

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»  
(БГТУ им. В.Г.Шухова)**

Расчетно-графическое задание  
дисциплина:  
Технологии web-программирования

Выполнил: ст. группы ВТ-41  
Фаракшин Н. Р.  
Проверил: ст. пр. Картамышев С.В.

**Белгород 2020**

## Содержание

**Элементы оглавления не найдены.**

## **Введение**

Веб-приложение сегодня – это популярный способ создания кроссплатформенных приложений с клиент-серверной архитектурой. Технология одностраничных приложений позволяет с легкостью создавать интерактивные веб-страницы и позволять пользователю взаимодействовать с ними, не ожидая перезагрузки данных, и в то же время сохраняя результаты работы на удаленном сервере. Для функционирования такого приложения необходим только веб-браузер с поддержкой современных стандартов; такой браузер можно найти практически для любой актуальной операционной системы.

## **Цель работы**

Целью расчетно-графического задания является создание веб-приложения и изучение в ходе выполнения следующих вопросов:

1. Работа протокола передачи гипертекста HTTP, языка разметки HTML и CSS.
2. Основы паработки frontend-приложений, в частности, с использованием языка JavaScript и библиотеки React.
3. Основы работы технологии Docker.
4. Основы разработки REST API, в частности, с помощью фреймворков Django и Django REST Framework.
5. Основы взаимодействия frontend-приложения с backend-приложением.

## **Выполнение**

В качестве предметной области выбран многопользовательский блог. Веб-приложение должно предоставлять следующие возможности:

- Создание и просмотр постов (записей) пользователей
- Возможность оставлять и просматривать комментарии
- Возможность просматривать профили пользователей

## **Глава 1. Верстка HTML**

После описания концепции приложения, была создана логическая структура сайта с помощью языка разметки HTML. Верстка макетов далее производилась с помощью языка описания стилей CSS и библиотеки Bootstrap, предоставляющей единую систему стилей для веб-страниц. Для добавления иконок была использована CSS-библиотека FontAwesome.

Исходный код некоторых страниц можно найти в приложении А.

Результатом выполнения этого этапа стали следующие макеты:

