**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

[Введение 3](#_Toc60250352)

[Цель работы 3](#_Toc60250353)

[Выполнение 3](#_Toc60250354)

[Глава 1. Верстка HTML 3](#_Toc60250355)

[Глава 2. Приложение React 6](#_Toc60250356)

[Глава 3. Серверная часть. Приложение Django. Docker 8](#_Toc60250357)

[Глава 4. Разработка и проектирование базы данных 11](#_Toc60250358)

[Глава 5. REST API 14](#_Toc60250359)

[Глава 6. Работа с HTTP-запросами 16](#_Toc60250360)

[Приложение 20](#_Toc60250361)

# 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 раработки 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:

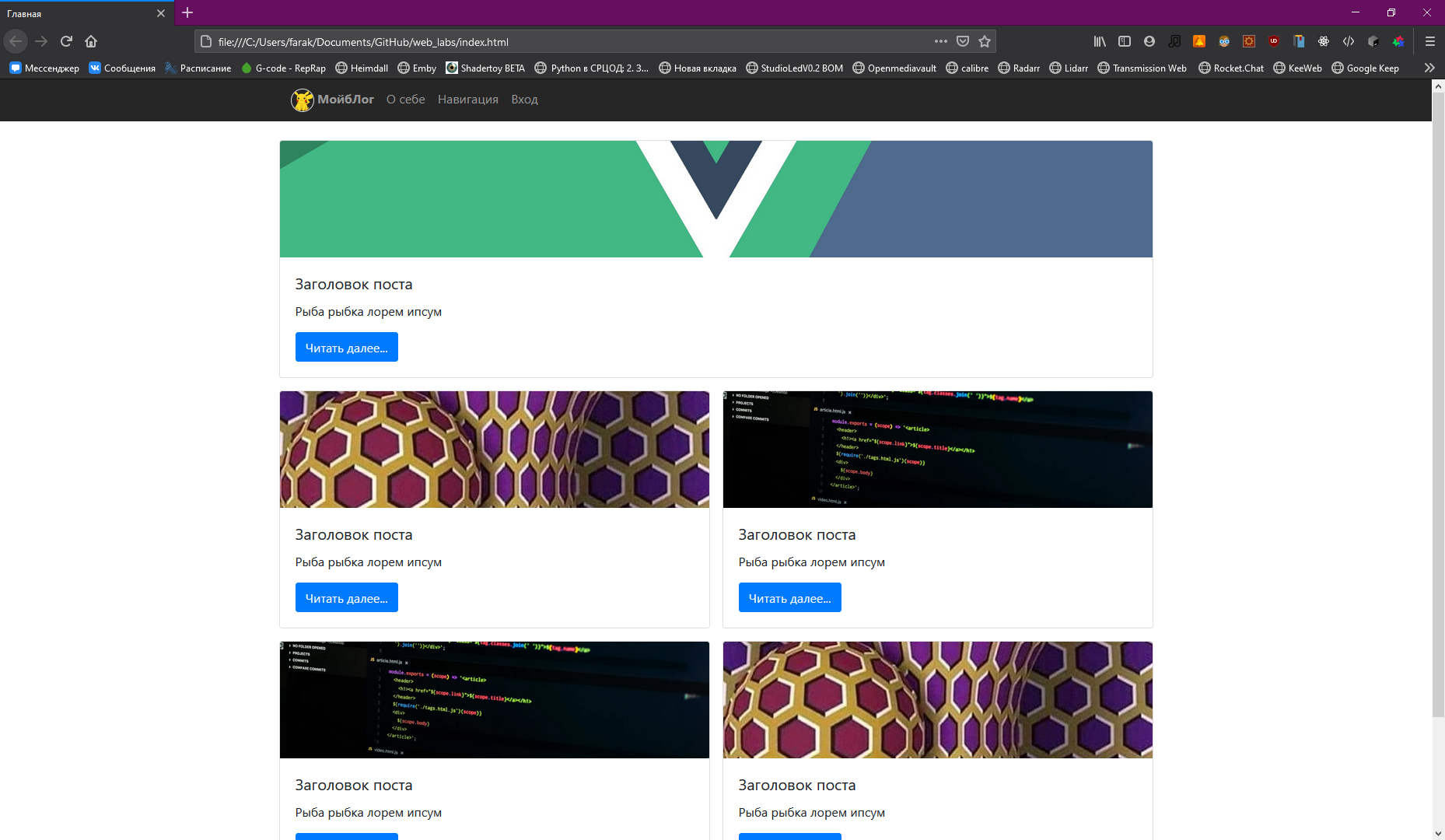


Рис. 1 Главная страница

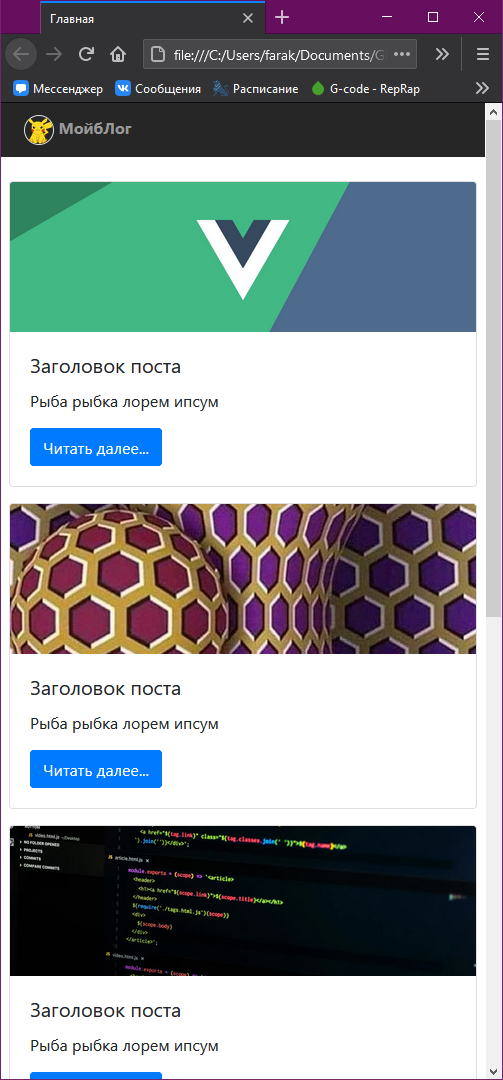


Рис. 2 Главная страница на узком экране

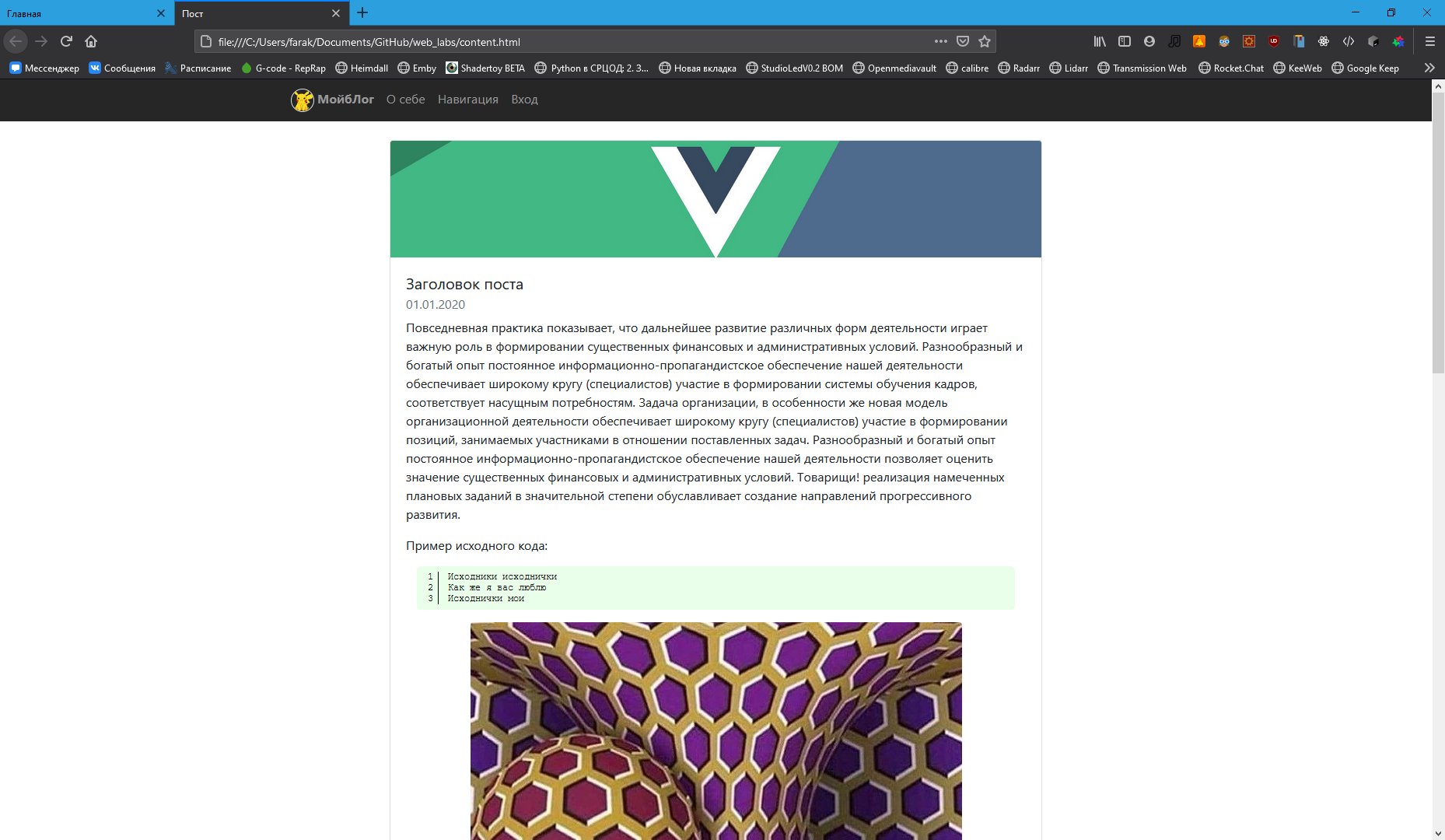


Рис. 3 Страница с контентом

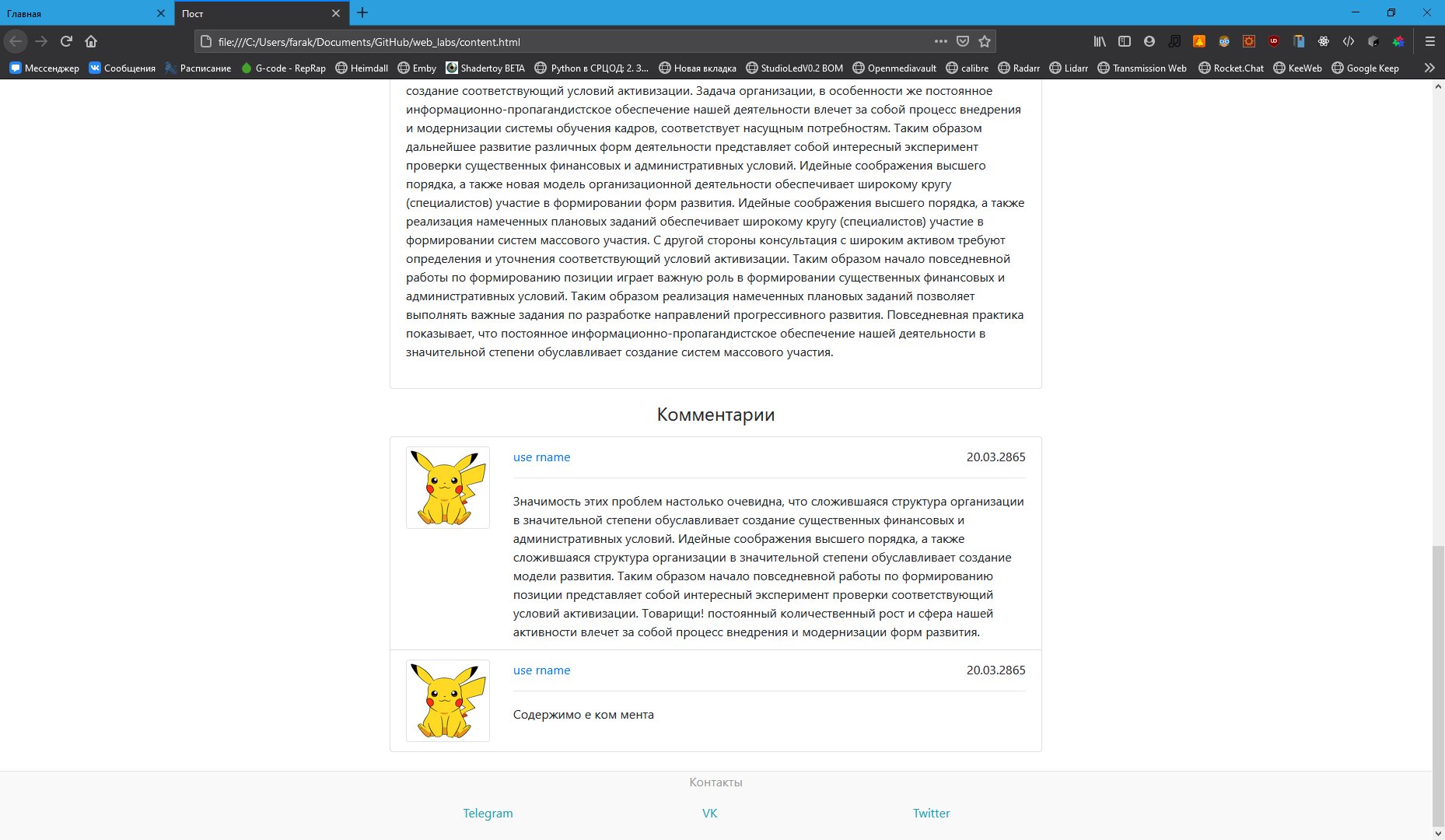


Рис. 4 Комментарии на странице с контентом

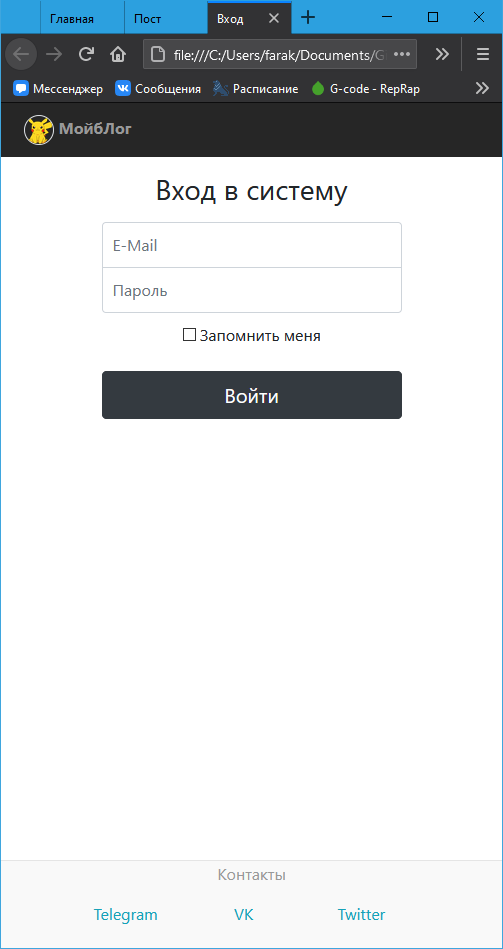


Рис. 5 Форма входа

