МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра информационных технологий

КУРСОВАЯ РАБОТА

РАЗРАБОТКА ПРОФЕССИОНАЛЬНОГО САЙТА ДЛЯ IT-СПЕЦИАЛИСТОВ

Работу выполнил _____ Е.А. Иванов
(подпись)

Направление подготовки 01.03.02 Прикладная математика и информатика

Направленность Программирование и информационные технологии

Научный руководитель
канд. физ.-мат. наук, доц. _____ Е.П. Лукащик
(подпись)

Нормоконтролер
ст.преп. _____А.В. Харченко
(подпись)

Краснодар
2021

РЕФЕРАТ

Курсовая работа содержит 27 страниц, 16 рисунков, 8 источников.

PYTHON, DJANGO, СОВРЕМЕННЫЙ САЙТ, СТАТЬИ

Объектом исследования являются язык программирования Python и фреймворк Django.

Цель курсовой работы – изучение языка программирования Python и освоение фреймворка Django. Итог проделанной работы – создание современного сайта для программистов.

В результате была реализована программная реализация сайта, функциями которого являются:

– возможность делиться опытом и знаниями с другими программистами;

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Интернет стал наиболее эффективным средством рекламы и продвижения и является одним из важных элементов современной цивилизации. Интернет может удовлетворить все потребности современного человека: это покупки, заключение деловых отношений, поиск клиентов и так далее. Но многие заходят в Интернет читать, развлекаться, общаться, узнавать что-то новое для себя. Поэтому создание сайтов стало таким популярным.

В начале XXI века появилось множество инновационных технологий, которые развиваются с каждым днем все быстрее и быстрее. Многие программисты, которые не успевают за техническим прогрессом и темпом развития программирования, остаются за бортом. В такое время очень важно делиться опытом с другими, помогать развиваться в IT-сфере.

В данной курсовой работе будет рассмотрено создание программной реализации сайта для программистов с целью делиться опытом и навыками с другими IT-специалистами.

Целью курсовой работы является изучение популярного языка программирования Python, фреймворка Django и создание современного сайта для программистов.

1 Теоретические сведения и используемые технологии

1.1 Python

1.1.1 Общая информация

Python - высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ.

Язык является полностью объектно-ориентированным — всё является объектами. Необычной особенностью языка является выделение блоков кода пробельными отступами.

Синтаксис ядра языка минималистичен, за счёт чего на практике редко возникает необходимость обращаться к документации.

Сам же язык известен как интерпретируемый и используется в том числе для написания скриптов.

Недостатками языка являются зачастую более низкая скорость работы и более высокое потребление памяти написанных на нём программ по сравнению с аналогичным кодом, написанным на компилируемых языках, таких как Си или C++.

1.1.2 Где используется

Чаще всего Python используется в веб-разработке и анализе больших данных. Чтобы дополнить функциональность языка, используются разные фреймворки: Django, Pyramid, Flask и другие.

Но Python подходит и для создания прикладных приложений или игр. Например, графический редактор GIMP написан именно на Python. Торрент-клиент BitTorrent вплоть до 6 версии тоже разработан на этом языке. Python

применялся и в ходе разработки игровых проектов класса AAA: EVE Online, Battlefield 2, World of Tanks и других.

Чаще всего в таких случаях на Python пишут один из компонентов проекта. То есть не всю игру или приложение, а какой-то модуль, например серверную часть.

А ещё язык Python используется в системном администрировании, для автоматизации задач. Он задействован практически во всех серверах с ОС Linux.

«Питон» очень хорош и для работы с данными в научных исследованиях — в набирающей обороты Data Science. На этом языке пишут алгоритмы машинного обучения и анализа данных.

1.1.3 Python в крупных компаниях

В Amazon и Spotify используют Python для анализа пользовательских данных, информации о продажах и разработки персонализированных рекомендаций.

В Walt Disney применяют этот язык в качестве скриптового для анимации.

YouTube и Instagram… Эти проекты полностью написаны на Python. Кроме того, холдинг Alphabet использует «питон» для скрейпинга в Google — извлечения данных со страниц веб-ресурсов.

Netflix создала свой рекомендательный сервис с нуля на Python.

Autodesk в своём редакторе 3D-анимации Maya с помощью Python создаёт мультипликацию. Так же язык использует студия Pixar.

JPMorgan Chase, крупный американский финансовый холдинг, применяет Python для прогнозирования рынка.

NASA работает с проектами на этом языке программирования, чтобы проводить научные вычисления.

1.2 Django

1.2.1 Общая информация

Django - свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC. Проект поддерживается организацией Django Software Foundation.

Сайт на Django строится из одного или нескольких приложений, которые рекомендуется делать отчуждаемыми и подключаемыми. Это одно из существенных архитектурных отличий этого фреймворка от некоторых других.

Один из основных принципов фреймворка — DRY (Don't repeat yourself).

Также, в отличие от других фреймворков, обработчики URL в Django конфигурируются явно при помощи регулярных выражений.

Для работы с базой данных Django использует собственный ORM, в котором модель данных описывается классами Python, и по ней генерируется схема базы данных.

