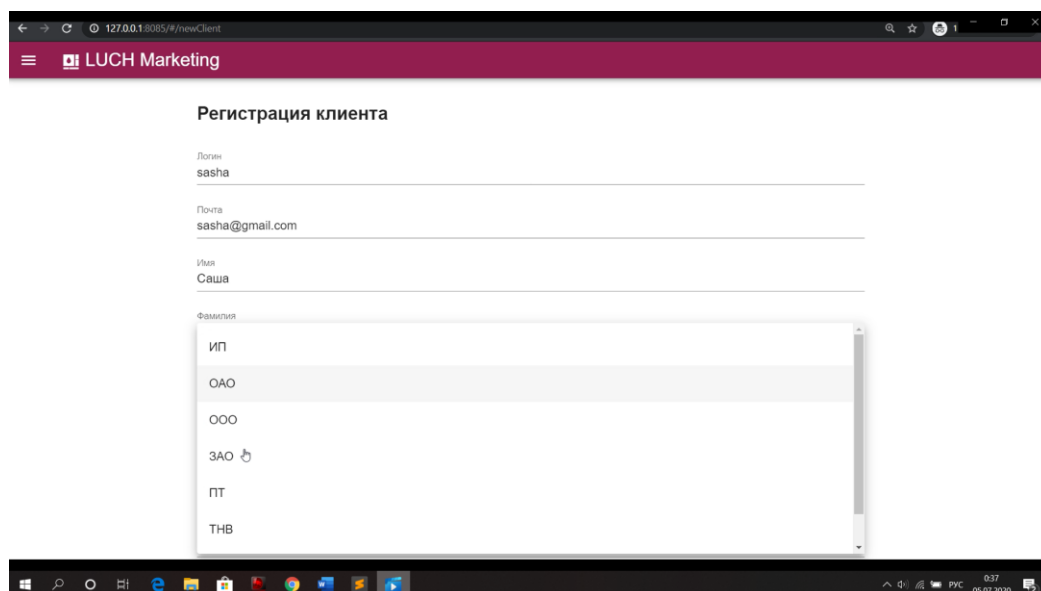


Рисунок 37 – Интерфейс «Форма регистрации клиента»

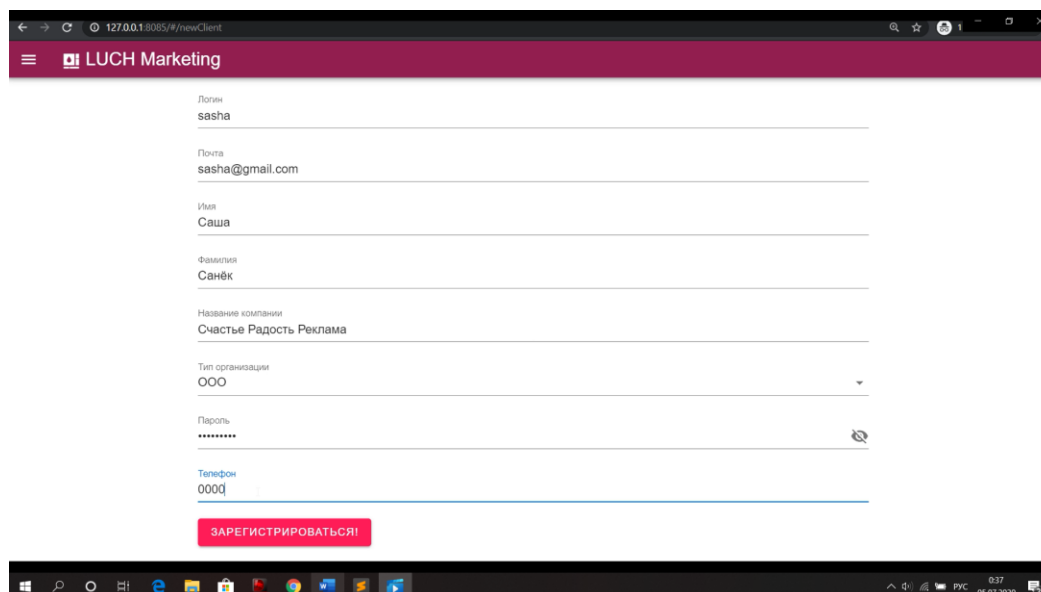


Рисунок 38 – Интерфейс «Форма регистрации клиента» (2)