# 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:

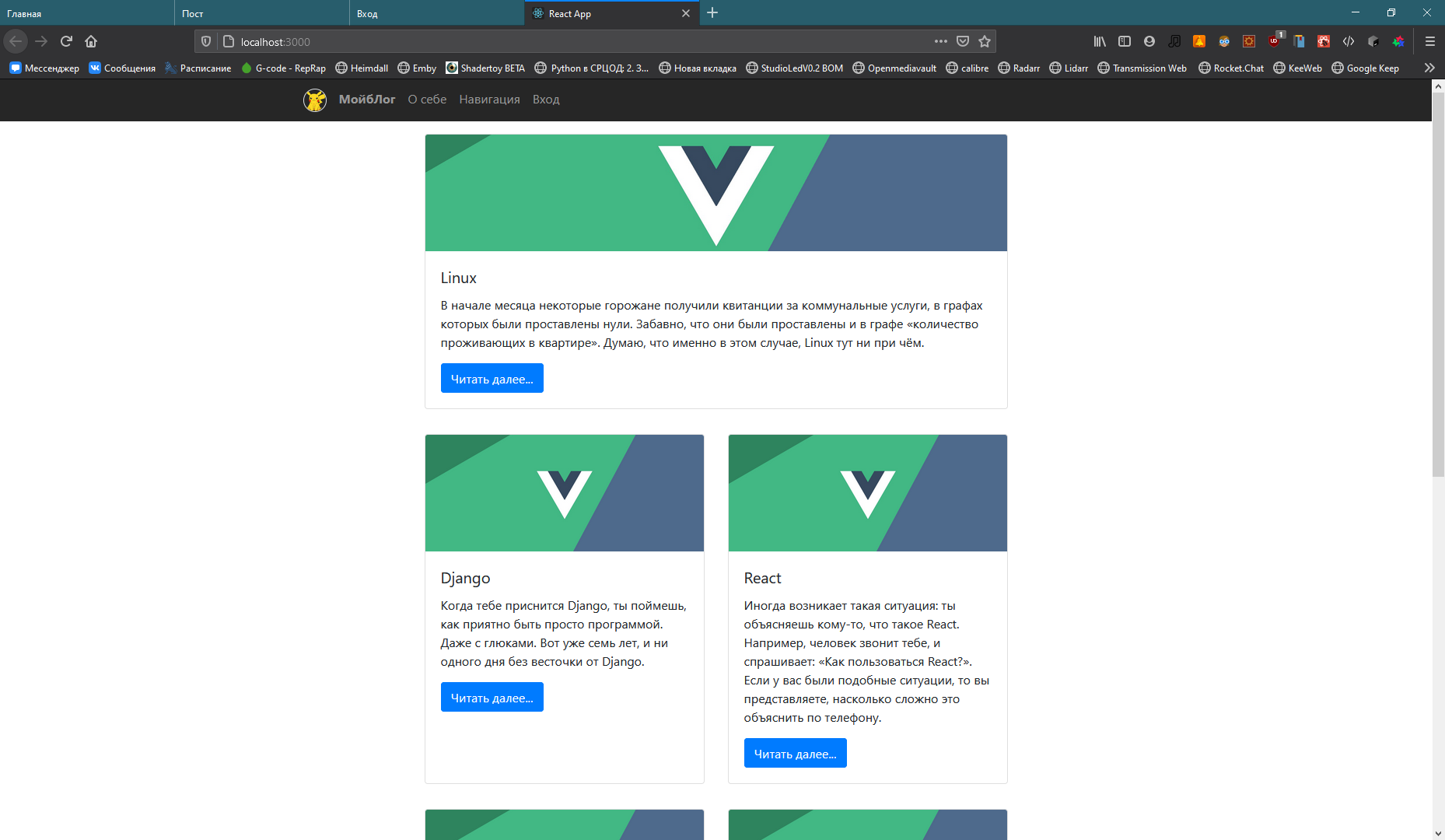


Рис. 6 Карточки постов на главном экране

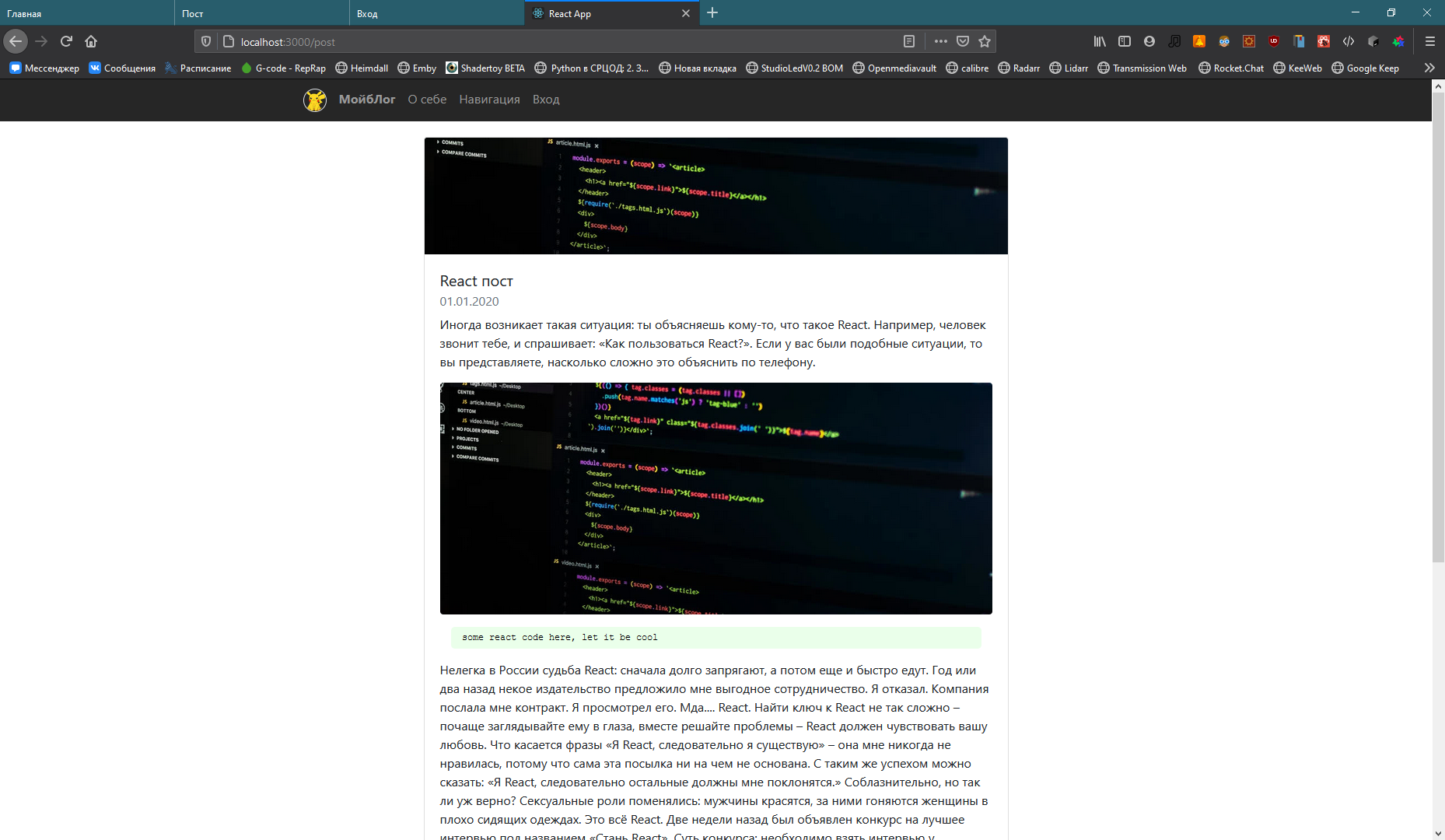


Рис. 7 Компонент с содержимым поста

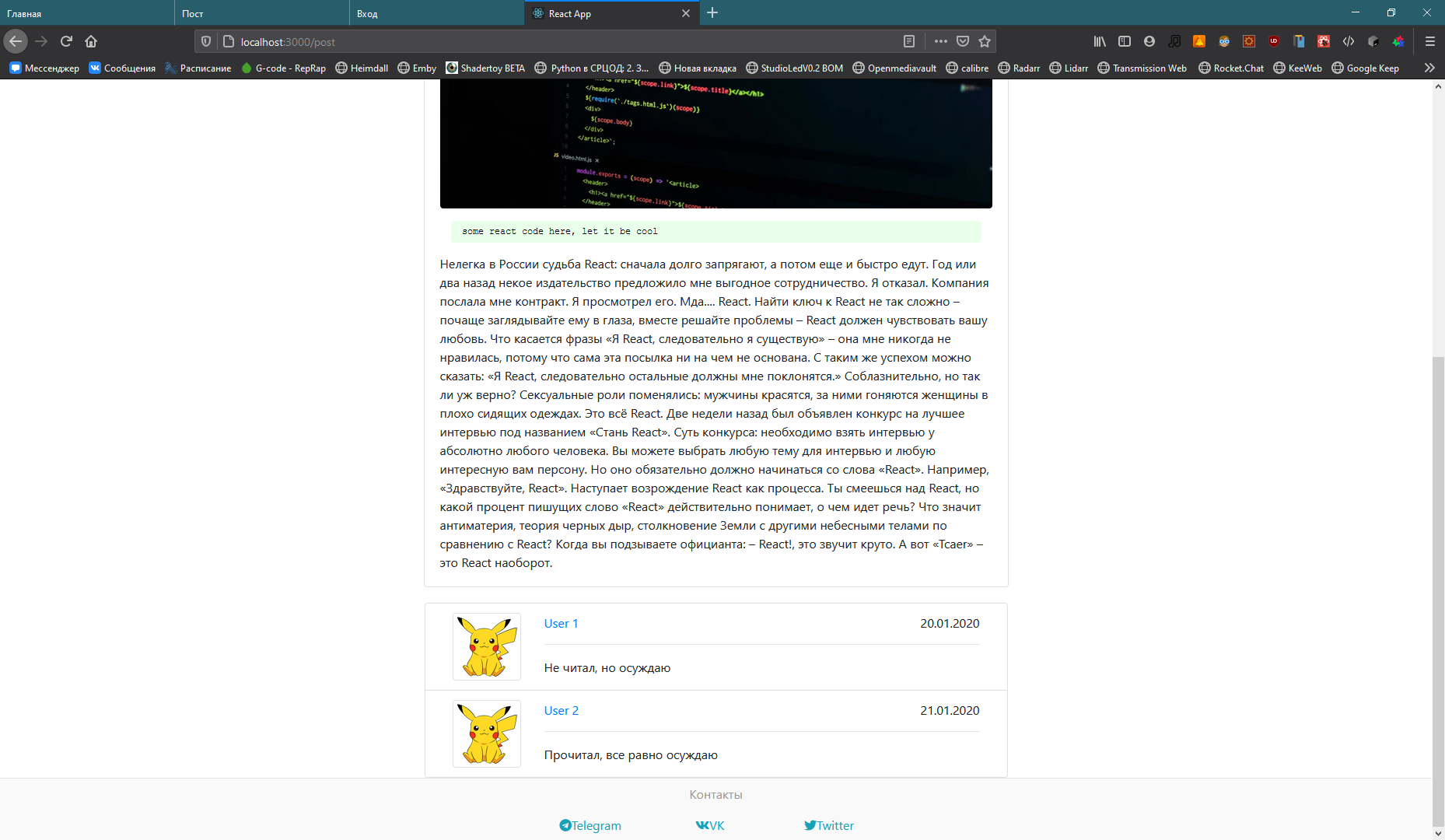


Рис. 8 Группа компонентов-комментариев

# 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 получаются статические данные.

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

Демонстрация работы приведена ниже:

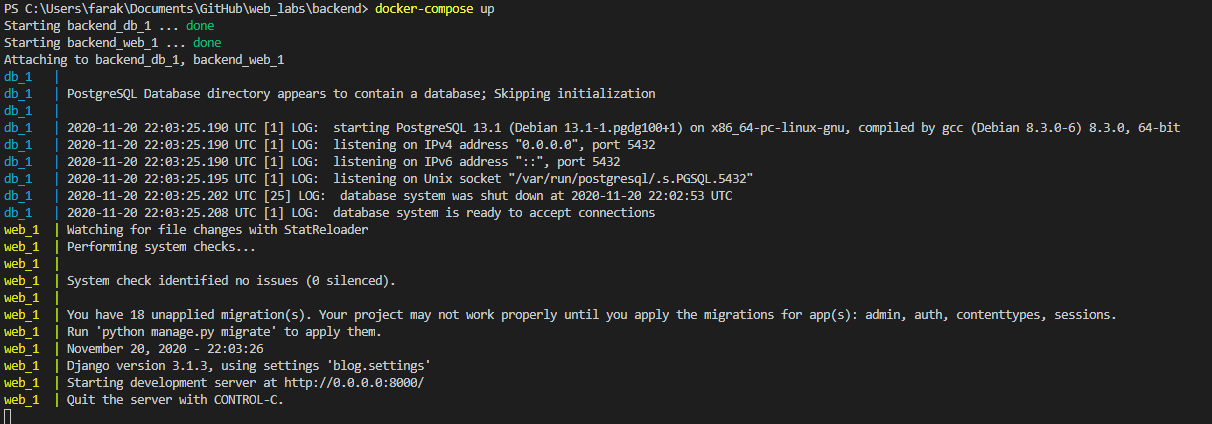


Рис. 9 Запущено два контейнера Docker с помощью Docker-Compose

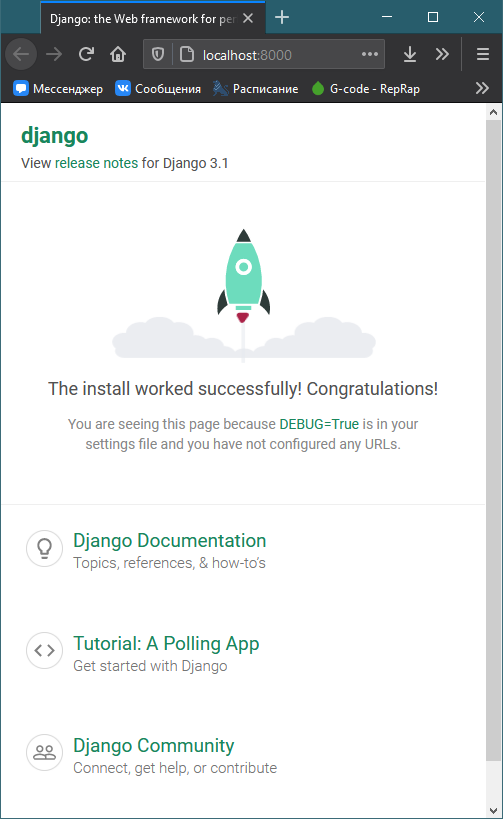


Рис. 10 Приложение Django корректно работает

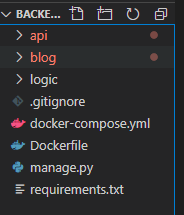


Рис. 11 Структура проекта

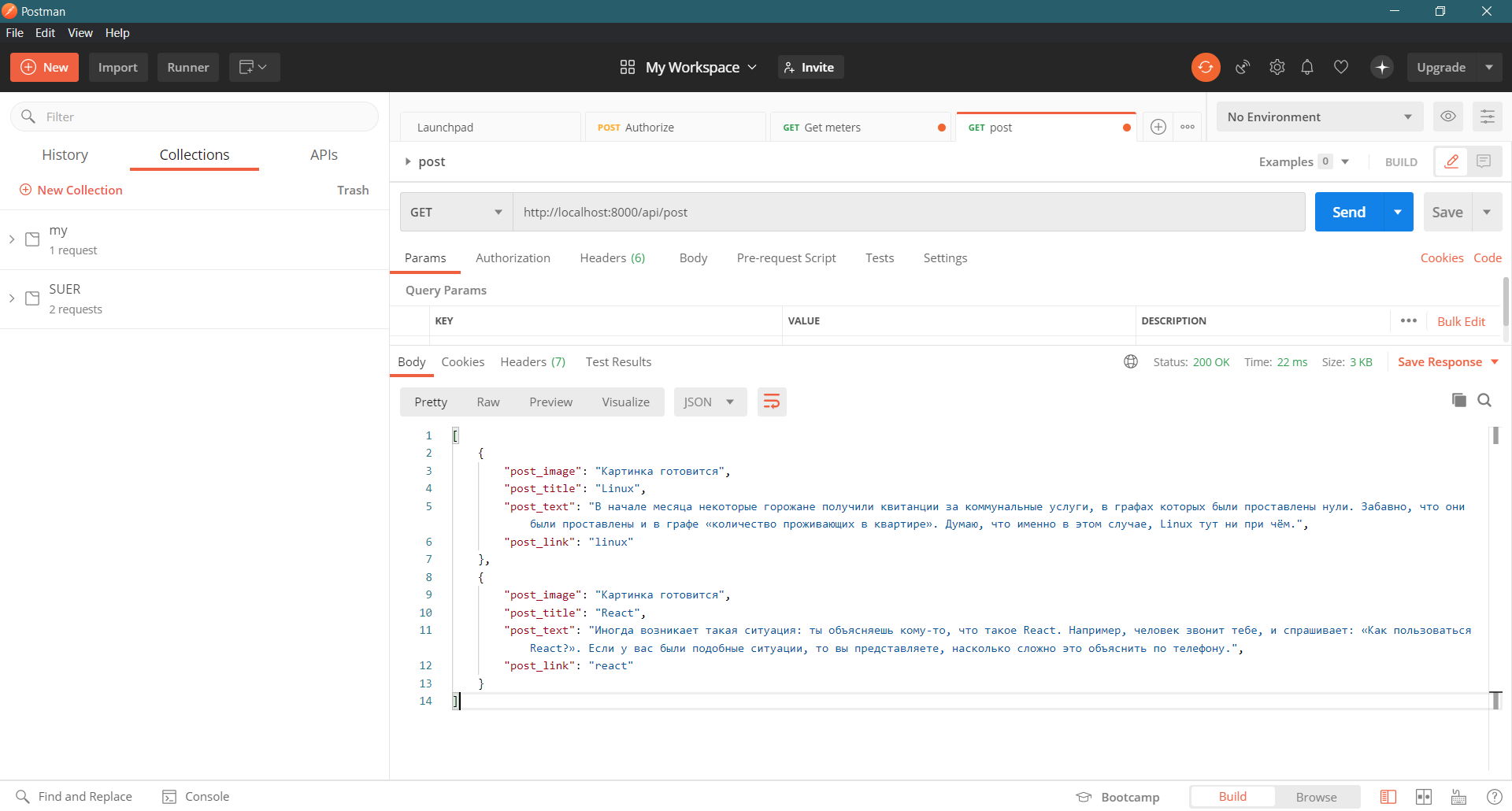