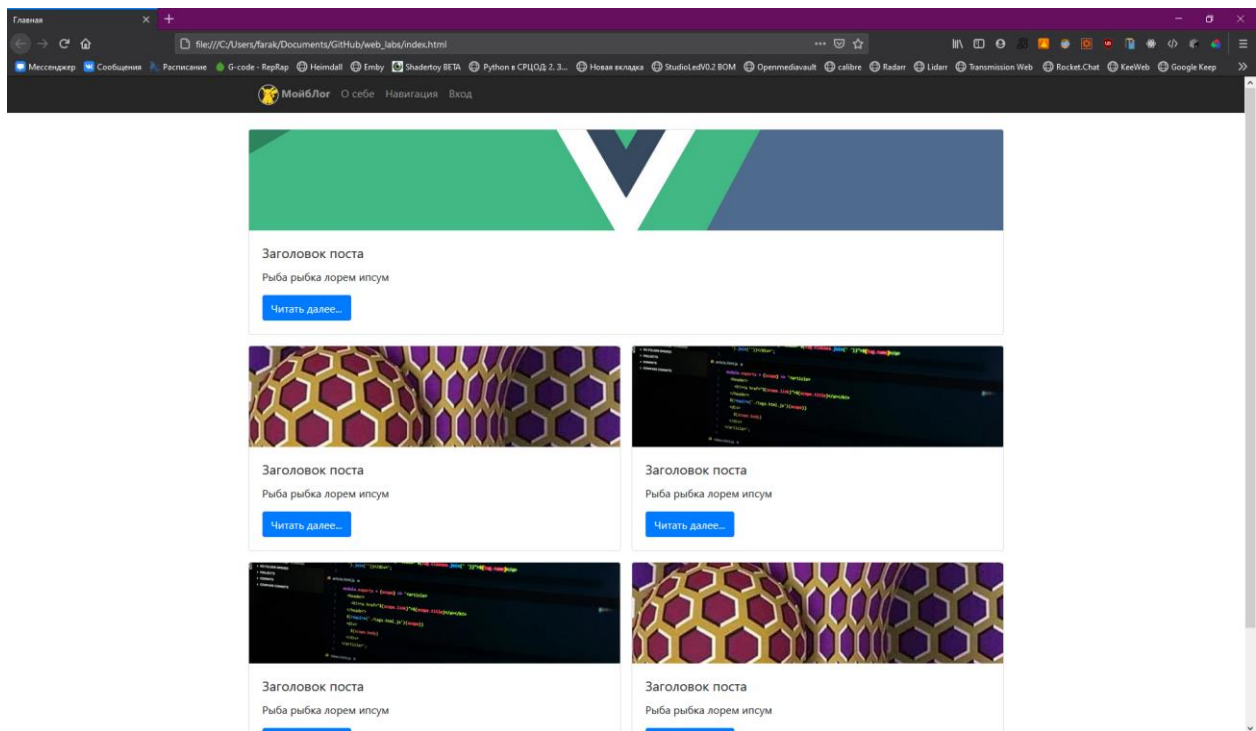


Рис. 1 Главная страница

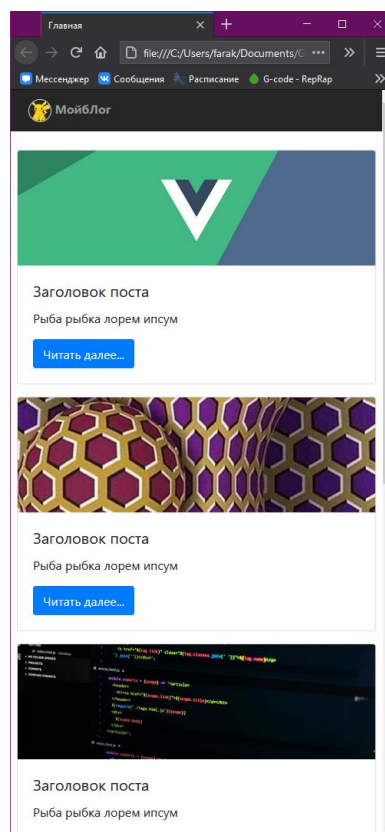


Рис. 2 Главная страница на узком экране

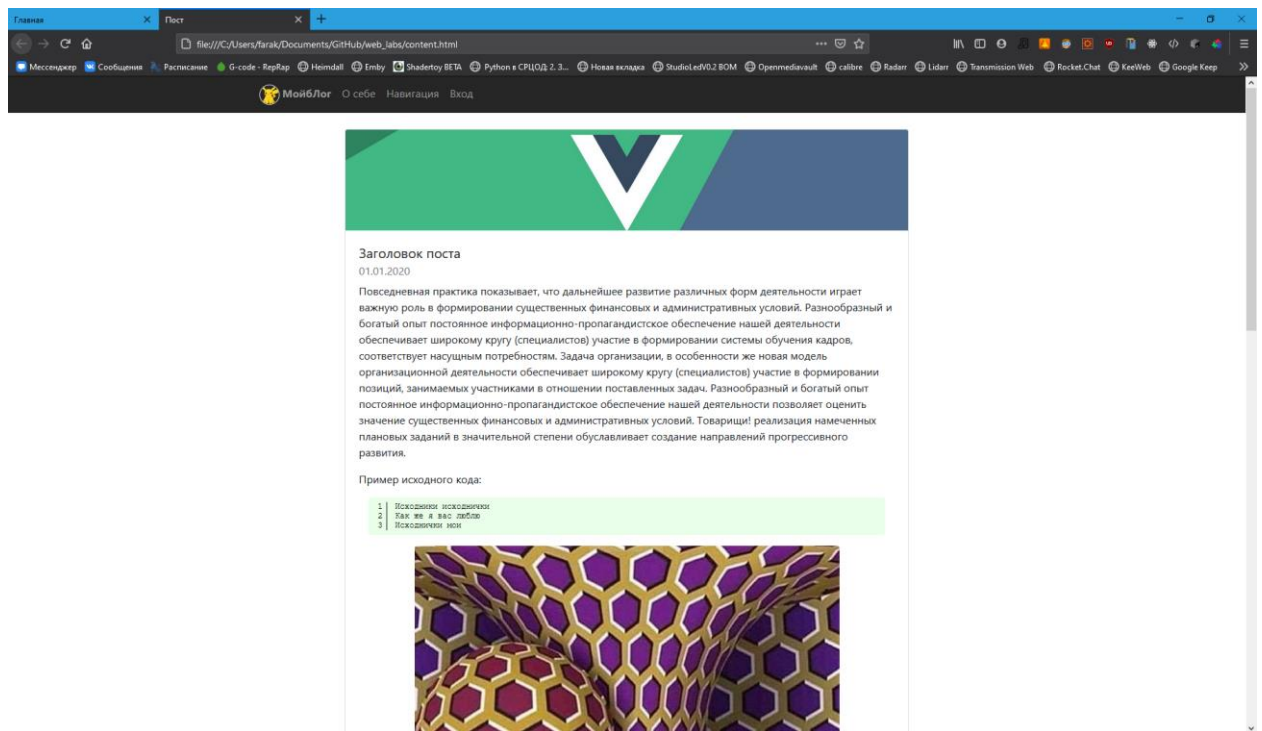


Рис. 3 Страница с контентом

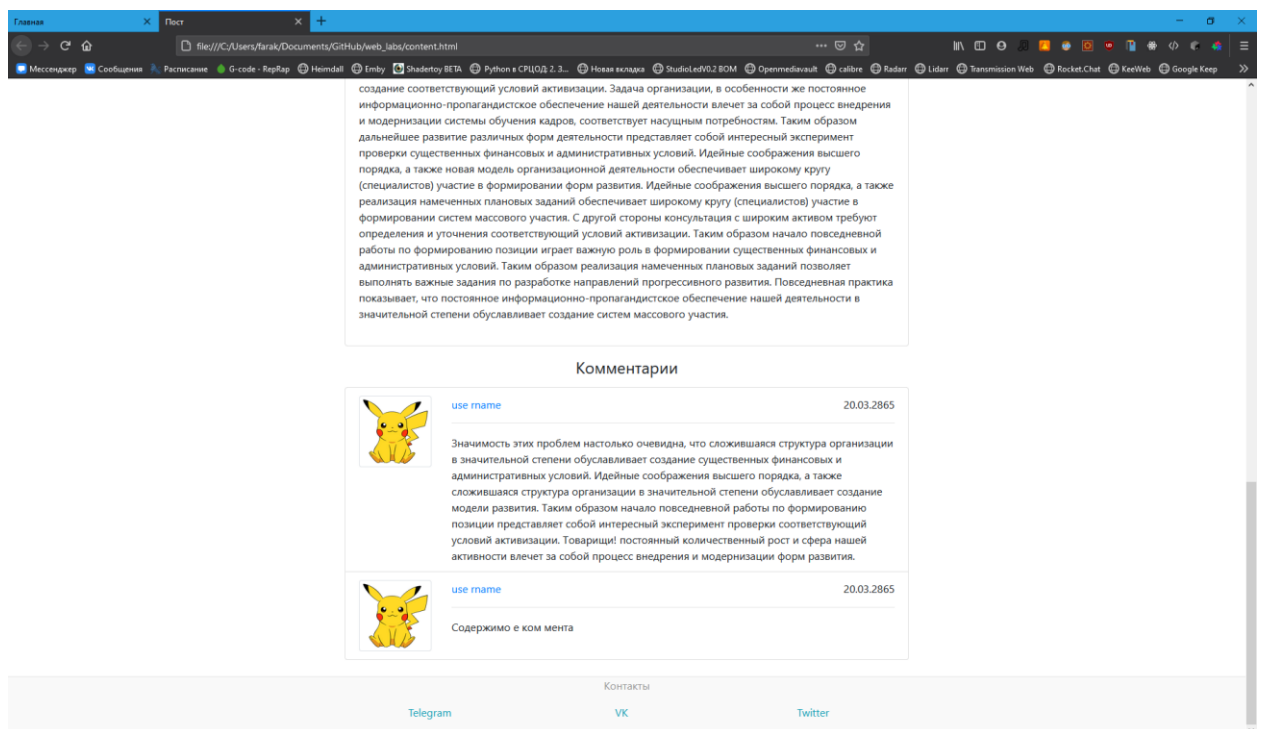
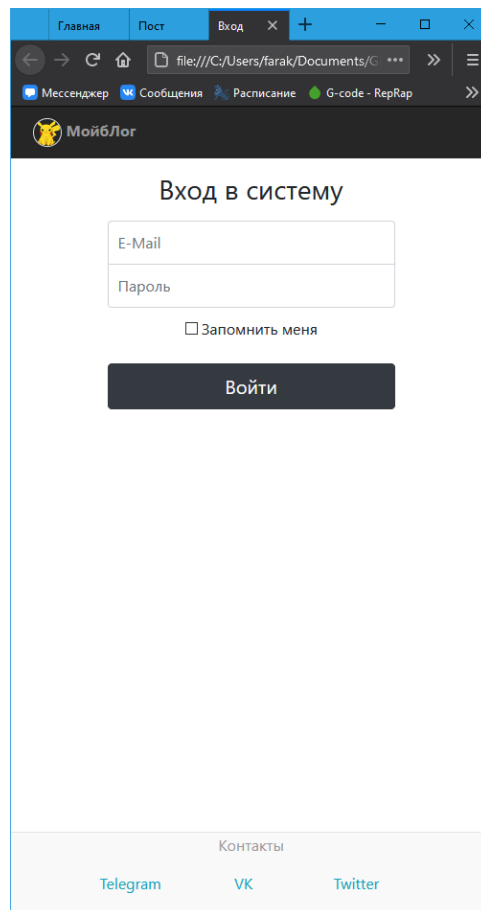


Рис. 4 Комментарии на странице с контентом



*Рис. 5 Форма входа*

## Глава 2. Приложение React

Полученные на предыдущем этапе макеты были перенесены на фреймворк React в виде компонентов. Это позволило переиспользовать некоторые элементы, такие, как Header и Footer, а также добавить логику работы со структурами данных в приложение.

Исходный код некоторых компонентов можно найти в приложении.

Результатом данного этапа стали страницы следующего вида:

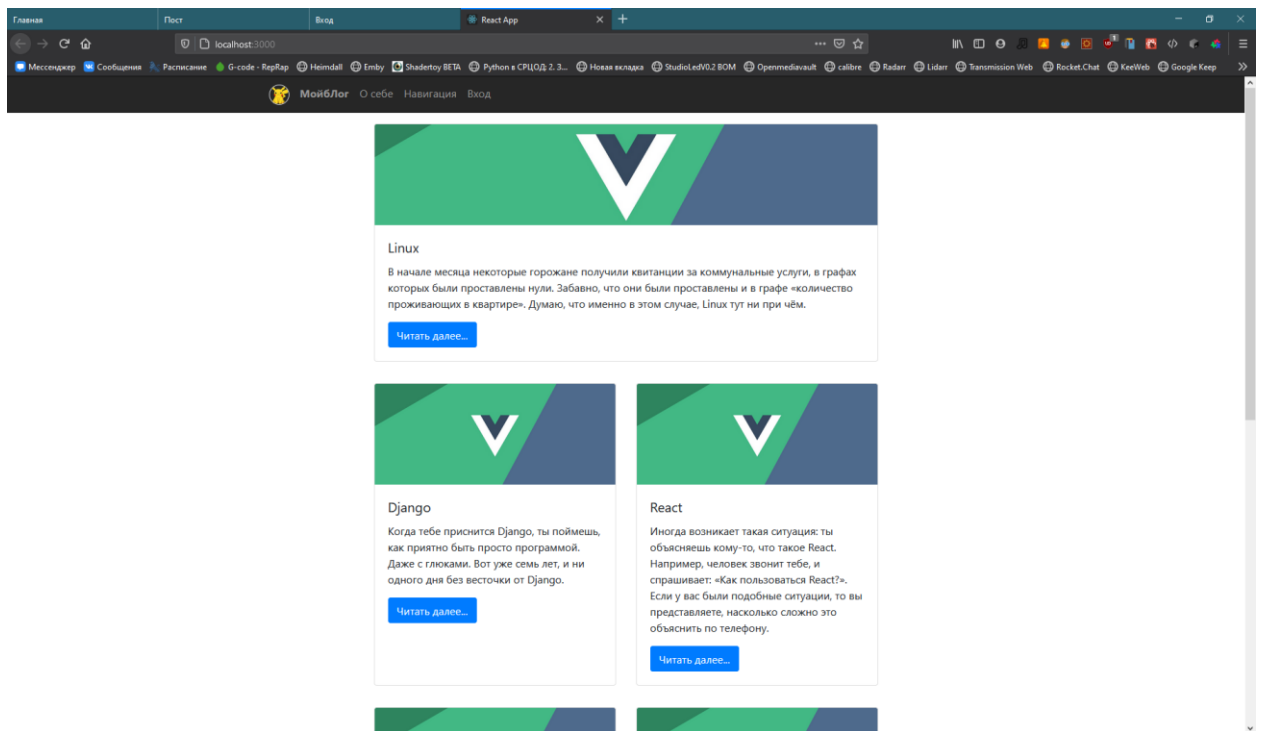


Рис. 6 Карточки постов на главном экране

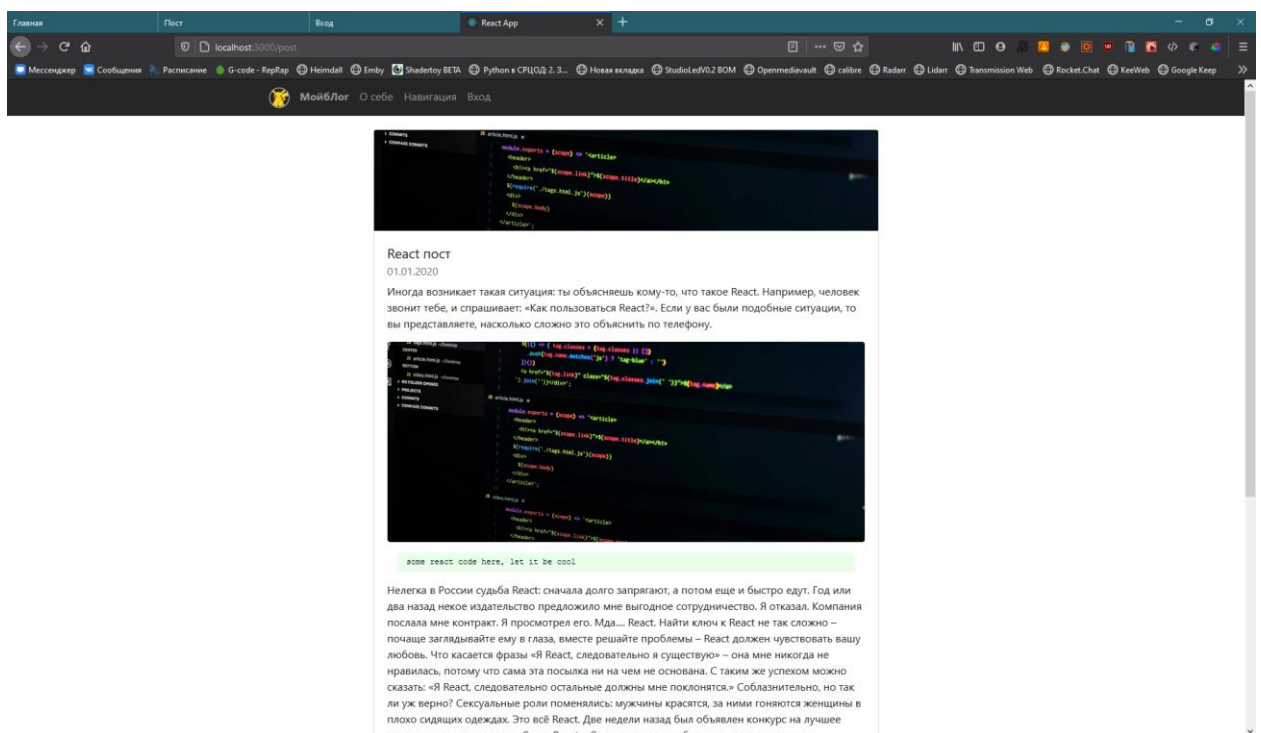


Рис. 7 Компонент с содержимым поста

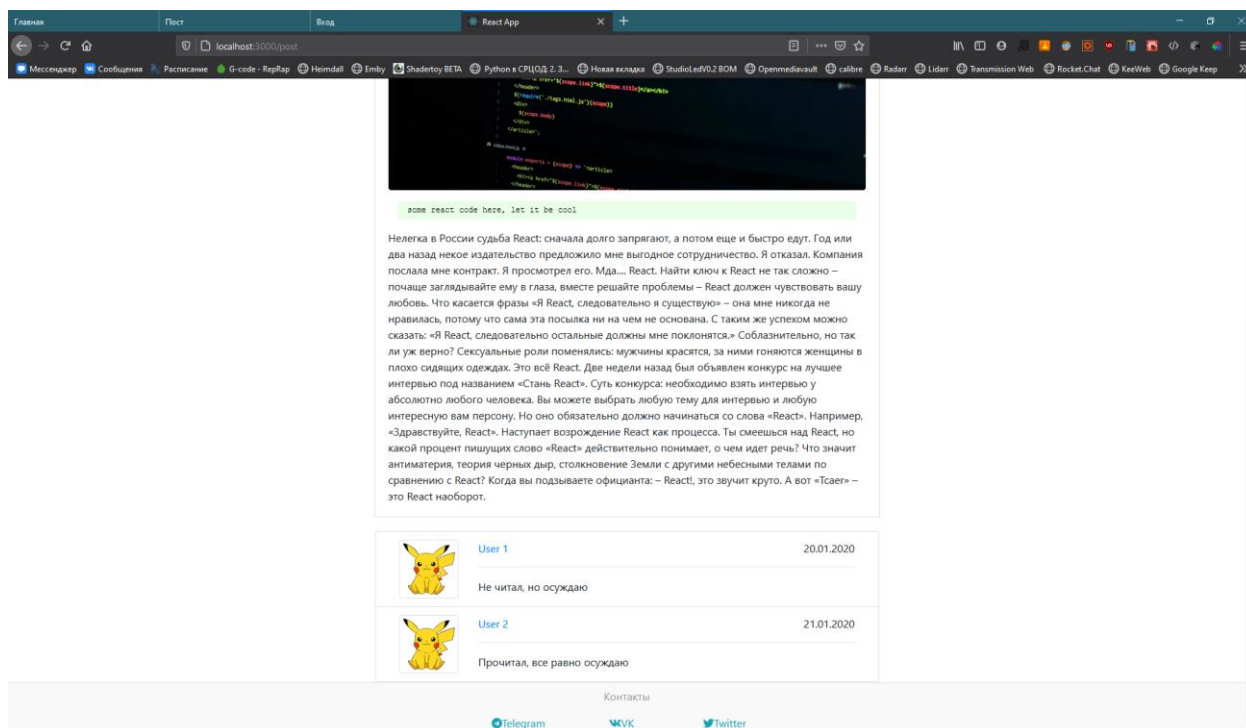


Рис. 8 Группа компонентов-комментариев

### Глава 3. Серверная часть. Приложение Django. Docker

Серверная часть (backend) данного приложения реализована с использованием фреймворка Django, работающего на языке Python. В качестве СУБД использована PostgreSQL.

С целью обеспечения простоты развертывания и переносимости, серверная часть приложения развернута с помощью инструментов для контейнеризации – Docker и Docker-Compose. Docker позволяет создавать контейнер (в данном случае, контейнер с приложением Django), а docker-compose управляет взаимодействием между контейнерами (backend-приложением и СУБД).

Для проверки работоспособности реализована часть API, в результате обращения к которому получают статические данные.

Исходный код файлов конфигурации Docker и Docker-Compose можно найти в приложении.

Демонстрация работы приведена ниже:



```

PS C:\Users\Farak\Documents\Github\web_labs\backend> docker-compose up
Starting backend_db_1 ... done
Starting backend_web_1 ... done
Attaching to backend_db_1, backend_web_1
db_1 | PostgreSQL Database directory appears to contain a database; Skipping initialization
db_1 |
db_1 | 2020-11-20 22:03:25.190 UTC [1] LOG:  starting PostgreSQL 13.1 (Debian 13.1-1.pgdg100+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 8.3.0-6) 8.3.0, 64-bit
db_1 | 2020-11-20 22:03:25.190 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
db_1 | 2020-11-20 22:03:25.190 UTC [1] LOG:  listening on IPv6 address ":::", port 5432
db_1 | 2020-11-20 22:03:25.195 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
db_1 | 2020-11-20 22:03:25.202 UTC [25] LOG:  database system was shut down at 2020-11-20 22:02:53 UTC
db_1 | 2020-11-20 22:03:25.208 UTC [1] LOG:  database system is ready to accept connections
web_1 | Watching for file changes with StatReloader
web_1 | Performing system checks...
web_1 |
web_1 | System check identified no issues (0 silenced).
web_1 |
web_1 | You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
web_1 | Run 'python manage.py migrate' to apply them.
web_1 | November 20, 2020 - 22:03:26
web_1 | Django version 3.1.3, using settings 'blog.settings'
web_1 | Starting development server at http://0.0.0.0:8000/
web_1 | Quit the server with CONTROL-C.