Выполненные запросы:

Таблица 7 – Запросы в RegClient.vue

POSTMAN	axios	Назначение
POST http://127.0.0.1:8000/api/auth/users/<params>	.post(`http://\${window.location.hostname}:8000/api/auth/users/`, userdata)	Создание пользователя (POST-запрос с userdata)
POST http://127.0.0.1:8000/api/company/new<params>	.post(`http://\${window.location.hostname}:8000/api/company/new`, compdata)	Создание новой компании (compdata)

POST http://127.0.0.1:8000/api/ client/new/?<params>	.post(`http://\${window.location.hostna me}:8000/api/client/new`, clientdata)	Создание клиента с FK на созданного пользователя (clientdata)
--	--	--

Форма авторизации выглядит таким образом:

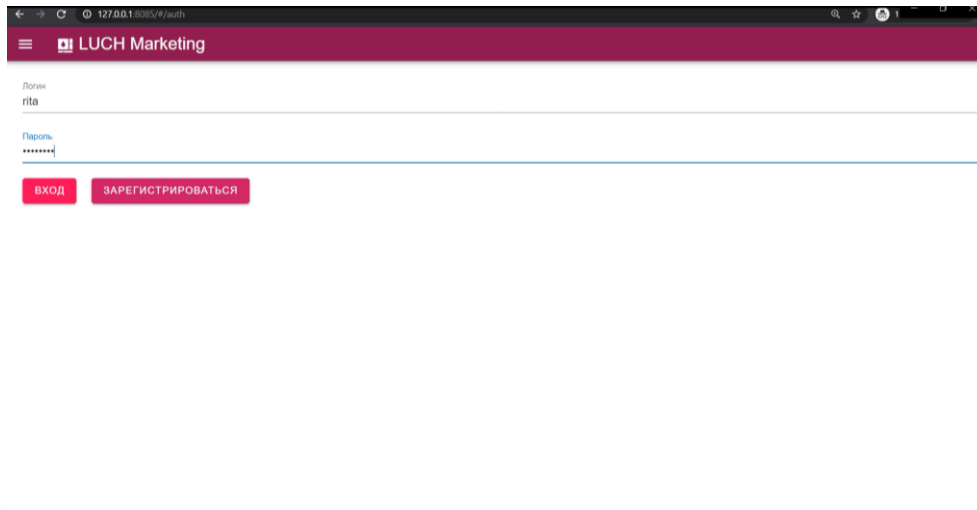


Рисунок 39 – Интерфейс «Форма авторизации»

Авторизация по токenu стандартная и реализована средствами Djoser (в компоненте Auth.vue).

4. ЗАКЛЮЧЕНИЕ

Цель курсовой работы состояла в реализации web-сервиса marketing (маркетинговое агентство «ЛУЧ»), которое упростило бы организацию работы с клиентами, а именно учёт поступающих и выполненных заявок.

Предложенные для решения задачи технологии, а именно PostgreSQL, Django REST framework и Vue.js, позволил быстро и относительно просто создать целостное web-приложение, отвечающее сформулированным функциональным требованиям.

В дальнейшем созданный сервис может быть доработан путём создания более привлекательного дизайна и добавления большего количества возможных запросов и отчётов (а то есть и новых интерфейсов, так как из варианта задания были выбраны либо переработаны только основные функции). Также может быть, например, реализован чат для общения между клиентом и исполнителем в самом сервисе. Тем не менее, задачи, описанные в начале курсовой, выполнены в полной мере.

СПИСОК ЛИТЕРАТУРЫ

1. Django Rest Framework. Документация Django Rest Framework [Электронный ресурс]. URL: <https://www.django-rest-framework.org> (дата обращения: 01.07.2020).
2. Vue.js. Документация Vue.js [Электронный ресурс]. URL: <https://vuejs.org> (дата обращения: 01.07.2020).
3. Vuetify. Документация Vuetify [Электронный ресурс]. URL: <https://vuetifyjs.com/ru/> (дата обращения: 01.07.2020).
4. Djoser. Документация Djoser 2.0.1 [Электронный ресурс]. URL: <https://djoser.readthedocs.io/en/latest/> (дата обращения: 01.07.2020).