Рис. 12 Запрос к API в приложении Postman успешно получает данные



Рис. 13 Запрос к другой конечной точке API

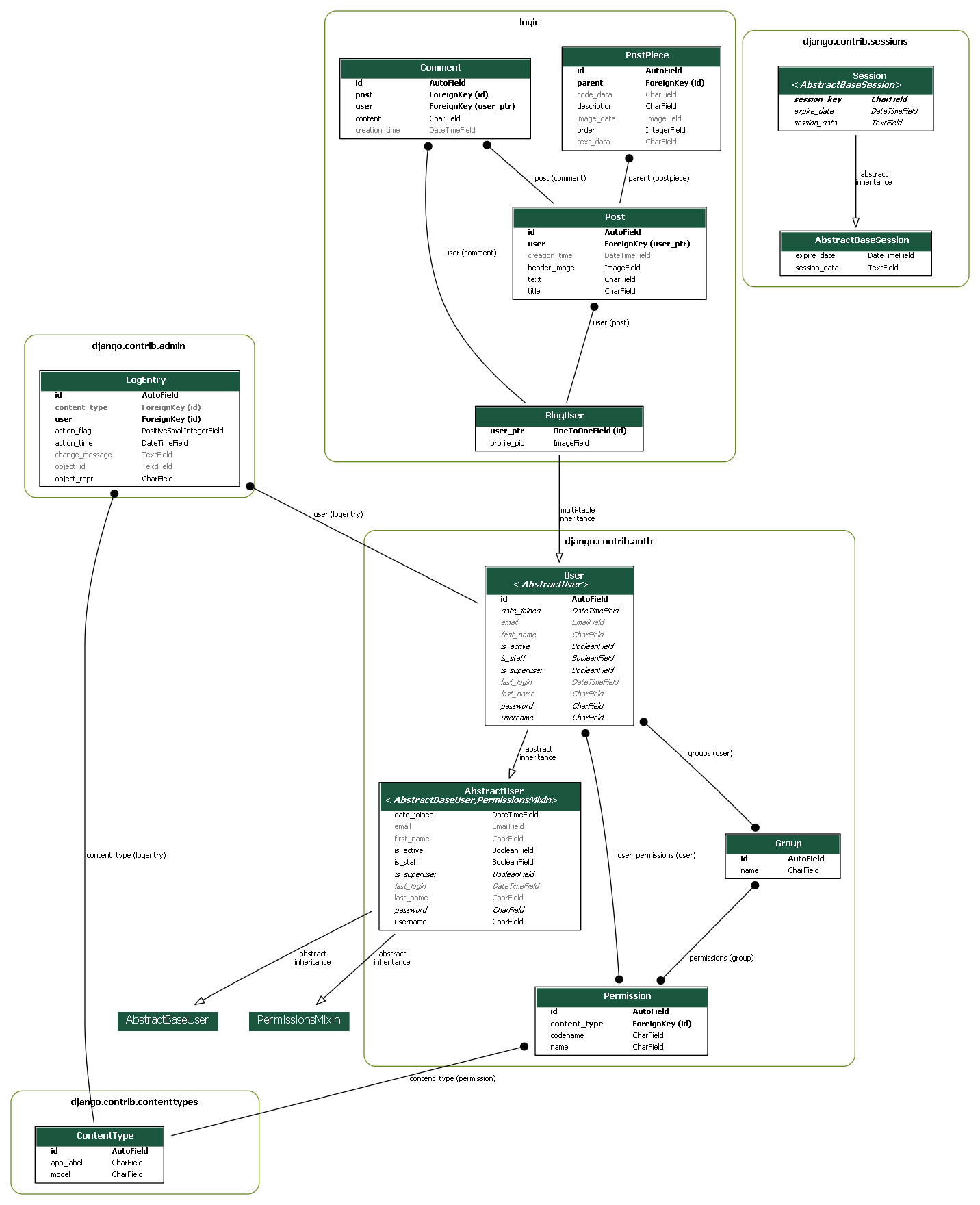
# Глава 4. Разработка и проектирование базы данных

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:

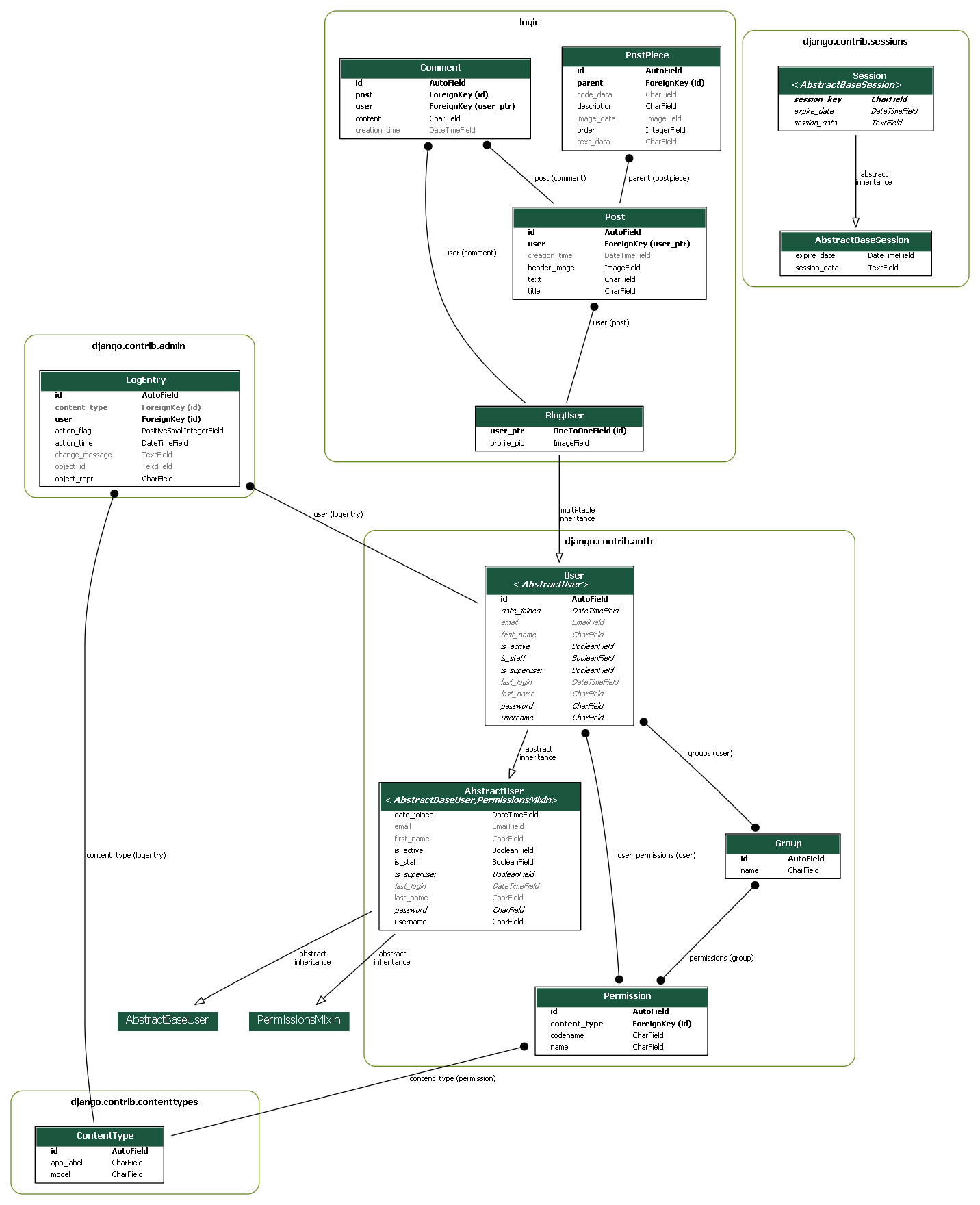
* Пользователь
* Пост
* Комментарий.