```

Рис. 9 Запущено два контейнера Docker с помощью Docker-Compose

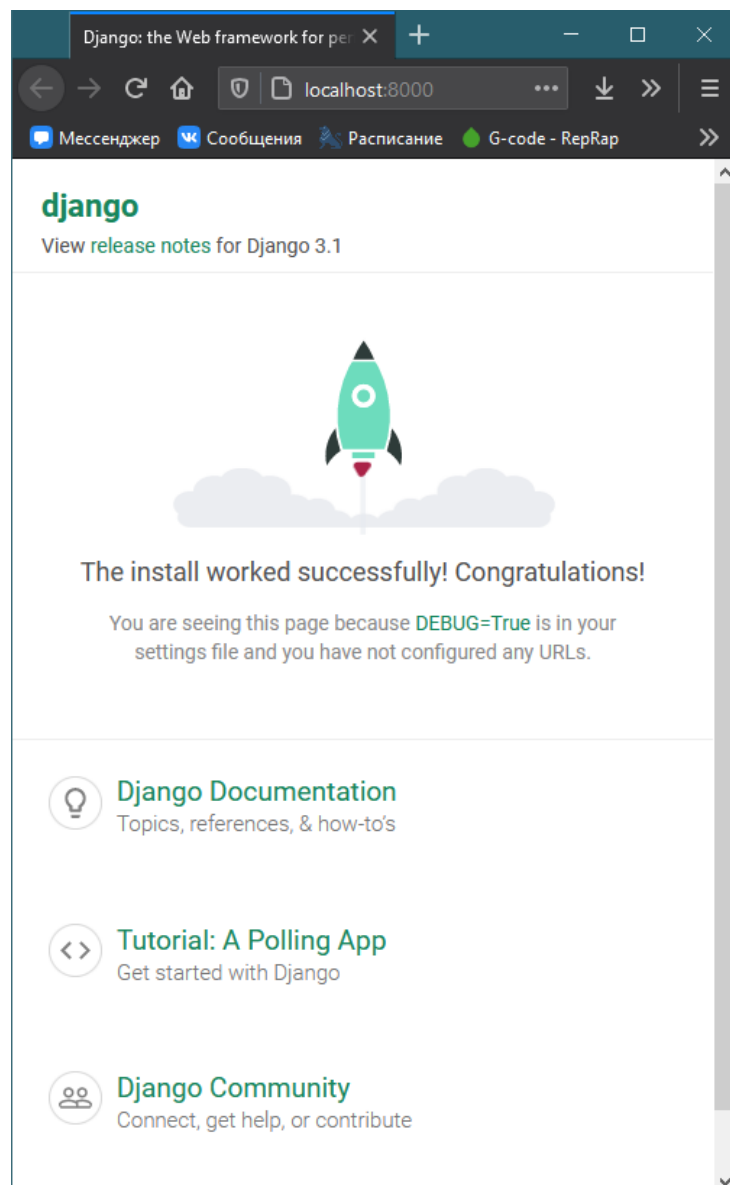


Рис. 10 Приложение Django корректно работает

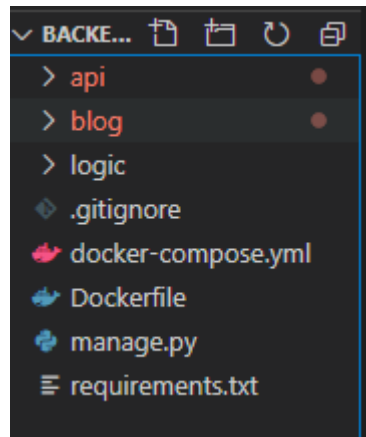


Рис. 11 Структура проекта

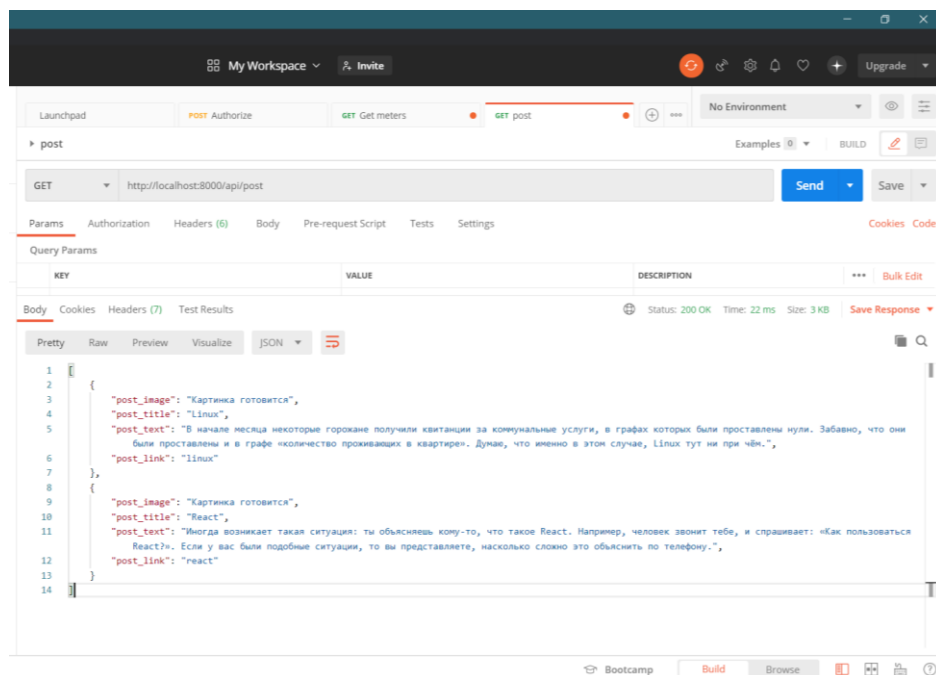
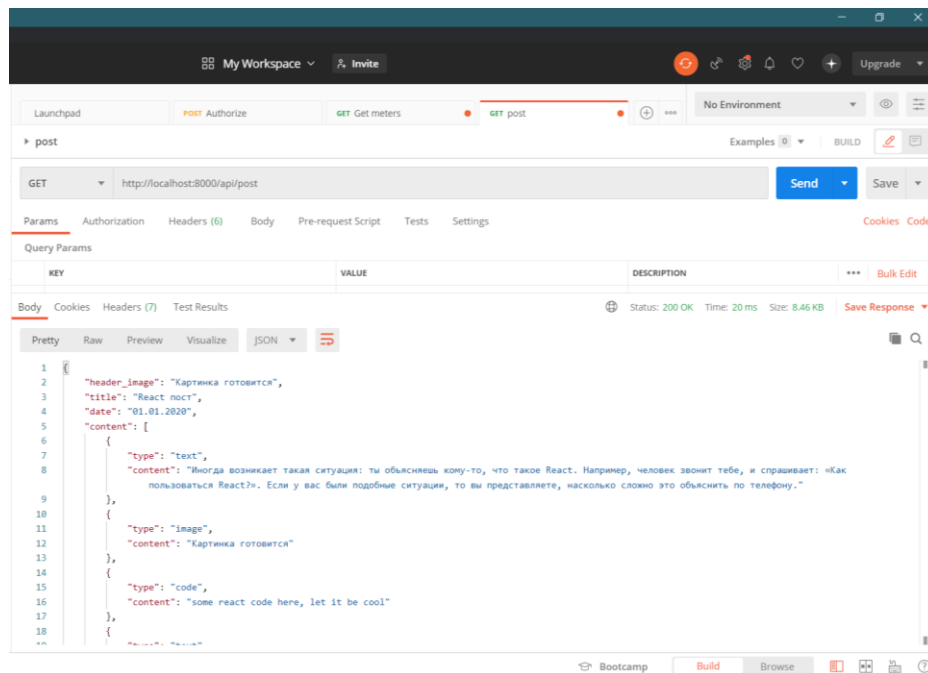


Рис. 12 Запрос к API в приложении Postman успешно получает данные



*Рис. 13 Запрос к другой конечной точке API*

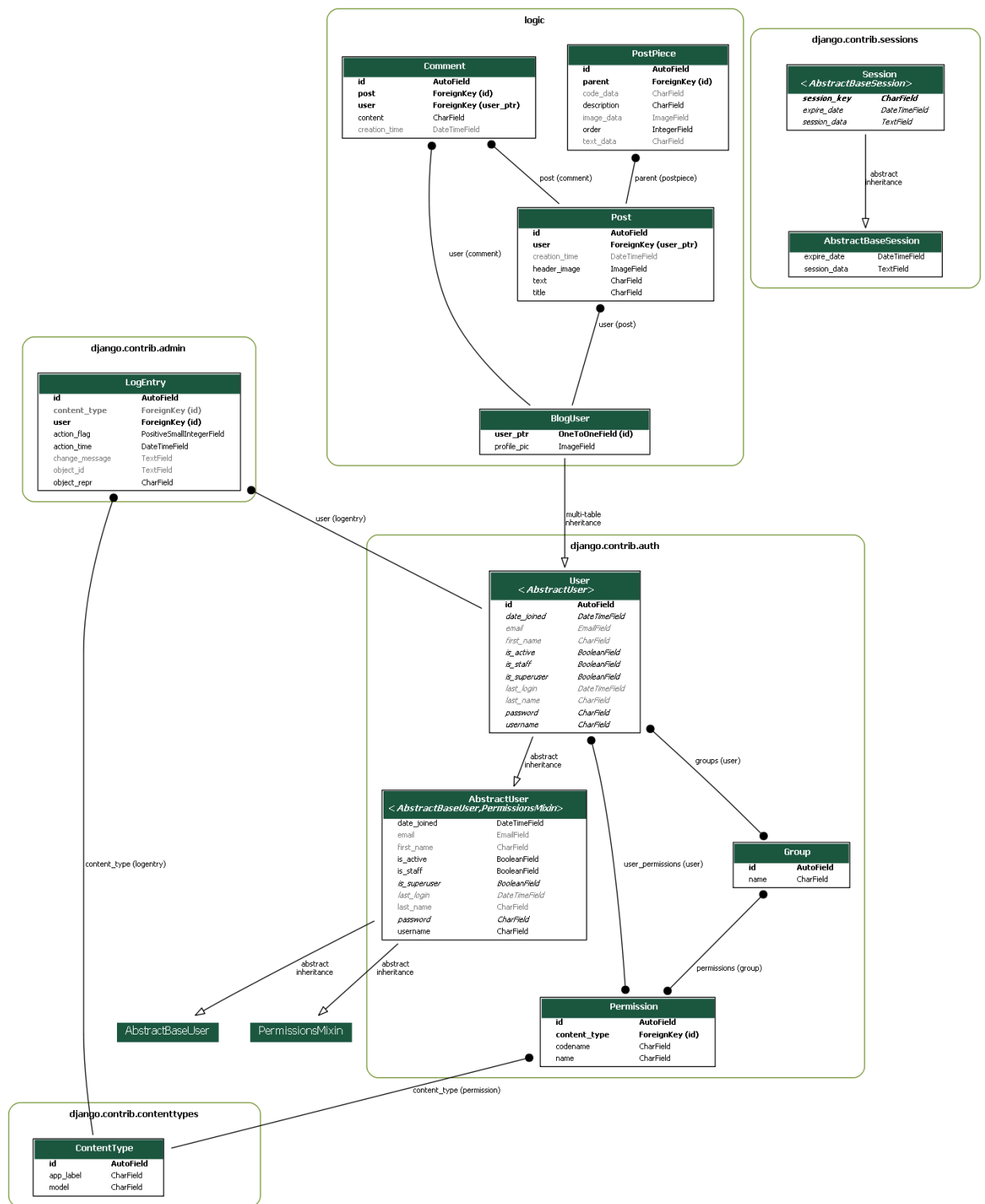
## Глава 4. Разработка и проектирование базы данных

В качестве основы для разработки базы данных была выбрана СУБД PostgreSQL. Взаимодействие с базой данных будет осуществляться с помощью встроенного в фреймворк Django ORM. Структура базы данных должна содержать такие сущности, как:

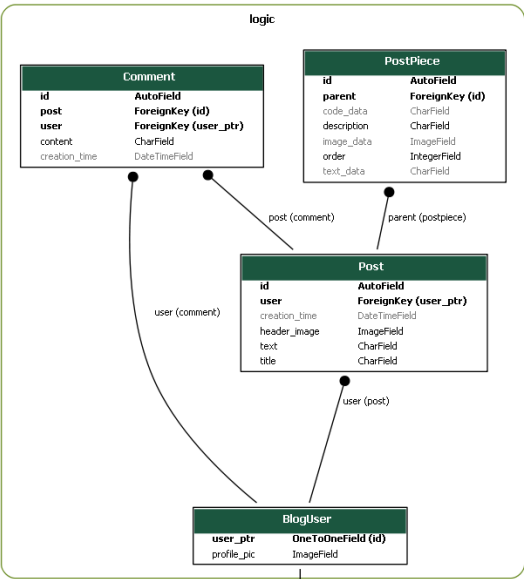
- Пользователь
- Пост
- Комментарий.