Приложение 1. Dockerfile & docker-compose.yml

```
1  version: '3'
2
3  services:
4    marketing_db:
5      image: postgres
6      ports:
7        - "5436:5432"
8      environment:
9        - POSTGRES_USER=postgres
10       - POSTGRES_PASSWORD=postgres
11       - POSTGRES_DB=marketingdb
12      volumes:
13        - ./dbs/postgres-data:/var/lib/postgresql
14    backend:
15      container_name: marketing
16      build: ./marketing
17      command: bash -c "
18        sleep 3 &&
19        python3 manage.py makemigrations && python3 manage.py migrate &&
20        python3 manage.py runserver --insecure 0.0.0.0:8000";
21      volumes:
22        - ./marketing:/marketing
23      ports:
24        - "8000:8000"
25      depends_on:
26        - marketing_db
27    frontend:
28      container_name: vue-marketing
29      build:
30        context: ./vue-marketing
31        dockerfile: Dockerfile
32      command: npm start --start;
33      volumes:
34        - ./vue-marketing:/vue-marketing
35        - /vue-marketing/node_modules
36      ports:
37        - "8085:8080"
38      depends_on:
39        - backend
```

```
1  FROM python:3.6.9
2
3  ENV PYTHONBUFFERED 1
4
5  RUN mkdir /marketing
6
7  WORKDIR /marketing
8
9  COPY . /marketing
10
11 RUN pip3 install -r requirements.txt
```

```
1  FROM node:12
2
3  WORKDIR /vue-marketing
4
5  COPY package*.json ./
6
7  RUN npm i
8
9  COPY . .
```

Приложение 2. models.py

```
1  from django.db import models
2  from django.contrib.auth.models import User
3
4  class Company(models.Model):
5      name = models.CharField(max_length=50)
6      TYPES = [
7          ('1', 'ИП'),
8          ('2', 'ООО'),
9          ('3', 'ООО'),
10         ('4', 'ЗАО'),
11         ('5', 'ПТ'),
12         ('6', 'ТНВ'),
13         ('7', 'ГКП')
14         # и другие виды ОПФ
15     ]
16     company_type = models.CharField(max_length=1, choices=TYPES)
17
18
19  class Service(models.Model):
20      name = models.CharField(max_length=100)
21      TYPES = [
22          ('1', 'реклама в СМИ'),
23          ('2', 'наружная реклама'),
24          ('3', 'реклама на транспорте'),
25          ('4', 'реклама на месте продаж'),
26          ('5', 'сувенирка'),
27          ('6', 'печатная реклама'),
28          ('7', 'директ-реклама'),
29          ('8', 'реклама в интернете'),
30          ('9', 'ивент-реклама'),
31          ('10', 'другое')
32          # и т.д. и т.п.
33     ]
34     service_type = models.CharField(max_length=2, choices=TYPES)
35     price = models.IntegerField()
36
```

```

37
38 class Employee(models.Model):
39     user = models.ForeignKey(User, on_delete=models.CASCADE)
40     POSITIONS = [
41         ('e', 'executor'),
42         ('h', 'head')
43     ]
44     position = models.CharField(max_length=1, choices=POSITIONS, default='e')
45     phone = models.CharField(max_length=14)
46     first_name = models.CharField(max_length=150, null=True)
47     last_name = models.CharField(max_length=150, null=True)
48
49
50 class Client(models.Model):
51     user = models.ForeignKey(User, on_delete=models.CASCADE)
52     company = models.ForeignKey(Company, on_delete=models.CASCADE)
53     phone = models.CharField(max_length=14)
54     first_name = models.CharField(max_length=150, null=True)
55     last_name = models.CharField(max_length=150, null=True)
56
57
58 class Request(models.Model):
59     service = models.ForeignKey(Service, on_delete=models.CASCADE)
60     client = models.ForeignKey(Client, on_delete=models.CASCADE)
61     employee = models.ForeignKey(Employee, on_delete=models.CASCADE, default="1")
62     description = models.CharField(max_length=280)
63     materials = models.CharField(max_length=280)
64     date = models.DateTimeField(auto_now_add=True, null=True)
65     STATUSES = [
66         ('0', 'in progress'),
67         ('1', 'completed')
68     ]
69     status = models.CharField(max_length=1, choices=STATUSES, default='0')
70
71
72 class Payment(models.Model):
73     request = models.ForeignKey(Request, on_delete=models.CASCADE)
74     amount = models.IntegerField()
75     day_requested = models.DateTimeField(auto_now_add=True, null=True)
76     day_paid = models.DateTimeField(null=True, blank=True)
77     STATUSES = [
78         ('0', 'unpaid'),
79         ('1', 'paid')
80     ]
81     status = models.CharField(max_length=1, choices=STATUSES, default='0')
82
83
84 class Product(models.Model):
85     request = models.ForeignKey(Request, on_delete=models.CASCADE)
86     name = models.CharField(max_length=100)
87     description = models.CharField(max_length=280)
88     image = models.CharField(blank=True, max_length=280)
89