1.2.2 Архитектура

Архитектура Django похожа на «Модель-Представление-Контроллер» (MVC). Контроллер классической модели MVC примерно соответствует уровню, который в Django называется Представление, а презентационная логика Представления реализуется в Django уровнем Шаблонов. Из-за этого уровневую архитектуру Django часто называют «Модель-Шаблон-Представление» (MTV).

Первоначальная разработка Django как средства для работы новостных ресурсов достаточно сильно отразилась на его архитектуре: он предоставляет ряд средств, которые помогают в быстрой разработке веб-сайтов информационного характера. Так, например, разработчику не требуется

создавать контроллеры и страницы для административной части сайта, в Django есть встроенное приложение для управления содержимым, которое можно включить в любой сайт, сделанный на Django, и которое может управлять сразу несколькими сайтами на одном сервере. Административное приложение позволяет создавать, изменять и удалять любые объекты наполнения сайта, протоколируя все совершённые действия, и предоставляет интерфейс для управления пользователями и группами (с пообъектным назначением прав).

1.2.3 Возможности

У Django очень много необходимых для качественной разработки возможностей. Вот некоторые из них:

1) ORM, API доступа к БД;
2) встроенный интерфейс администратора, с уже имеющимися переводами на многие языки;
3) диспетчер URL на основе регулярных выражений;
4) расширяемая система шаблонов с тегами и наследованием;
5) система кеширования;
6) “Generic views” – шаблоны функций котроллеров;
7) авторизация и аутентификация, подключение внешних модулей аутентификации;
8) библиотека для работы с формами.

2 Программная реализация и функционал

2.1 Функции, реализованные сайте

На сайте реализованы следующие функции:

1) чтение статей и комментариев к ним;
2) регистрация и авторизация;
3) право оставлять комментарии (могут только авторизованные пользователи);
4) подборка похожих постов;
5) поиск статей по названию через поисковую строку;
6) фильтрация постов по популярности и дате публикации;
7) поиск постов по категориям;
8) возможность добавления постов для прочтения позже и их удаления (могут только авторизованные пользователи);
9) возможность написать автору статьи на почту, чтобы задать вопрос или скорректировать информацию по статье.

2.2 Главная страница

На главной странице пользователя встречает приветственный заголовок сайта и основная навигация по сайту (рис. 1).

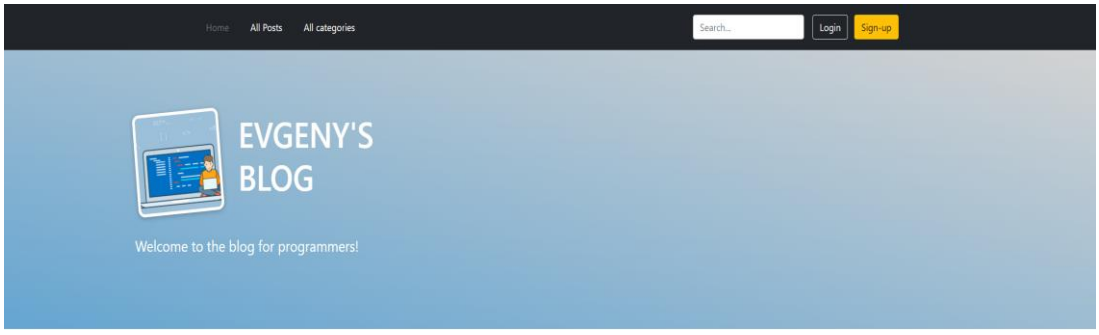

Рисунок 1 – Навигация и заголовок

В навигации доступны все страницы, которые может посетить пользователь. Также располагается поисковая строка по сайту и кнопки регистрации и авторизации.

Также пользователь может посмотреть расположенные ниже приветствия самые популярные посты, которые фильтруются по количеству зашедших на пост зарегистрированных пользователей. В целях объективности просмотры анонимных пользователей не учитываются. Также, в случае неоднократного посещения поста зарегистрированным пользователем, будет засчитан только один просмотр. Панель самых популярных постов вы можете увидеть на рисунке 2.

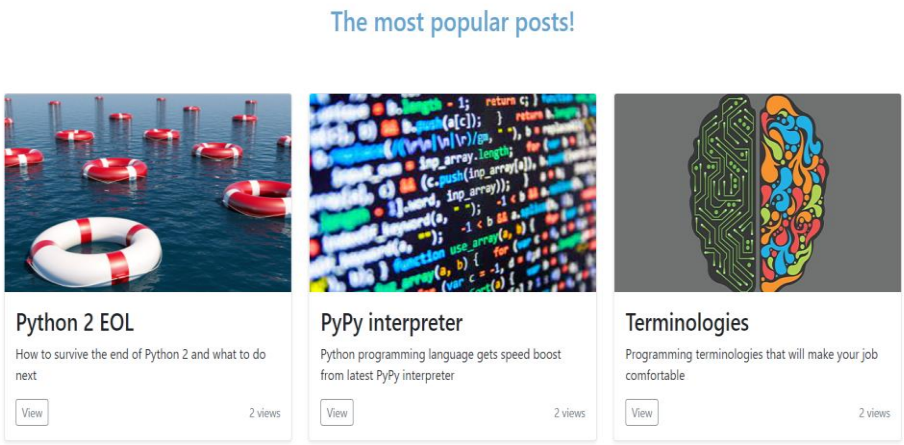

Рисунок 2 – Панель самых популярных постов

Также пользователь может просмотреть самые последние, новые посты, которые располагаются ниже самых популярных. Эти посты фильтруются на основе даты их добавления. Панель самых последних постов вы можете увидеть на рисунке 3.