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

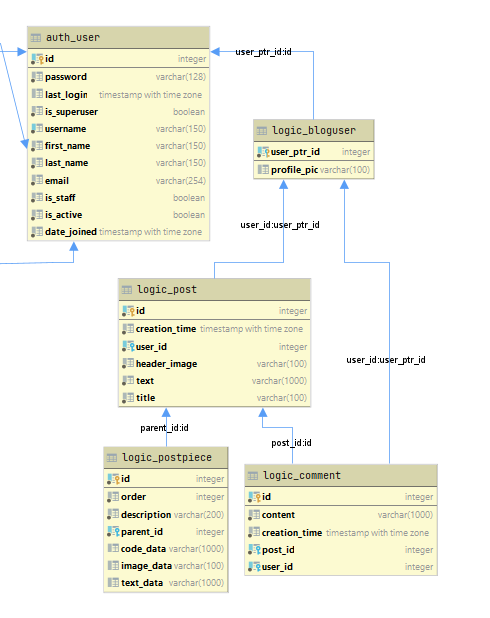
Структура базы данных, включающая в себя также стандартные модели Django:



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

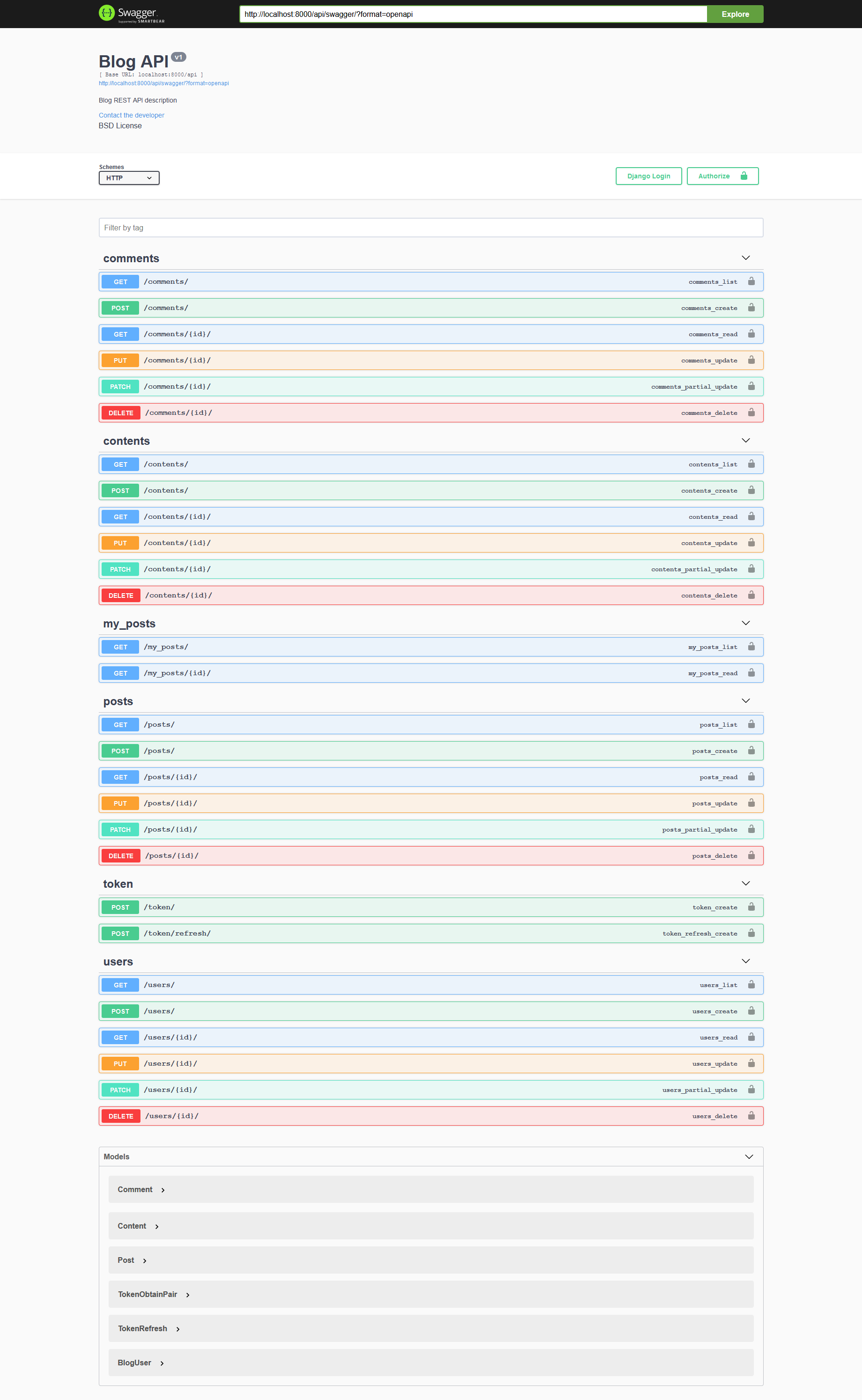
Для реализации REST API была использована библиотека Django REST Framework. В качестве вспомогательных библиотек использовались библиотеки Django-filters для обеспечения фильтрации запросов согласно параметрам и drf-simple-jwt для предоставления возможности аутентификации с помощью JWT токенов.