```

Приложение 3. serializers.py

```
1  from rest_framework import serializers
2  from .models import *
3  from django.contrib.auth.models import User
4
5
6  class UserSerializer(serializers.ModelSerializer):
7      class Meta:
8          model = User
9          fields = ['id', 'username', 'email']
10
11
12  class NewCompanySerializer(serializers.ModelSerializer):
13      class Meta:
14          model = Company
15          fields = '__all__'
16
17
18  class CompanySerializer(serializers.ModelSerializer):
19      company_type = serializers.CharField(source='get_company_type_display')
20      class Meta:
21          model = Company
22          fields = '__all__'
23
24  class ServiceSerializer(serializers.ModelSerializer):
25      service_type = serializers.CharField(source="get_service_type_display")
26      class Meta:
27          model = Service
28          fields = '__all__'
29
30
```

```

31 class ClientSerializer(serializers.ModelSerializer):
32     user = UserSerializer(read_only=True)
33     company = CompanySerializer(read_only=True)
34
35     class Meta:
36         model = Client
37         fields = '__all__'
38
39
40 class NewClientSerializer(serializers.ModelSerializer):
41     class Meta:
42         model = Client
43         fields = '__all__'
44
45
46 class EmployeeSerializer(serializers.ModelSerializer):
47     user = UserSerializer()
48     position = serializers.CharField(source="get_position_display")
49
50     class Meta:
51         model = Employee
52         fields = '__all__'
53
54
55 class CreateRequestSerializer(serializers.ModelSerializer):
56     class Meta:
57         model = Request
58         fields = '__all__'
59
60
61 class RequestSerializer(serializers.ModelSerializer):
62     #status = serializers.CharField(source="get_status_display")
63     service = ServiceSerializer(read_only=True)
64     client = ClientSerializer(read_only=True)
65     employee = EmployeeSerializer(read_only=True)
66     class Meta:
67         model = Request
68         fields = '__all__'
69
70
71 class PaymentSerializer(serializers.ModelSerializer):
72     class Meta:
73         model = Payment
74         fields = '__all__'
75
76
77 class ProductSerializer(serializers.ModelSerializer):
78     class Meta:
79         model = Product
80         fields = '__all__'
81