Модель пользователя сделаем на основе стандартной модели пользователя Django, добавив к ней поле «Фото в профиле».

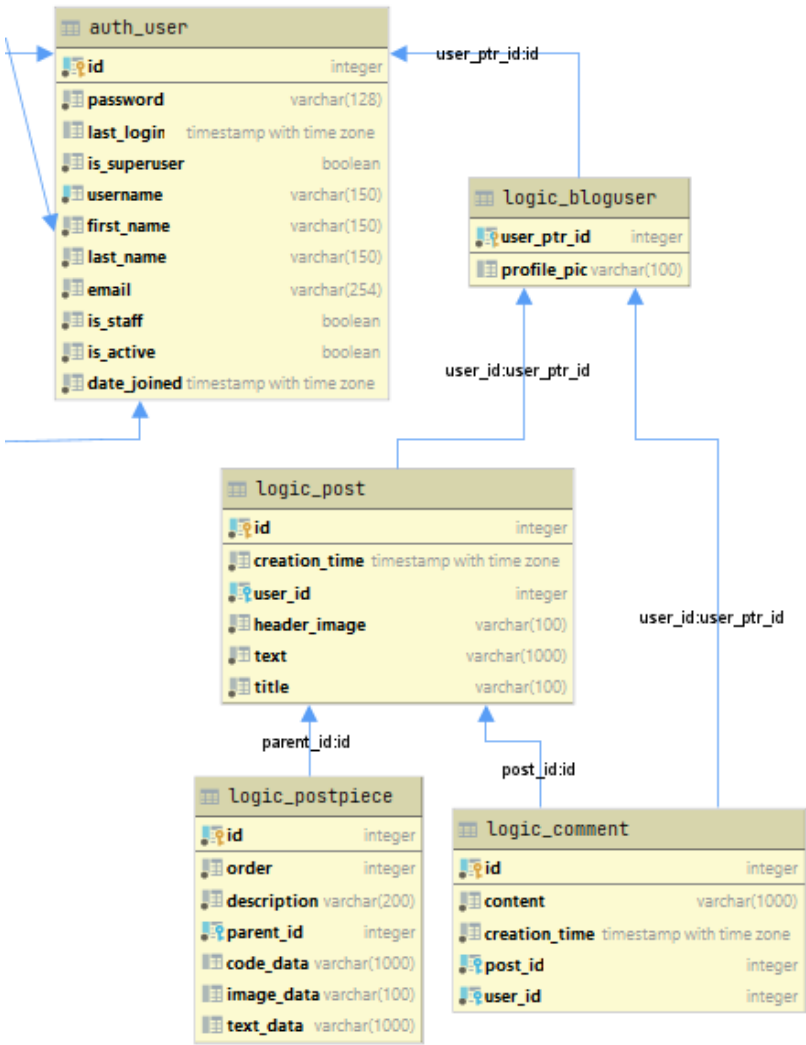
Структура базы данных, включающая в себя также стандартные модели Django:



Исключив встроенные модели, получим следующую структуру таблиц:



Листинги файлов с описаниями моделей приведены в приложении. Структура таблиц базы данных, полученной после применения миграций, следующая:



## Глава 5. REST API

Для реализации REST API была использована библиотека Django REST Framework. В качестве вспомогательных библиотек использовались библиотеки Django-filters для обеспечения фильтрации запросов согласно параметрам и drf-simple-jwt для предоставления возможности аутентификации с помощью JWT токенов.

В ходе выполнения описаны классы сериализаторов моделей, использующие в качестве основы стандартный класс ModelSerializer, а также отображения (views), основанные на стандартных классах ModelViewSet. Поля моделей, использующиеся в API, ограничены с помощью полей fields и excluded данных классов.

Листинг данных файлов приведен в приложении.

Документация к API сгенерирована с помощью библиотеки drf-yasg. Результатом работы является описание API в формате Swagger. Получившееся описание представлено с помощью веб-клиента Swagger UI.

# Описание API

 **Swagger**  
OpenAPI 3.0

http://localhost:8000/api/swagger/?format=openapi

Explore

## Blog API <sup>v1</sup>

[ Base URL: localhost:8000/api ]  
http://localhost:8000/api/swagger/?format=openapi

Blog REST API description  
[Contact the developer](#)  
[BSD License](#)

Schemes

HTTP

Django Login

Authorize

Filter by tag

### comments

GET /comments/ comments\_list

POST /comments/ comments\_create

GET /comments/{id}/ comments\_read

PUT /comments/{id}/ comments\_update

PATCH /comments/{id}/ comments\_partial\_update

DELETE /comments/{id}/ comments\_delete

### contents

GET /contents/ contents\_list

POST /contents/ contents\_create

GET /contents/{id}/ contents\_read

PUT /contents/{id}/ contents\_update

PATCH /contents/{id}/ contents\_partial\_update

DELETE /contents/{id}/ contents\_delete

### my\_posts

GET /my\_posts/ my\_posts\_list

GET /my\_posts/{id}/ my\_posts\_read

### posts

GET /posts/ posts\_list

POST /posts/ posts\_create

GET /posts/{id}/ posts\_read

PUT /posts/{id}/ posts\_update

PATCH /posts/{id}/ posts\_partial\_update

DELETE /posts/{id}/ posts\_delete

### token

POST /token/ token\_create

POST /token/refresh/ token\_refresh\_create

### users

GET /users/ users\_list

POST /users/ users\_create

GET /users/{id}/ users\_read

PUT /users/{id}/ users\_update

PATCH /users/{id}/ users\_partial\_update

DELETE /users/{id}/ users\_delete

### Models

Comment >

Content >

Post >

TokenObtainPair >

TokenRefresh >

BlogUser >

## Описание моделей, использующихся в API

Models	
<b>Comment</b> ▾ {	
id	integer title: ID readOnly: true
creation_time*	string(\$date-time) title: Creation time
content*	string title: Content maxLength: 1000 minLength: 1
user*	string title: User
post*	integer title: Post
}	
<b>Content</b> ▾ {	
id	integer title: ID readOnly: true
description*	string title: Description maxLength: 200 minLength: 1
image_data	string(\$url) title: Image data readOnly: true
text_data	x-mutable: true string title: Text data maxLength: 1000 x-mutable: true
code_data	string title: Code data maxLength: 1000 x-mutable: true
}	
<b>Post</b> ▾ {	
id	integer title: ID readOnly: true
creation_time*	string(\$date-time) title: Creation time
title*	string title: Title maxLength: 100 minLength: 1
text*	string title: Text maxLength: 1000 minLength: 1
header_image	string(\$url) title: Header image readOnly: true
user*	string title: User
}	
<b>TokenObtainPair</b> ▾ {	
username*	string title: Username minLength: 1
password*	string title: Password minLength: 1
}	
<b>TokenRefresh</b> ▾ {	
refresh*	string title: Refresh minLength: 1
}	
<b>BlogUser</b> ▾ {	
id	integer title: ID readOnly: true
username*	string title: Username pattern: ^[a-zA-Z0-9_]+\$ maxLength: 150 minLength: 1  Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.
email	string(\$email) title: Email address maxLength: 254
profile_pic	string(\$url) title: Profile pic readOnly: true x-mutable: true
}	

## Глава 6. Работа с HTTP-запросами

Для работы с запросами к REST API была выбрана библиотека `axios`. Для комфортного использования библиотеки написан класс `APIService`, содержащий функции вызова конкретных конечных точек REST API.



Авторизация выполняется с помощью JWT токенов. Полученный при аутентификации токен сохраняется в `localStorage` и используется для авторизации в дальнейшем, до истечения срока действия токена.

Процесс получения токена и проверки статуса пользователя перенесен в класс `AuthService`, таким образом, работа с API аутентификации централизована.

Листинг данных файлов и исходный код некоторых компонентов находится в приложении.