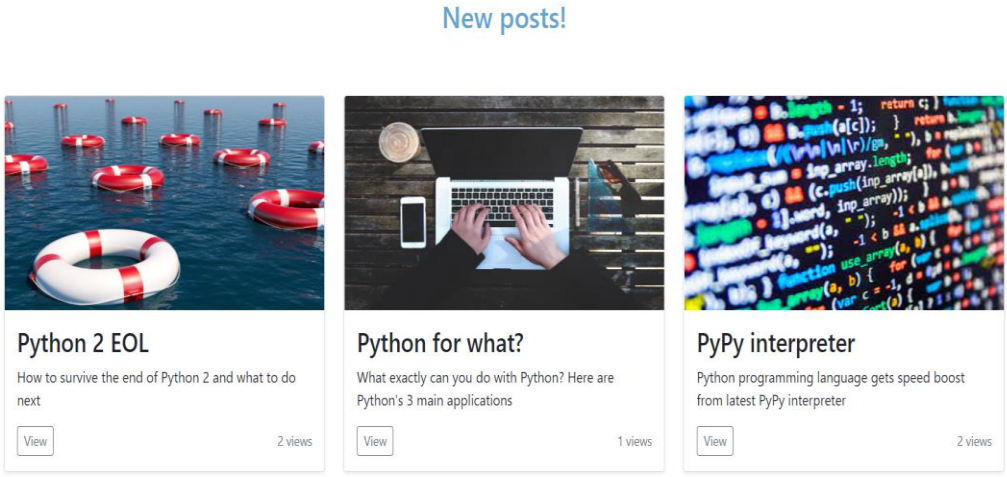

Рисунок 3 – Панель самых последних постов

2.3 Регистрация

На сайте присутствует регистрация новых пользователей, зашедших на страницу. Окно представляет собой 4 поля: никнейм пользователя, его почта, пароль и подтверждение пароля. В случае несовпадения поля пароля с полем его подтверждения пользователь увидит предупреждение. Все поля, кроме почты, необходимо заполнить, чтобы создать нового пользователя.

Необходимость создание пользователя на сайте может возникнуть у посетителей страницы из-за урезанного функционала для анонимных гостей.

Также присутствует возможность перехода пользователя на страницу авторизации при нажатии “Sign in” в случае, если у него уже существует аккаунт и он хочет в него зайти.

Панель регистрации вы можете увидеть на рисунке 4.

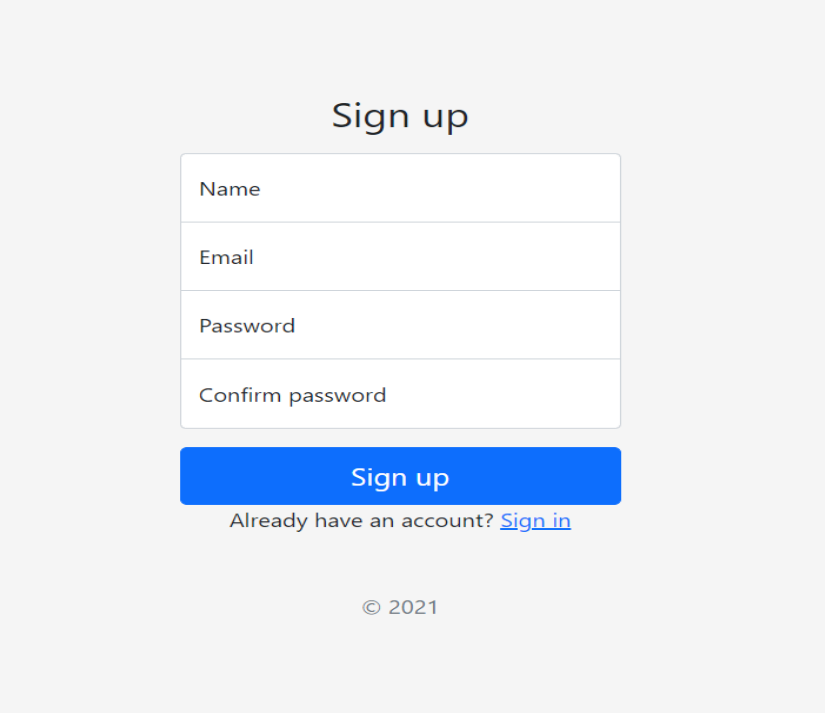

Рисунок 4 – Панель регистрации пользователя

2.4 Авторизация

В случае существования аккаунта у пользователя у него нет необходимости регистрироваться заново, поскольку его аккаунт уже есть в базе данных.