```

Приложение 4. views.py

```
1  from django.shortcuts import render
2  from rest_framework.response import Response
3  from rest_framework.views import APIView
4  from rest_framework import generics
5  from django.db.models import Q
6  from django.contrib.auth import get_user_model
7
8  from .serializers import *
9  from .models import *
10
11 # Requests
12 class CreateRequestView(generics.CreateAPIView):
13     queryset = Request.objects.all()
14     serializer_class = CreateRequestSerializer
15
16
17 class GetRequestView(generics.RetrieveUpdateDestroyAPIView):
18     queryset = Request.objects.all()
19     serializer_class = RequestSerializer
20
21
22 class GetRequestsView(generics.ListAPIView):
23     serializer_class = RequestSerializer
24     def get_queryset(self):
25         queryset = Request.objects.all()
26         user = self.request.query_params.get('user', None)
27         #client = self.request.query_params.get('client', None)
28         #employee = self.request.query_params.get('employee', None)
29         #if client:
30             #    queryset = queryset.filter(client=client)
31         if user:
32             queryset = queryset.filter(Q(employee__user__username=user) | Q(client__user__username=user))
33         queryset = queryset.order_by('-date')
34         return queryset
35
36
```

```

37 class AdminFilterView(generics.ListAPIView):
38     serializer_class = RequestSerializer
39
40     def get_queryset(self):
41         queryset = Request.objects.all()
42         params = self.request.query_params
43
44         service = params.get('service', None)
45         client = params.get('client', None)
46         employee = params.get('employee', None)
47         status = params.get('status', None)
48         after = params.get('after', None)
49         before = params.get('before', None)
50
51         if service:
52             queryset = queryset.filter(service__id=service)
53
54         if client:
55             queryset = queryset.filter(client__id=client)
56
57         if employee:
58             queryset = queryset.filter(employee__id=employee)
59
60         if status:
61             queryset = queryset.filter(status=status)
62
63         if after:
64             queryset = queryset.filter(date__date__gte=after)
65
66         if before:
67             queryset = queryset.filter(date__date__lte=before)
68
69         queryset = queryset.order_by('-date')
70         return queryset
71

```

```

72
73 # Company & Client
74 class CreateClientView(generics.CreateAPIView):
75     queryset = Client.objects.all()
76     serializer_class = NewClientSerializer
77
78
79 class CreateCompanyView(generics.CreateAPIView):
80     queryset = Company.objects.all()
81     serializer_class = NewCompanySerializer
82
83
84 class GetClientView(generics.ListAPIView):
85     serializer_class = ClientSerializer
86     def get_queryset(self):
87         params = self.request.query_params
88         user = params.get('user', None)
89         queryset = Client.objects.all()
90         if user:
91             queryset = queryset.filter(user__username=user)
92         return queryset
93
94
95 # All employees & clients
96 class GetClientsView(generics.ListAPIView):
97     queryset = Client.objects.all()
98     serializer_class = ClientSerializer
99
100
101 class GetEmployeesView(generics.ListAPIView):
102     queryset = Employee.objects.all()
103     serializer_class = EmployeeSerializer
104

```

```
105
106 # Check if employee is head
107 class GetHeaderView(generics.ListAPIView):
108     serializer_class = EmployeeSerializer
109     def get_queryset(self):
110         params = self.request.query_params
111         user = params.get('user', None)
112         queryset = Employee.objects.all()
113         if user:
114             queryset = queryset.filter(Q(user__username=user) & Q(position='h'))
115         return queryset
116
117
118 # Payments
119 class CreatePaymentView(generics.CreateAPIView):
120     queryset = Payment.objects.all()
121     serializer_class = PaymentSerializer
122
123
124 class GetPaymentView(generics.RetrieveUpdateDestroyAPIView):
125     queryset = Payment.objects.all()
126     serializer_class = PaymentSerializer
127
128
129 class GetPaymentsView(generics.ListAPIView):
130     serializer_class = PaymentSerializer
131     def get_queryset(self):
132         queryset = Payment.objects.all()
133         params = self.request.query_params
134         req = params.get('req', None)
135         if req:
136             queryset = queryset.filter(request__id=req)
137         return queryset
138
139
```



```

140 # Services
141 class GetServiceView(generics.RetrieveAPIView):
142     queryset = Service.objects.all()
143     serializer_class = ServiceSerializer
144
145
146 class GetServicesView(generics.ListAPIView):
147     queryset = Service.objects.all()
148     serializer_class = ServiceSerializer
149
150
151 # Products
152 class GetProductsView(generics.ListAPIView):
153     serializer_class = ProductSerializer
154     def get_queryset(self):
155         queryset = Product.objects.all()
156         params = self.request.query_params
157         req = params.get('req', None)
158         if req:
159             queryset = queryset.filter(request__id=req)
160         return queryset
161

```