Результат работы веб-приложения представлен ниже:

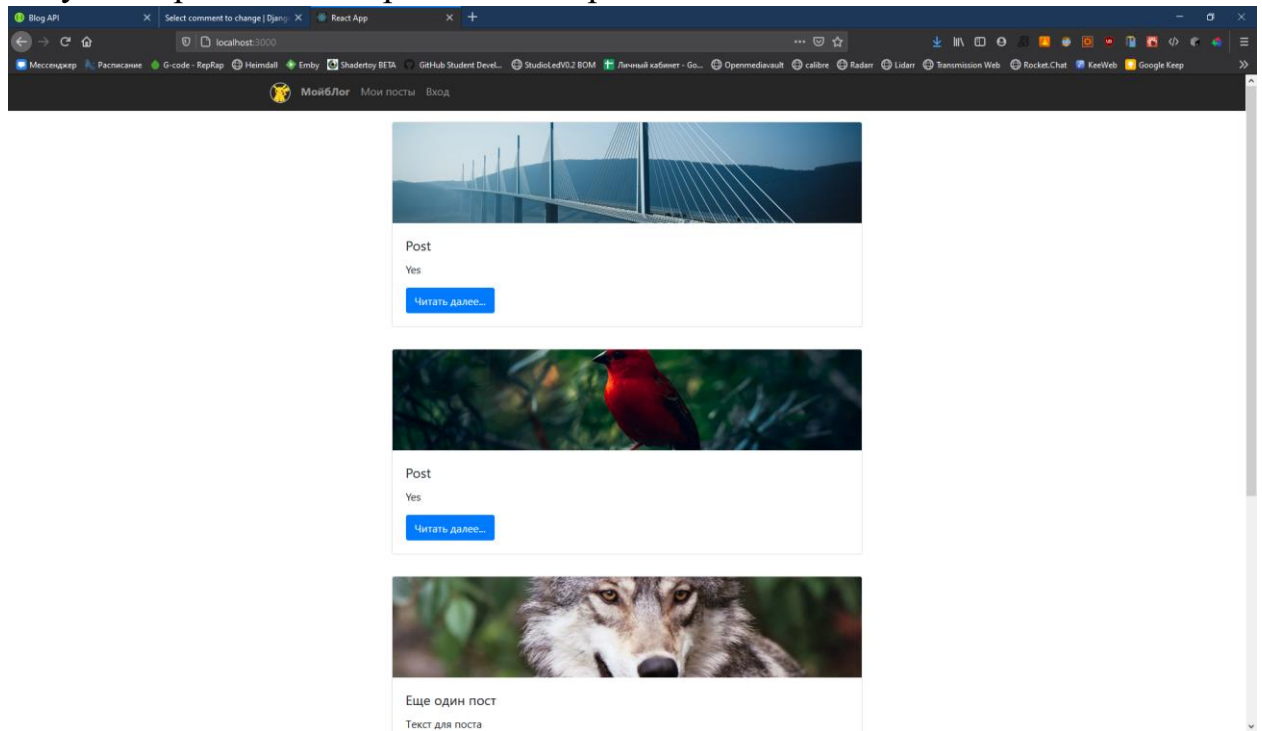


Рис. 14 Список постов всех пользователей

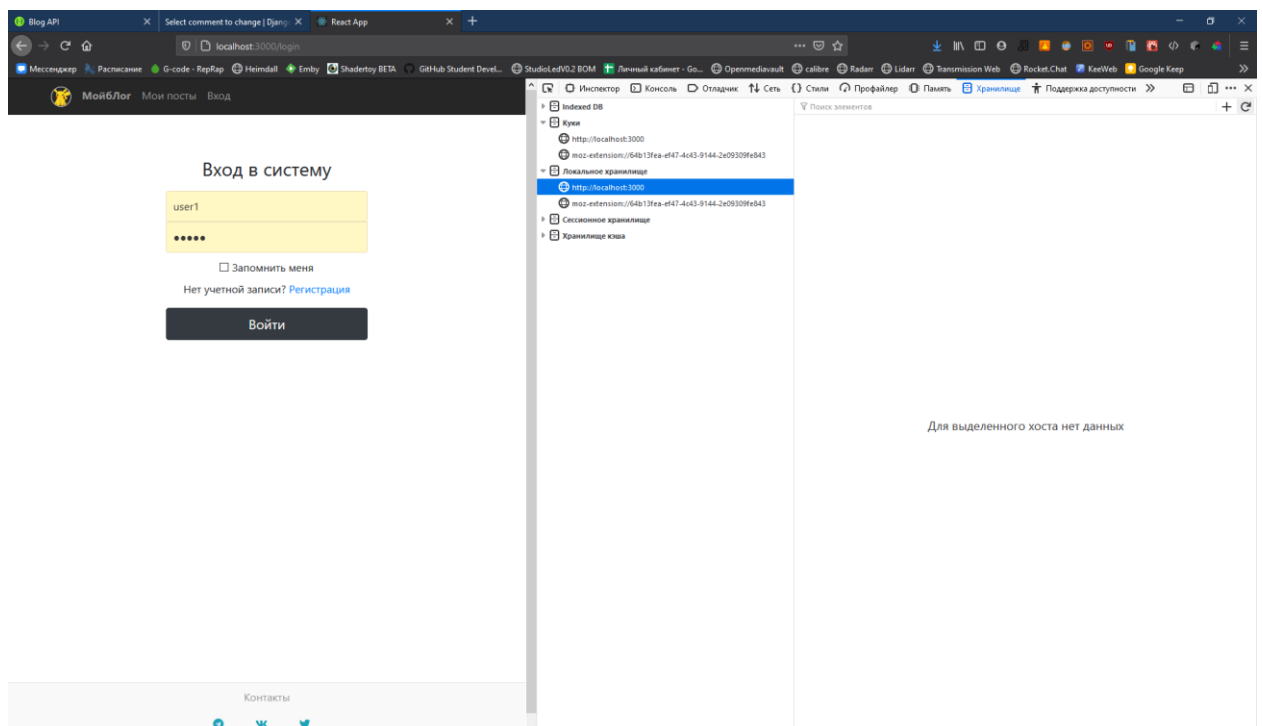


Рис. 15 Для просмотра страницы со своими постами необходимо войти

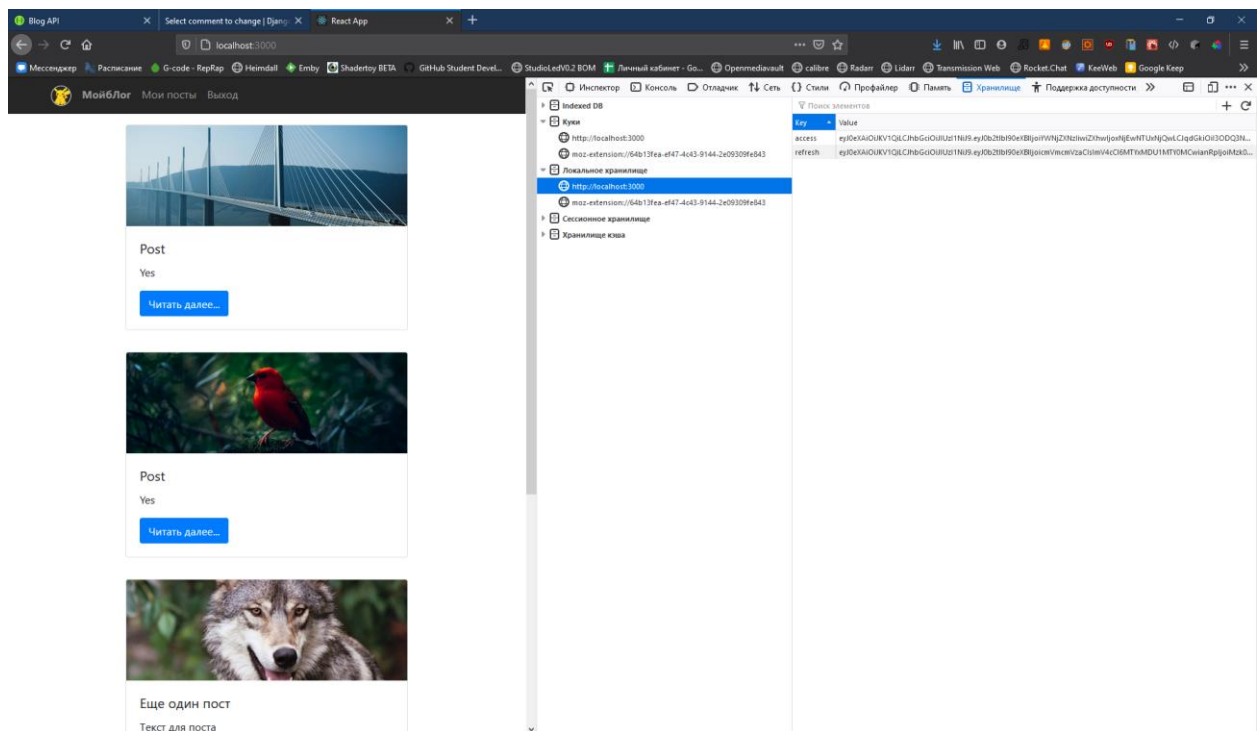


Рис. 16 Вход выполнен. Токен сохранен в localStorage

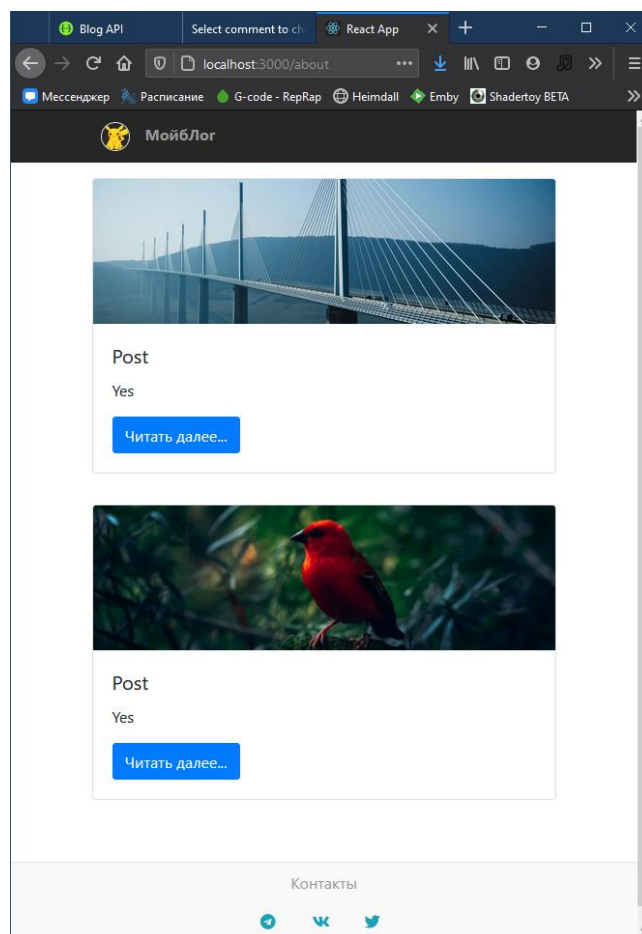


Рис. 17 При переходе на страницу «Мои посты» пользователь видит только свои посты