Для успешной авторизации пользователю необходимо ввести его никнейм и пароль. После чего ему будет доступен полный функционал сайта.

У анонимных пользователей отсутствует возможность комментирования постов, а также добавления постов во вкладу “Прочитать позже”.

В случае, если у пользователя нет аккаунта, но он перешел на страницу авторизации, у него есть возможность перейти на страницу регистрации, нажав на “Sign up”, для создания аккаунта на сайте.

Панель авторизации вы можете увидеть на рисунке 5.

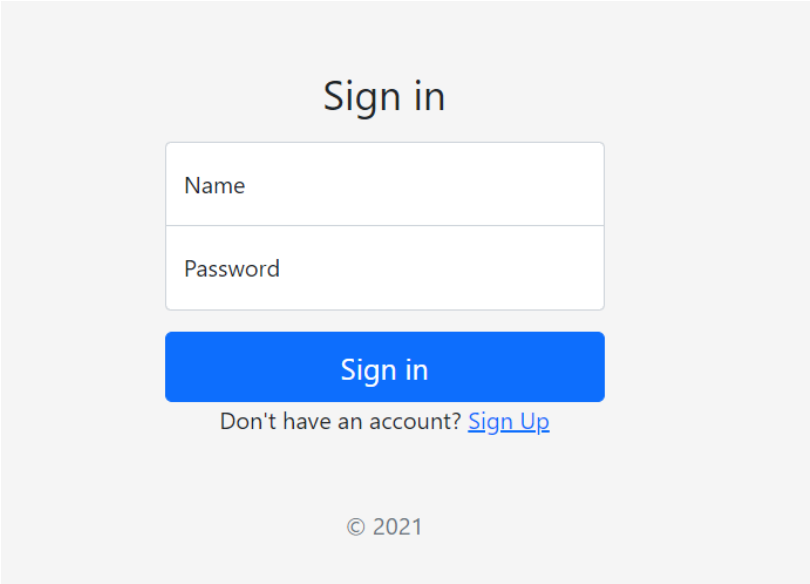

Рисунок 5 – Панель авторизации пользователя

2.5 Все посты

Перейдя в навигации вверху сайта пользователь может попасть на страницу со всеми постами, которые есть на сайте. Эти посты отфильтрованы по дате и времени их добавления. Страницу со всеми постами вы можете увидеть на рисунке 6.

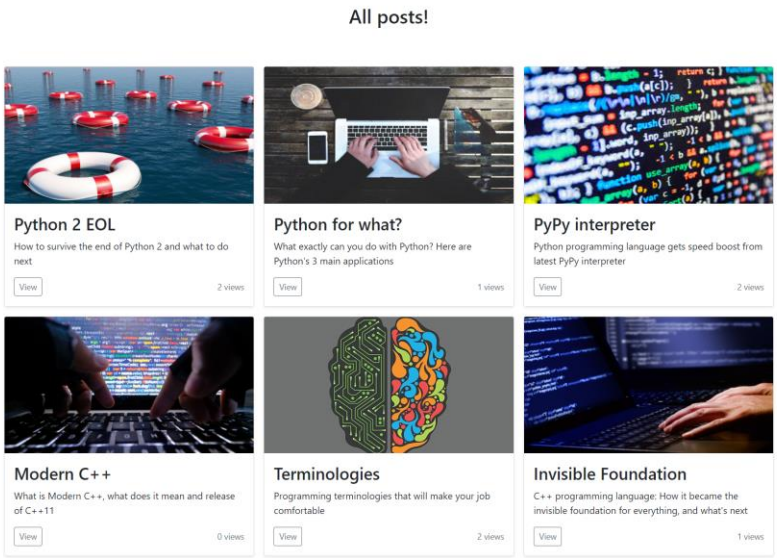

Рисунок 6 – Страница со всеми постами

2.6 Подробная информация о посте

Пользователь может перейти на детальное содержание поста, нажав кнопку View в левом нижнем углу каждого поста.

На странице с подробной информацией о посте можно увидеть карточку с названием поста, картинкой, ее автором, датой и временем добавления на сайт. У пользователя есть возможность написать автору статьи письмо на почту, наведя на его имя. Также в карточке присутствует тег поста, он определяет, какую тему затрагивает автор статьи. Эти теги нужны для сборки статей в категории, для возможности перехода с определенной статьи в категорию с этой тематикой, а также для подборки рекомендаций пользователю. Также только у авторизованных пользователей есть возможность добавлять посты на страничку “Read Later”. Туда сохраняются все статьи, которые пользователь выберет. У авторизованных пользователей для этого есть кнопка “Read Later” на каждой странице подробной информации о статье (рис. 9). Далее идет текст статьи с комментариями пользователей. В зависимости от того, авторизован ли пользователь или нет, будет отображаться окно для комментария (рис. 10). Комментарии остаются с ником зарегистрированного пользователя, самим комментарием и датой его публикации. После чего сайт предлагает пользователю посты на тематику, с которой связана открытая гостем статья. Все это можно увидеть на рисунках 7 и 8.