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**

****

**Описание моделей, использующихся в API**

****

# 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:

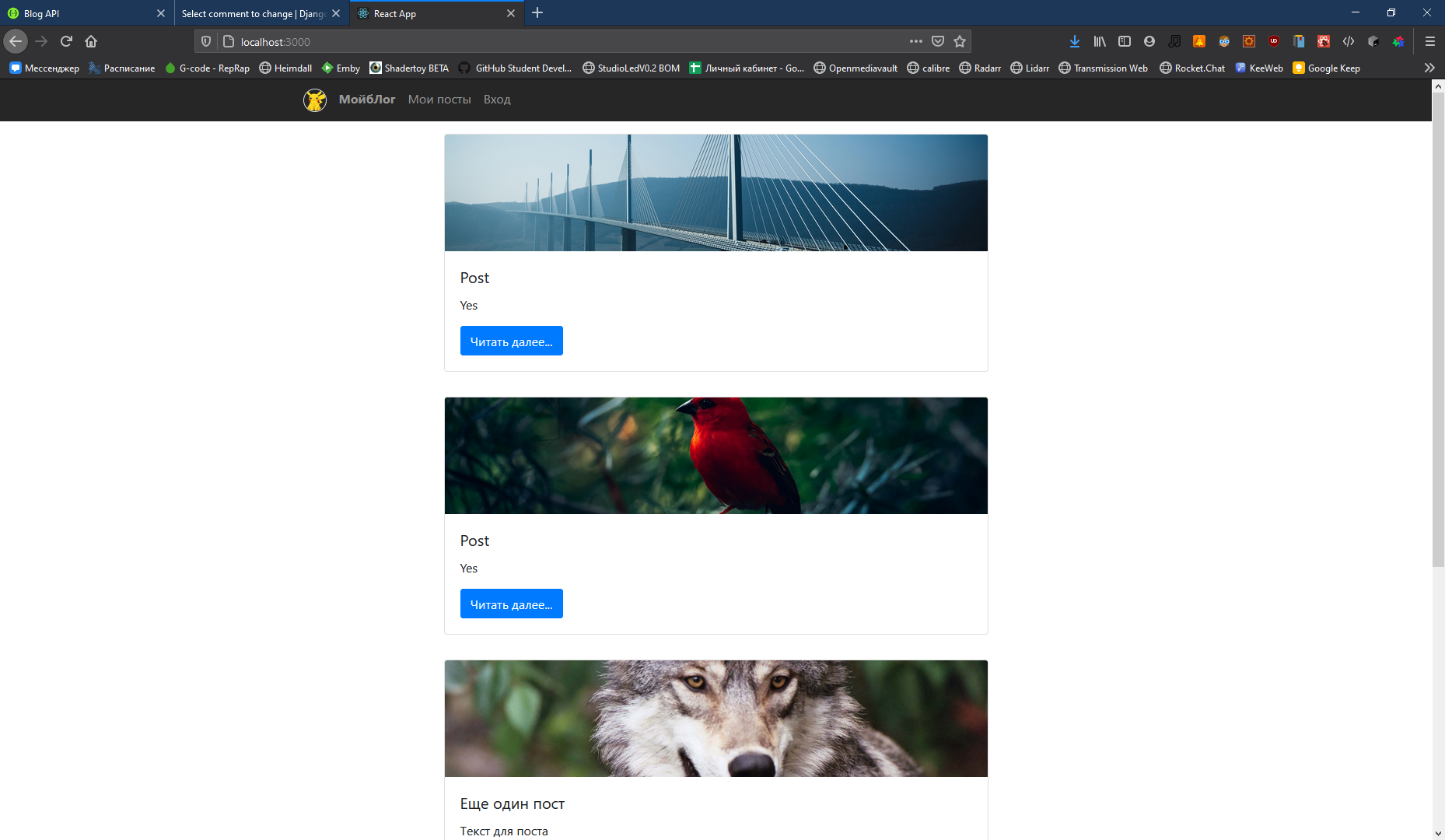


Рис. 14 Список постов всех пользователей

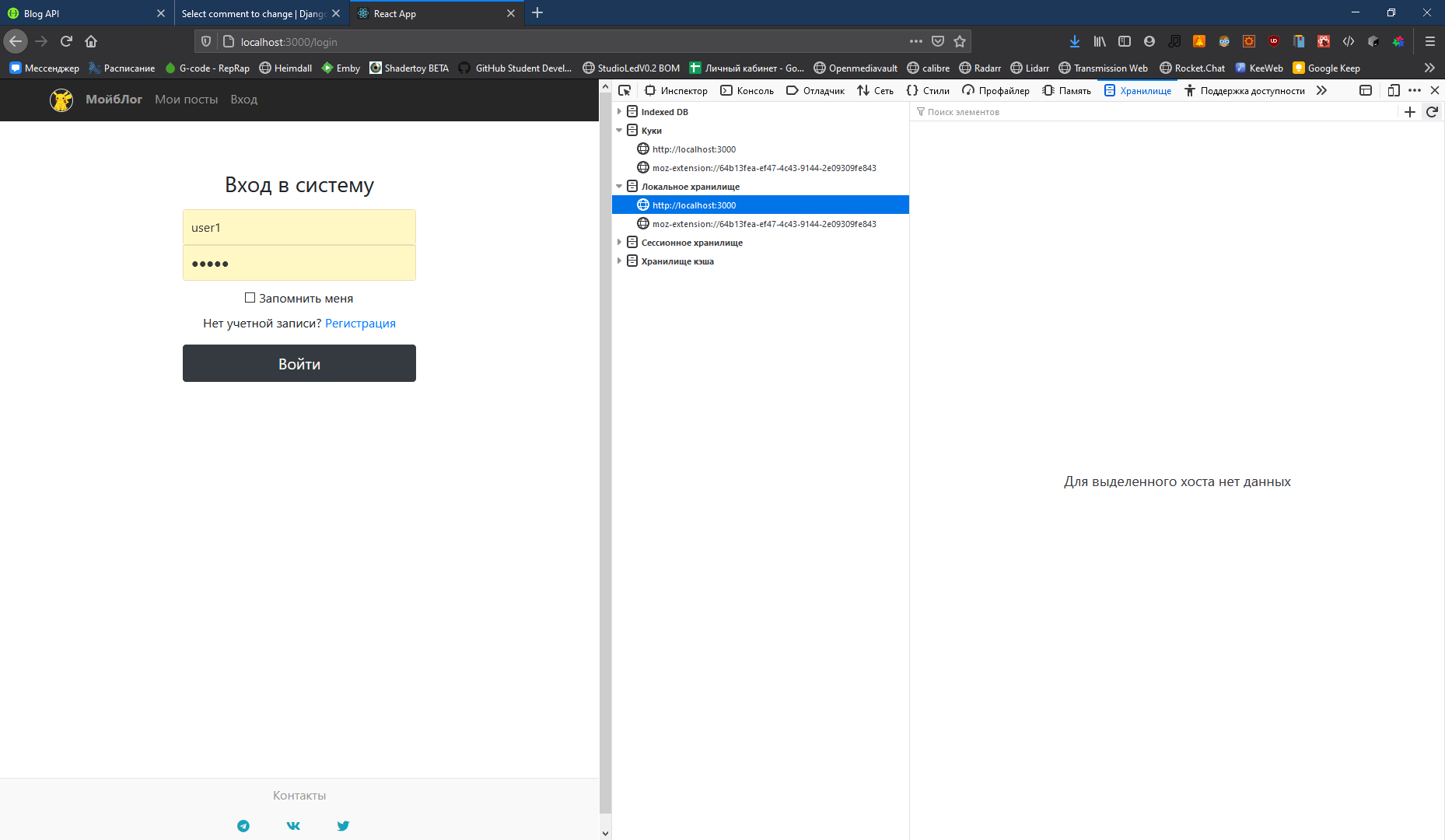


Рис. 15 Для просмотра страницы со своими постами необходимо войти

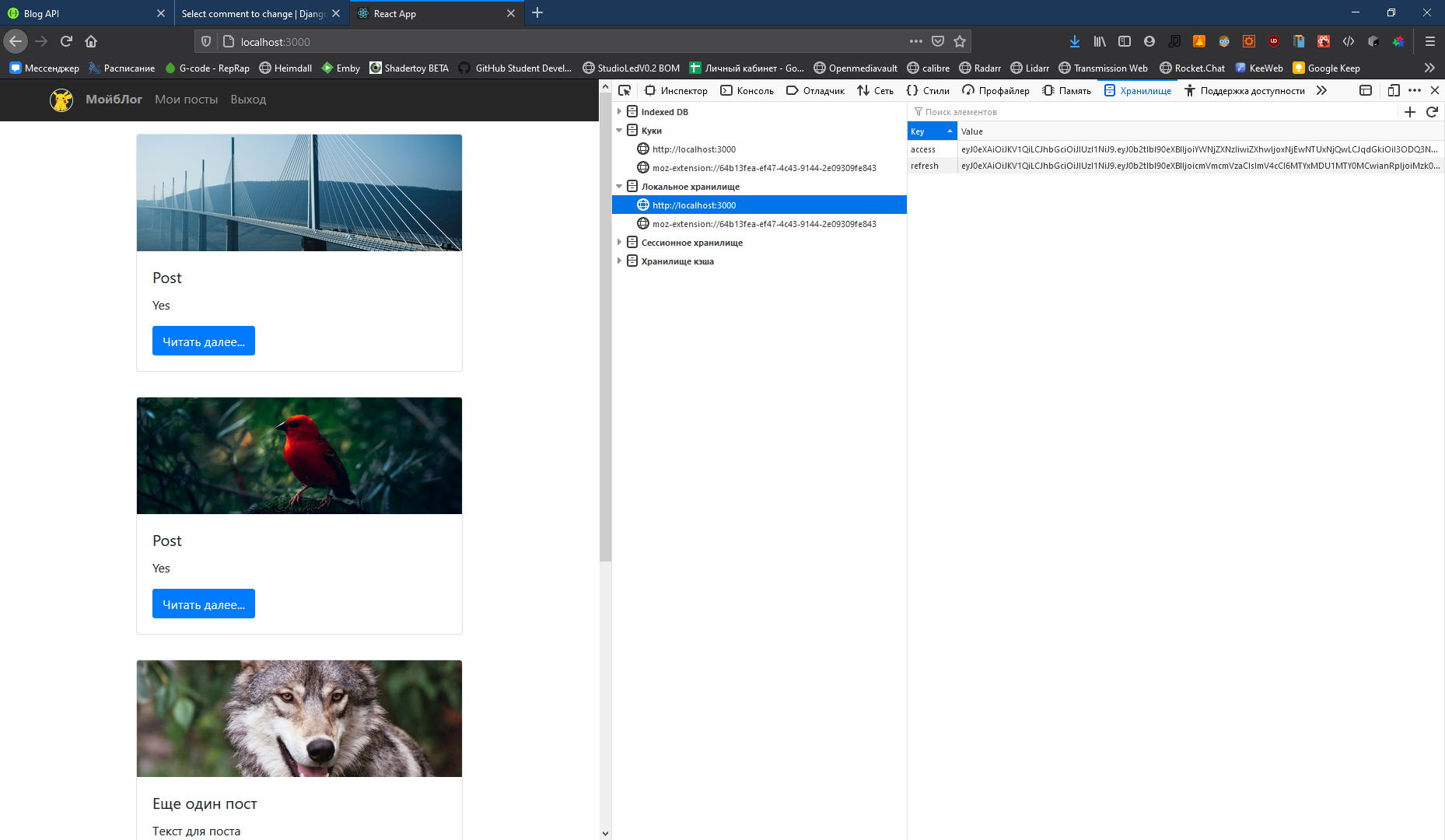


Рис. 16 Вход выполнен. Токен сохранен в localStorage

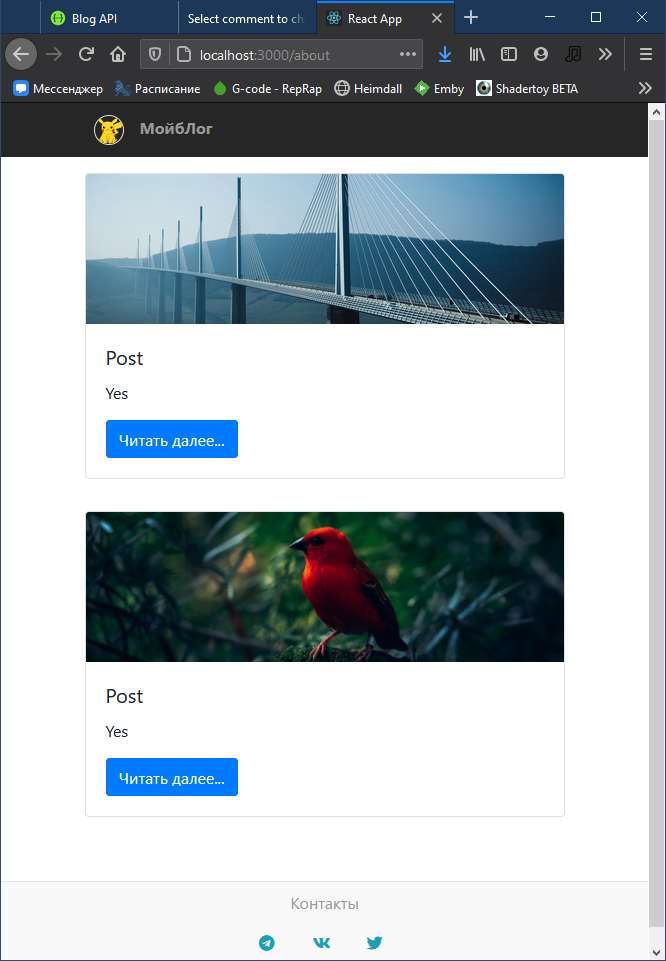


Рис. 17 При переходе на страницу «Мои посты» пользователь видит только свои посты

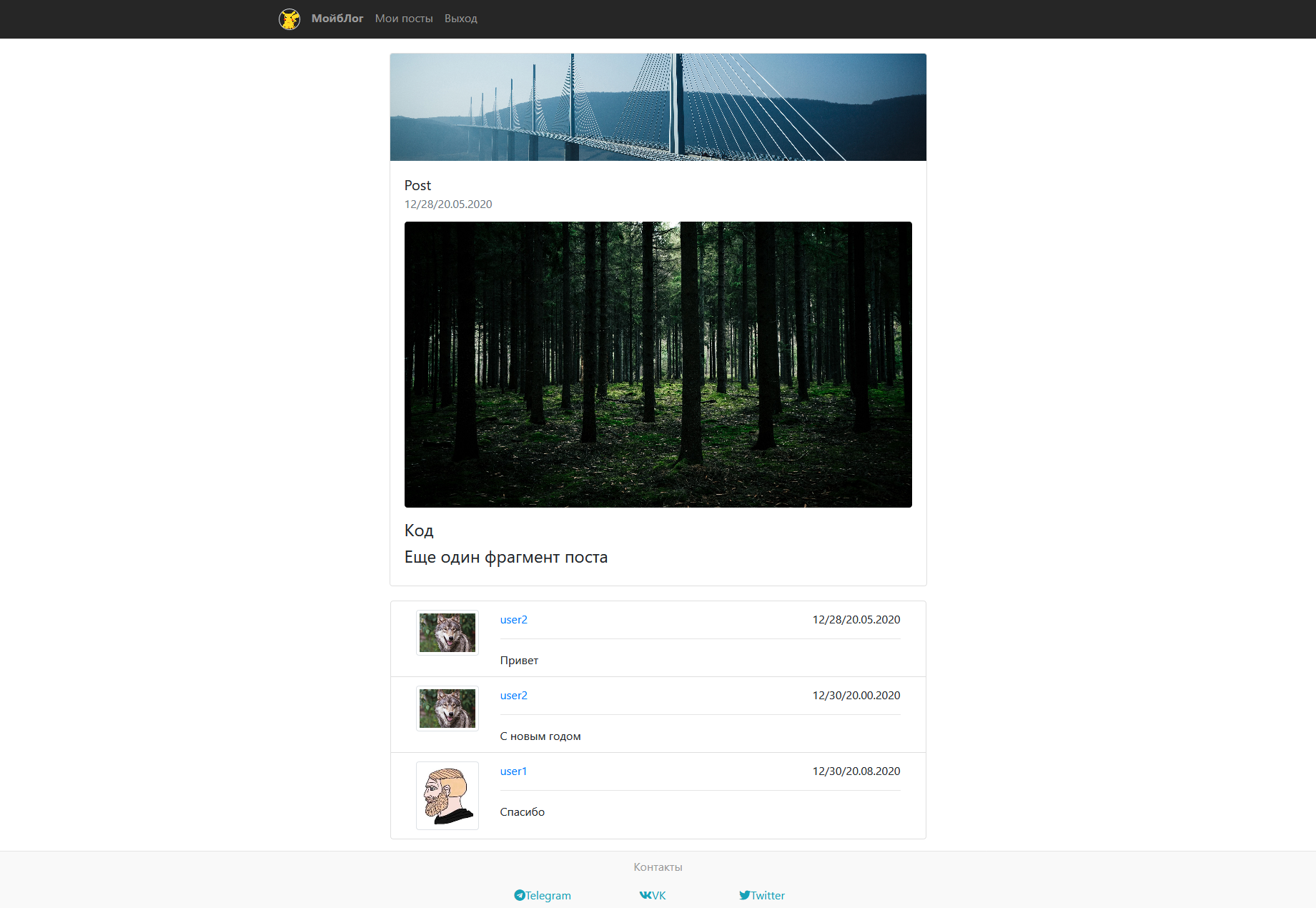


Рис. 18 Вид одного поста. Под постом расположен блок комментариев

# 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="anonymous">

    <!--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">

                <img class="rounded-circle border" width="30" height="30" src="static/images/logo.png"

                     alt="Логотип">

                <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">

                    <img class="post-img-top card-img-top" src="static/images/post\_1.png" alt="Картинка у поста">

                    <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 исходнички

Как же я вас люблю

Исходнички мои</code></pre>

                        <img class="mx-auto d-block rounded post-img m-3" src="static/images/post\_2.jpg"

                             alt="Картинка в посте">

                        <p>

                            Задача организации, в особенности же дальнейшее развитие 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">

                                    <img class="img-thumbnail" src="static/images/logo.png" alt="Картинка в отпуске">

                                </div>

                                <div class="col-10">

                                    <div class="text-inline justify-content-between d-flex">

                                        <a href="#" class="">use rname</a> <div class="comments-date">20.03.2865</div>

                                    </div>

                                    <hr/>

                                    <div class="text-inline">

                                        Значимость этих проблем настолько очевидна, 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">

                                    <img class="img-thumbnail" src="static/images/logo.png" alt="Картинка в отпуске">

                                </div>

                                <div class="col-10">

                                    <div class="text-inline justify-content-between d-flex">

                                        <a href="#" class="">use rname</a> <div class="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>

</footer>

</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="">{this.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}`)

    }

}