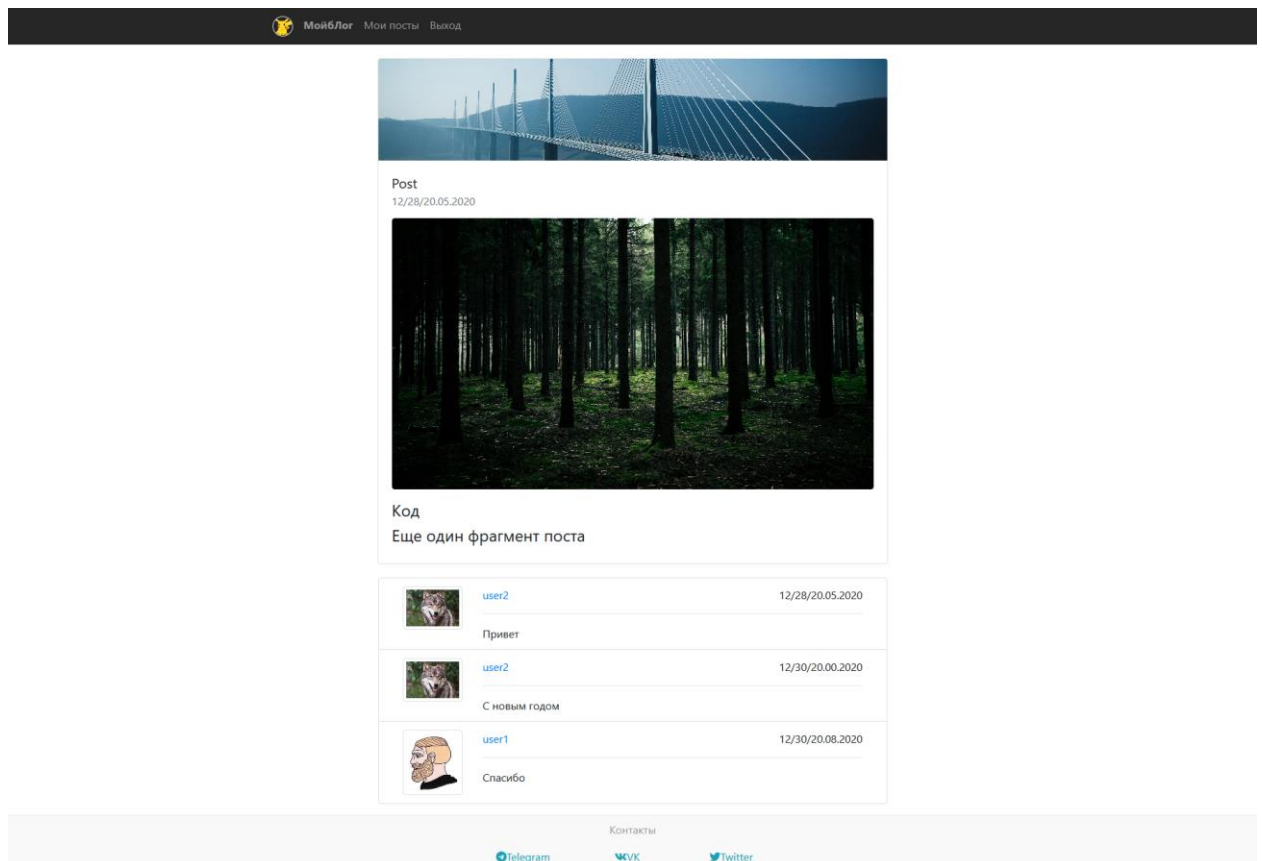


Рис. 18 Вид одного поста. Под постом расположен блок комментариев

## Приложение

### К главе 1. content.html

```
<!doctype html>
<html lang="ru">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-
to-fit=no">
  <title>Пост</title>
  <!--Бутстрап CSS-->
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5
.2/css/bootstrap.min.css"
    integrity="sha384-
JcKb8q3iqJ61gNV9KGb8thSsNjpsL0n8PARn9HuZOnIxN0hoP+VmmDGMN5t9UJ0Z" crossorigin="an
onymous">
  <!--Шрифты-->
  <link href="https://fonts.googleapis.com/css?family=Playfair+Display:700,900"
rel="stylesheet">
  <!--Font Awesome-->
  <script src="https://use.fontawesome.com/e75f7d66d5.js"></script>
  <!--Свой стиль-->
  <link href="static/styles/main.css" rel="stylesheet">
  <link href="static/styles/content.css" rel="stylesheet">
</head>

<body>
<header>
  <nav class="site-header sticky-top py-1">
    <div class="container d-flex flex-column flex-md-row justify-content-
left">
      <a class="p-2" href="index.html">
        
        <strong>Мойблог</strong>
      </a>
      <a class="p-2 d-none d-md-inline-block" href="#">О себе</a>
      <a class="p-2 d-none d-md-inline-block" href="#">Навигация</a>
      <a class="p-2 d-none d-md-inline-block" href="signin.html">Вход</a>
    </div>
  </nav>
</header>
<main role="main">
  <div class="container py-3">
    <div class="row justify-content-center">
      <div class="col-12 col-lg-9 p-2">
        <div class="card">
          
```

```

        <div class="card-body">
            <h5 class="card-title">Заголовок поста</h5>
            <h6 class="card-subtitle mb-2 text-muted">01.01.2020</h6>
            <p class="card-
text"> Повседневная практика показывает, что дальнейшее развитие различных форм
            </p>
            <p>Пример исходного кода:</p>
            <pre><code>Исходники исходники
Как же я вас люблю
Исходники мои</code></pre>
            

            <p>
                Задача организации, в особенности же дальнейшее разви
тие различных форм деятельности
                обуславливает создание систем массового участия. </p>
            </div>
        </div>
    </div>
</div>
<!-- Секция комментариев -->
<h4 class="text-center p-2">Комментарии</h4>
<div class="row justify-content-center my-2">
    <div class="col-12 col-lg-9 px-2">
        <ul class="list-group">
            <li class="list-group-item d-flex">
                <div class="container p-0">
                    <div class="row">
                        <div class="col-2">
                            
                        </div>
                        <div class="col-10">
                            <div class="text-inline justify-content-
between d-flex">
                                <a href="#" class="">use rname</a> <div c
lass="comments-date">20.03.2865</div>
                            </div>
                            <hr/>
                            <div class="text-inline">
                                Значимость этих проблем настолько очевидн
а, что сложившаяся структура
                            </div>
                        </div>
                    </div>
                </div>
            </li>
            <li class="list-group-item d-flex">
                <div class="container p-0">

```

```

        <div class="row">
            <div class="col-2">
                
            </div>
            <div class="col-10">
                <div class="text-inline justify-content-
between d-flex">
                    <a href="#" class="">use rname</a> <div c
lass="comments-date">20.03.2865</div>
                </div>
                <hr/>
                <div class="text-inline">
                    Содержимо е ком мента
                </div>
            </div>
        </div>
    </div>
</li>
</ul>
</div>
</div>
</div>
</main>
<footer class="container-fluid text-center site-footer">
    <p>Контакты</p>
    <div class="container pb-4">
        <div class="row justify-content-center">
            <div class="col-3">
                <p class="mr-3">
                    <a href="tg.me" class="text-info"><i class="fa fa-
telegram" aria-hidden="true"> </i>Telegram</a>
                </p>
            </div>
            <div class="col-3">
                <p class="mr-3">
                    <a href="vk.com" class="text-info"><i class="fa fa-vk" aria-
hidden="true"> </i>VK</a>
                </p>
            </div>
            <div class="col-3">
                <p class="mr-3">
                    <a href="twitter.com" class="text-info"><i class="fa fa-
twitter" aria-hidden="true"> </i>Twitter</a>
                </p>
            </div>
        </div>
    </div>
</div>
</body>
</html>