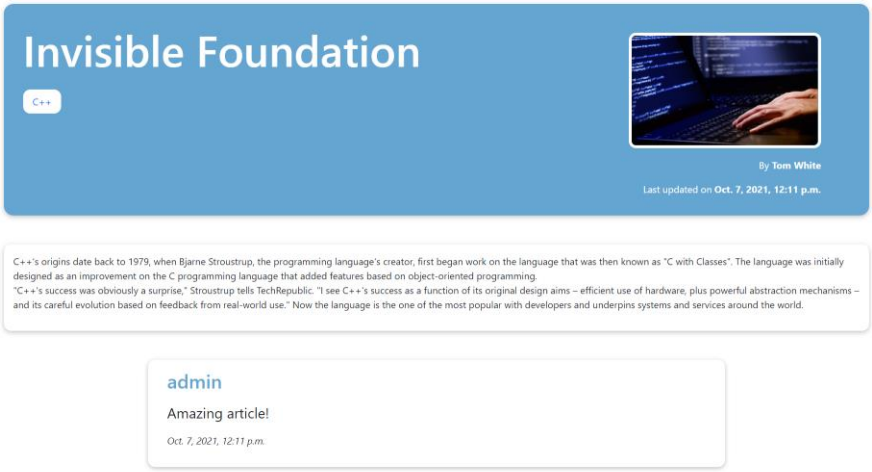

Рисунок 7 – Подробная информация о посте

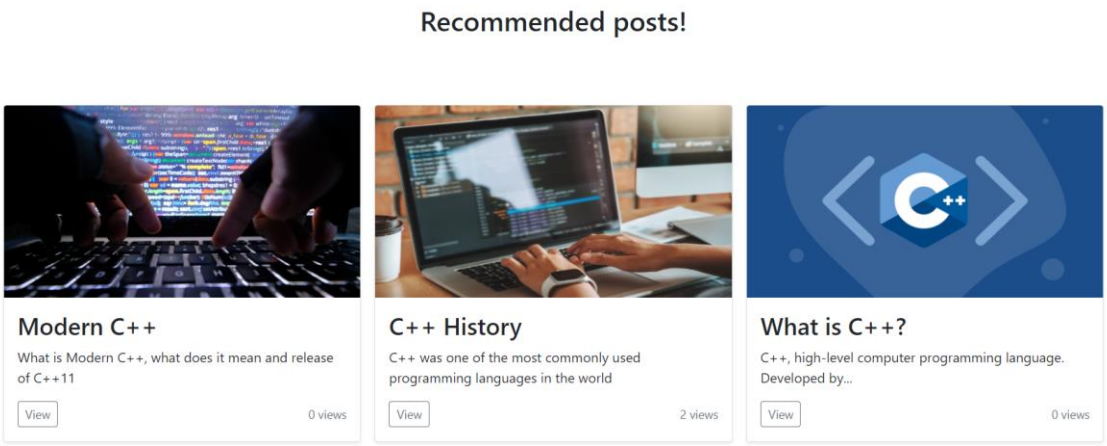

Рисунок 8 – Подробная информация о посте

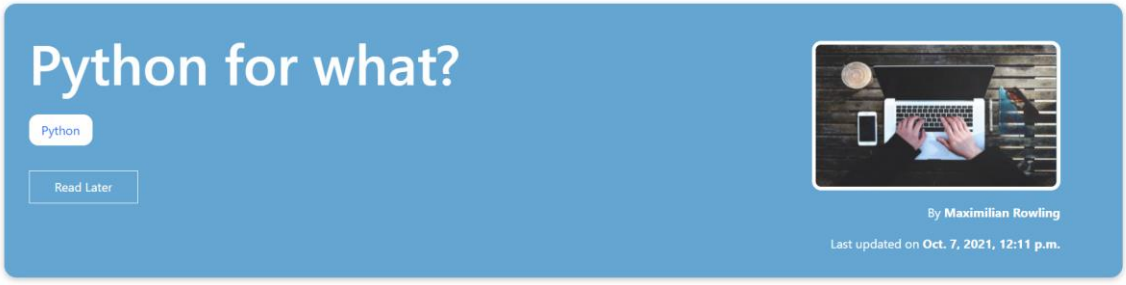

Рисунок 9 – Кнопка “Read Later”