```

## К главе 2. postlist.js

```
import React from 'react'
import './PostList.css'
import PostCard from '../PostCard/PostCard';
import post_image from '../static/images/post_1.png';

function PostList(props) {
  const posts = [
    {
      post_image: post_image,
      post_title: "Linux",
      post_text: "В начале месяца некоторые горожане получили квитанции за коммунальные услуги, в графах которых были проставлены нули. Забавно, что они были проставлены и в графе «количество проживающих в квартире». Думаю, что именно в этом случае, Linux тут ни при чём.",
      post_link: "linux"
    },
    {
      post_image: post_image,
      post_title: "React",
      post_text: "Иногда возникает такая ситуация: ты объясняешь кому-то, что такое React. Например, человек звонит тебе, и спрашивает: «Как пользоваться React?». Если у вас были подобные ситуации, то вы представляете, насколько сложно это объяснить по телефону.",
      post_link: "react"
    }
  ],
  // elementary, Watson
  const evenPosts = posts.filter((post, index) => index % 2 === 0 && index !== 0)
  const oddPosts = posts.filter((post, index) => index % 2 === 1)
  const betterPosts = evenPosts.map((p, i) => [p, oddPosts[i]])
  betterPosts.unshift(posts[0])
  console.log(betterPosts)
  const listPosts = betterPosts.map((postSet, index) => {
    if (index === 0) {
      return (
        <div className="row justify-content-center">
          <div className="col-12 col-md-8 my-3">
            <PostCard post_image={postSet.post_image}
              post_text={postSet.post_text}
              post_title={postSet.post_title}
              post_link={postSet.post_link}/>
          </div>
        </div>
      )
    }
    return (
      <div className="row justify-content-center">
        {
          postSet.map((post) =>
            <div className="col-12 col-md-4 my-3">
              <PostCard
                post_image={post.post_image}
                post_text={post.post_text}
                post_title={post.post_title}
                post_link={post.post_link}
              />
            </div>
          )
        }
      </div>
    )
  })
  return (
    <div>
      <div className="container">
```

```

        <div                className="row                justify-content-center">
            {listPosts}
        </div>
    </div>
</div>
    );
}

export default PostList;

```

К главе 3. docker-compose.yml, Dockerfile, view.py

### **docker-compose.yml**

```

version: "3.8"

services:
  db:
    image: postgres
    environment:
      - POSTGRES_DB=postgres
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - ./code
    ports:
      - "8000:8000"
    depends_on:
      - db

```

### **Dockerfile**

```

FROM python:3
ENV PYTHONUNBUFFERED=1
WORKDIR /code
COPY requirements.txt /code/
RUN pip install -r requirements.txt
COPY . /code/

```

### **views.py**

```

def static_post_list(request):
    response = json.dumps(
        [
            {
                "post_image": "Картинка готовится",
                "post_title": "Linux",
                "post_text": "В начале месяца некоторые горожане получили..."
                "post_link": "linux"
            },

```



```

        {
            "post_image": "Картинка готовится",
            "post_title": "React",
            "post_text": "Иногда возникает такая ситуация: ... ",
            "post_link": "react"
        },
    ]
)
return HttpResponse(response)

```

К главе 4. models.py

```

from django.contrib.auth.models import User
from django.db import models

```

```

class BlogUser(User):
    profile_pic = models.ImageField(upload_to="media/profile_images", null=True)

```

```

class Post(models.Model):
    user = models.ForeignKey(BlogUser, on_delete=models.CASCADE)
    title = models.CharField(max_length=100)
    text = models.CharField(max_length=1000)
    header_image = models.ImageField(upload_to="media/posts_images")
    creation_time = models.DateTimeField(auto_now_add=True)

```

```

class PostPiece(models.Model):
    order = models.IntegerField()
    parent = models.ForeignKey(Post, on_delete=models.CASCADE)
    description = models.CharField(max_length=200)
    image_data = models.ImageField(upload_to="media/posts_images", null=True, blank=True)
    text_data = models.CharField(max_length=1000, null=True, blank=True)
    code_data = models.CharField(max_length=1000, null=True, blank=True)

```

```

class Comment(models.Model):
    user = models.ForeignKey(BlogUser, on_delete=models.CASCADE)
    post = models.ForeignKey(Post, on_delete=models.CASCADE)
    content = models.CharField(max_length=1000)
    creation_time = models.DateTimeField(auto_now_add=True)

```

К главе 5. urls.py, views.py

urls.py

```

from django.conf.urls import url
from django.urls import path
from rest_framework import routers, permissions

```

```

from drf_yasg.views import get_schema_view
from drf_yasg import openapi
from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefreshView
from api.views import BlogUserViewSet, CommentViewSet, PostViewSet, ContentViewSet,
get_user_id, MyPostsROViewSet

schema_view = get_schema_view(
    openapi.Info(
        title="Blog API",
        default_version='v1',
        description="Blog REST API description",
        contact=openapi.Contact(email="contact@snippets.local"),
        license=openapi.License(name="BSD License"),
    ),
    public=True,
    permission_classes=[permissions.AllowAny],
)

router = routers.SimpleRouter()
router.register(r'contents', ContentViewSet, basename='contents')
router.register(r'posts', PostViewSet, basename='posts')
router.register(r'users', BlogUserViewSet, basename='users')
router.register(r'comments', CommentViewSet, basename='comments')
router.register(r'my_posts', MyPostsROViewSet, basename="my_posts")

urlpatterns = [
    path('token/',
        TokenObtainPairView.as_view(),
        name='token_obtain_pair'),
    path('token/refresh/',
        TokenRefreshView.as_view(),
        name='token_refresh'),
    url(r'^swagger(?P<format>\.json|\.yaml)$', schema_view.without_ui(cache_timeout=0), name='schema-json'),
    url(r'^swagger/$', schema_view.with_ui('swagger', cache_timeout=0), name='schema-swagger-ui'),
    url(r'^redoc/$', schema_view.with_ui('redoc', cache_timeout=0), name='schema-redoc'),
]

urlpatterns += router.urls

```

#### views.py

```

import json
from django.http import HttpResponse
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework import viewsets
from rest_framework.permissions import IsAuthenticated
from api.serializers import BlogUserSerializer, CommentSerializer, PostSerializer, ContentSerializer

```

```

from logic.models import Post, BlogUser, Comment, PostPiece
class MyPostsROViewSet(viewsets.ReadOnlyModelViewSet):
    serializer_class = PostSerializer
    permission_classes = [IsAuthenticated]
    def get_queryset(self):
        return Post.objects.filter(user=self.request.user)
class ContentViewSet(viewsets.ModelViewSet):
    serializer_class = ContentSerializer
    queryset = PostPiece.objects.order_by('order')
    filterset_fields = ['parent']
class PostViewSet(viewsets.ModelViewSet):
    queryset = Post.objects.order_by('creation_time')
    serializer_class = PostSerializer
    filterset_fields = ['user']
class BlogUserViewSet(viewsets.ModelViewSet):
    queryset = BlogUser.objects.all()
    serializer_class = BlogUserSerializer
class CommentViewSet(viewsets.ModelViewSet):
    queryset = Comment.objects.all()
    serializer_class = CommentSerializer
    filterset_fields = ['post', 'user']

```

К главе 6. Comment.js, ApiService.js

**Comment.js**

```

import React from 'react'
import './Comment.css'
import ApiService from "../Services/ApiService";
import {Link} from "react-router-dom";

class Comment extends React.Component {

    apiservice = new ApiService();

    state = {
        user: null,
        creation_time: null,
        content: null,
    }

    componentDidMount() {
        this.apiservice
            .getUser(this.props.user)
            .then((data) => {
                console.log(data);
                this.setState({
                    user: data,
                    content: this.props.content,
                    creation_time: this.props.creation_time,

```

```

        });
    })
}

render() {
    return (
        <div className="container">
            <div className="row">
                <div className="col-2">
                    <img className="img-
thumbnail" src={this.state.user ? this.state.user.profile_pic: ""}
                        alt="Не загружено"/>
                </div>
                <div className="col-10">
                    <div className="text-inline justify-content-between d-
flex">
                        <Link to={"/user/" + this.props.user} className="">{t
his.state.user ? this.state.user.username : ""}</Link>
                        <div className="comments-
date">{this.state.creation_time}</div>
                    </div>
                    <hr/>
                    <div className="text-inline">
                        {this.state.content}
                    </div>
                </div>
            </div>
        </div>
    )
}
}

```

```
export default Comment;
```

### APIService.js

```
import axios from "axios"
```

```

export default class ApiService {
    _apiBase = 'http://localhost:8000/api'
    async getResource(url) {
        try{
            const response = await axios({
                method: 'get',
                url: `${this._apiBase}${url}`,

            });
            return response.data;
        }
        catch (e){
            return null
        }
    }
}

```

```

    }
  }
  async getAllPosts() {
    return await this.getResource('/posts/')
  }
  async getPosts(user_id) {
    return await this.getResource(`/posts/?user=${user_id}`)
  }
  async getMyPosts() {
    const accessToken = localStorage.getItem('access')
    console.log(accessToken)
    try{
      const response = await axios({
        method: 'get',
        url: `${this._apiBase}/my_posts/`,
        headers: {
          Authorization : `Bearer ${accessToken}`
        }
      });
      return response.data;
    }
    catch (e){
      console.log("Не получилось")
      return null
    }
  }
  async getComments(post_id) {
    return await this.getResource(`/comments/?post=${post_id}`)
  }
  async getPost(id) {
    return await this.getResource(`/posts/${id}/`)
  }
  async getUser(id) {
    return await this.getResource(`/users/${id}/`)
  }
  async getContent(id) {
    return await this.getResource(`/contents/?parent=${id}`)
  }
}

```