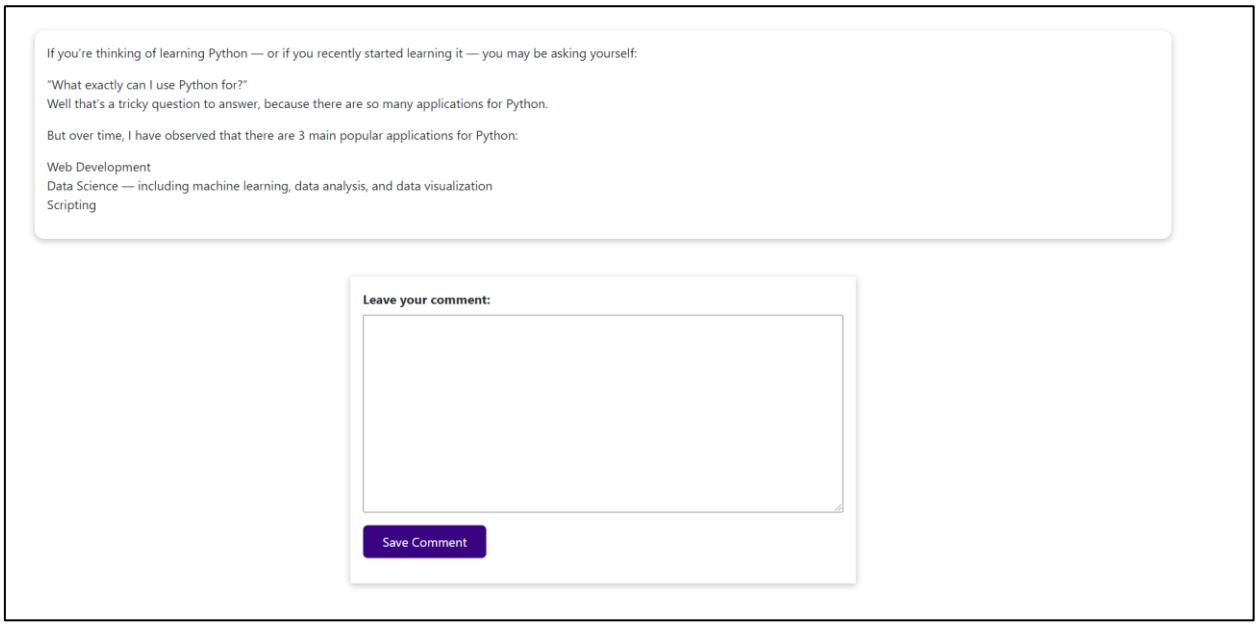

Рисунок 10 – Окно комментария

2.7 Прочитать позже

Данная страница доступна только для авторизованных пользователей. Именно сюда попадают посты, на страницах которых пользователь нажмет соответствующую кнопку “Read Later”. Количество постов на странице не ограничено. При нажатии на определенный пост, пользователь попадает на его подробное описание. Оттуда он может удалить пост со страницы “Read Later”.

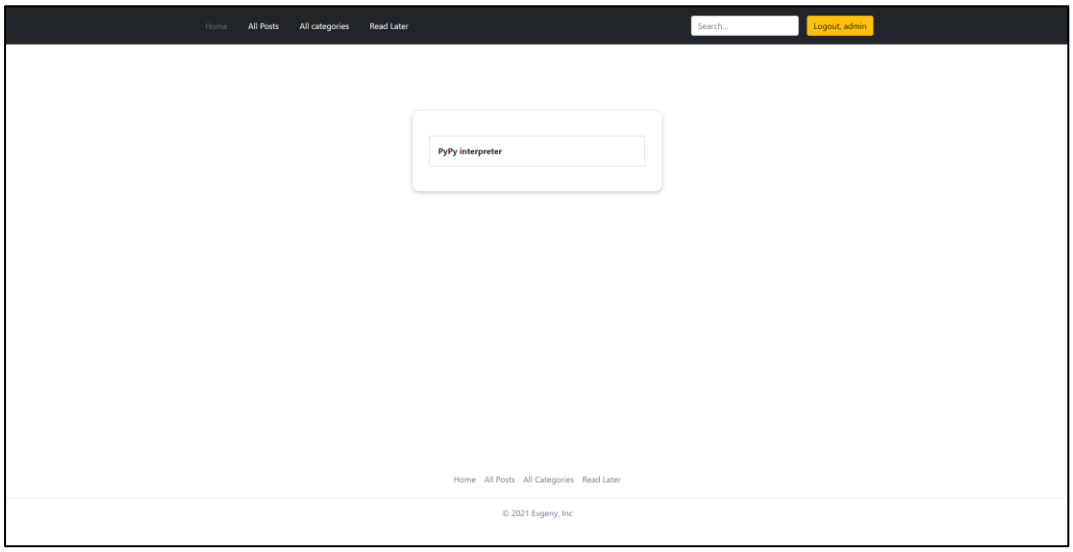

Рисунок 11 – Страница Read Later

Рисунок 12 – Кнопка удаления из “Read Later"

2.8 Категории

Из меню навигации пользователь может попасть на страницу всех категорий (рис. 13), которые есть на сайте. При добавлении тега в меню администратора категория добавляется на страницу автоматически. При нажатии на категорию пользователю предлагается список статей с этой категорией (рис. 14). Если статей в категории нет, то пользователя об этом сайт оповещает (рис. 15).

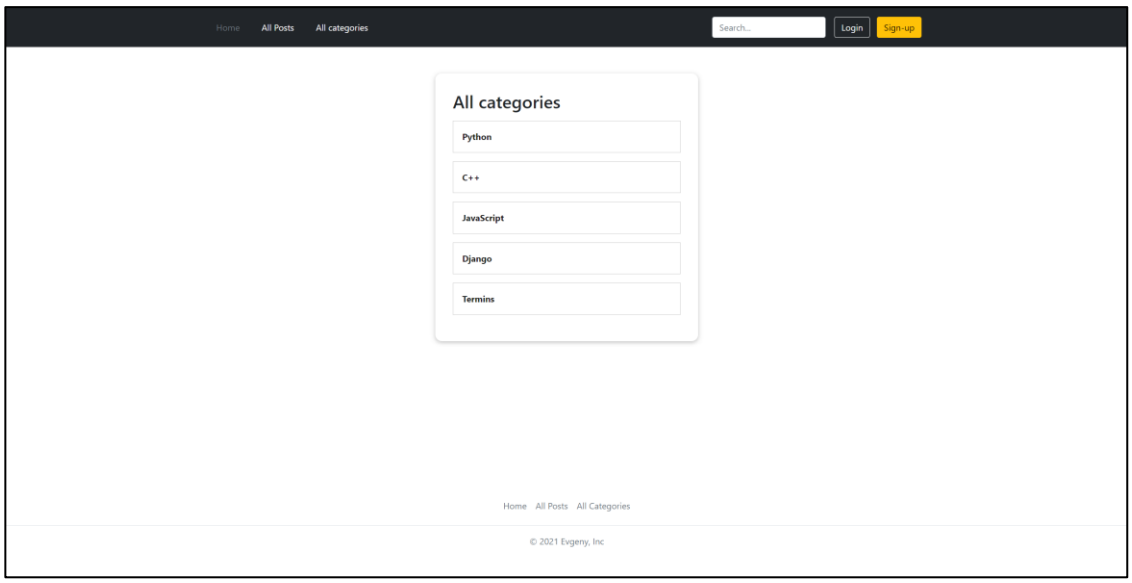

Рисунок 13 – Список категорий

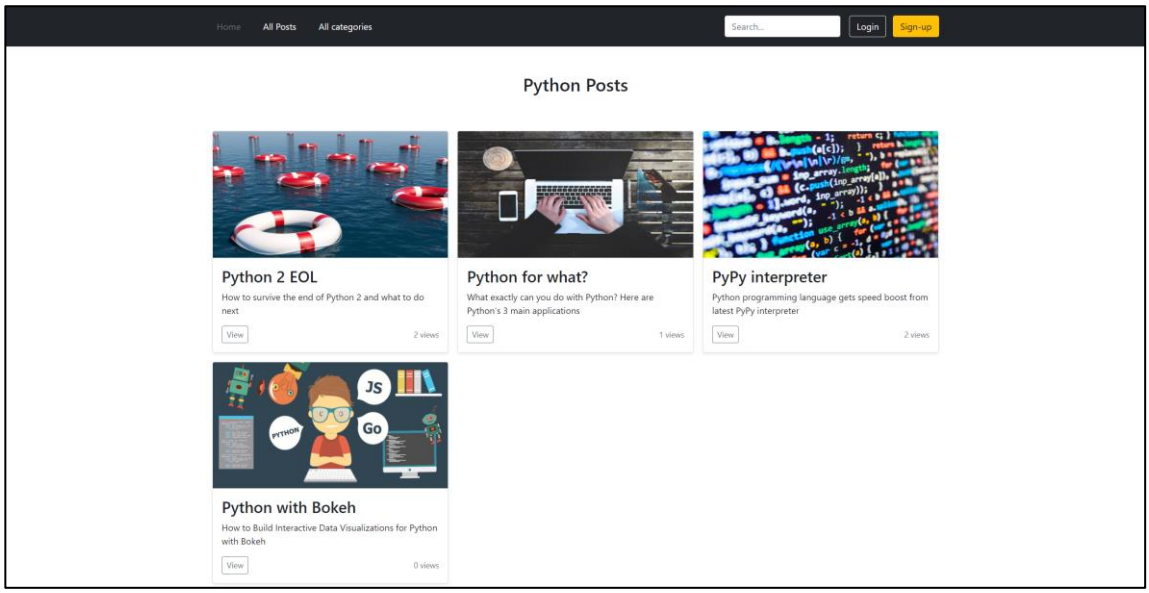

Рисунок 14 – Список постов по категории

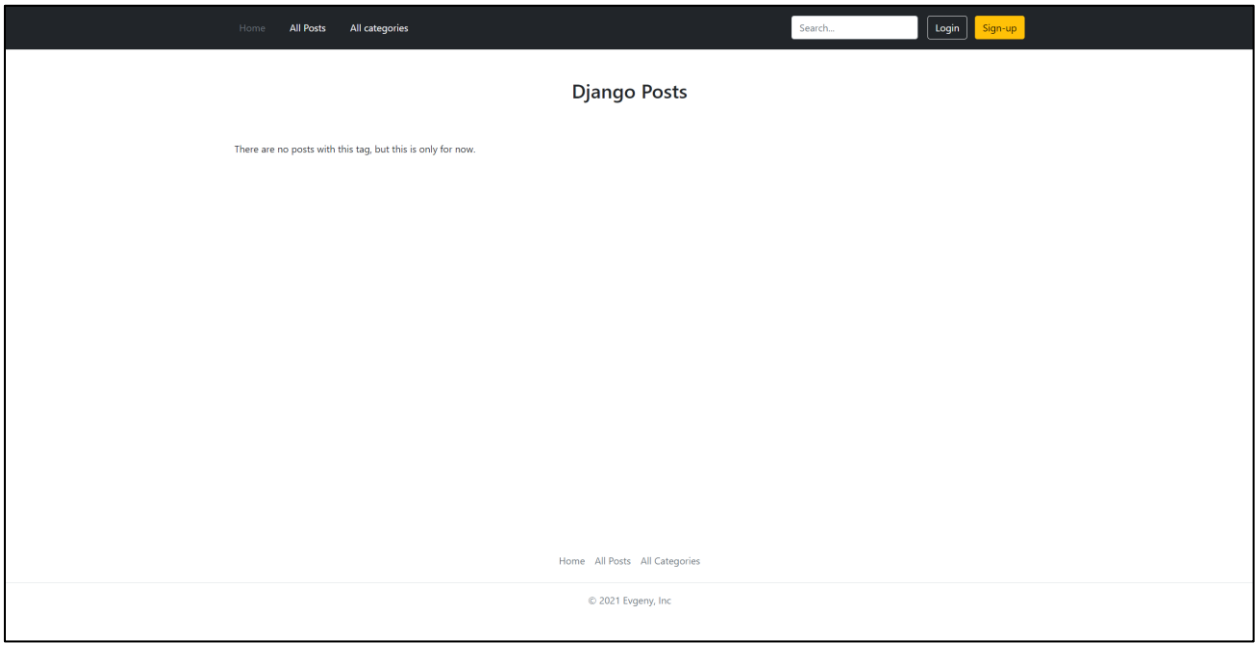

Рисунок 15 – Отсутствие постов в категории

2.9 Поисковая строка

В меню навигации пользователь может обратиться к поисковой строке. Поиск ведется исключительно по названию статьи (рис. 16).

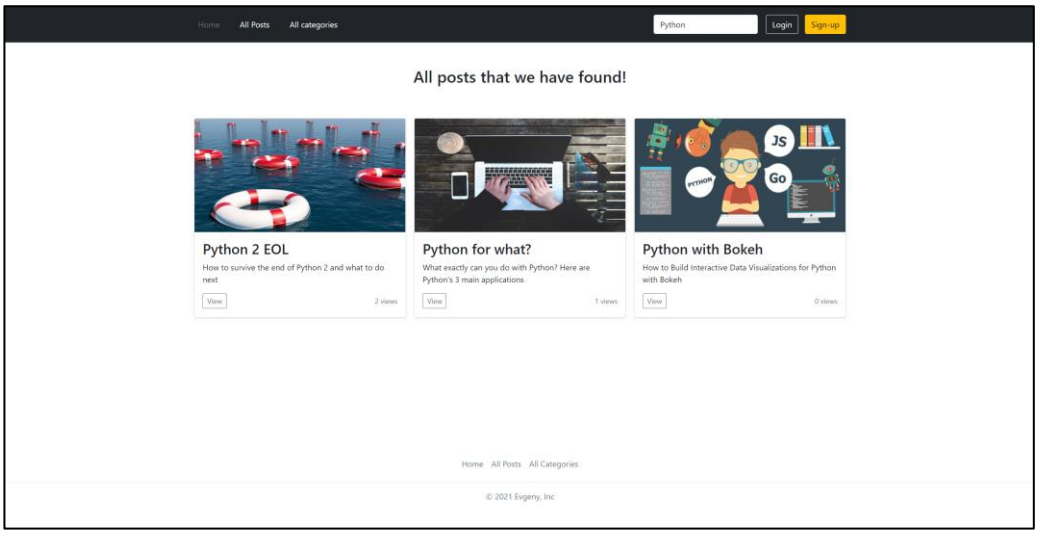

Рисунок 16 – Поисковая строка

ЗАКЛЮЧЕНИЕ

В результате для рассматриваемой проблемы нехватки знаний в сфере IT разработана программная реализация сайта, позволяющая пользователям делиться опытом и навыками в сфере программирования. Сайт предоставляет пользователям наиболее нужную им информацию, а именно – возможность просмотра статей по выбранной ими теме с помощью наличия на сайте вкладки “Категории” и поисковой строки. В дальнейшем разработанный web-сайт можно будет использовать в реальной жизни. Также возможно расширение списка предлагаемых пользователю постов и их категорий. В качестве инструментария выл выбран язык программирования Python и веб-фреймворк Django.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Документация по веб-фреймворку Django: официальный сайт – URL: https://www.djangoproject.com (дата обращения: 29.10.2021).

2 Документация по языку программирования Python: официальный сайт – URL: https://www.python.org/doc/ (дата обращения 20.10.2021).

3 Справочник по HTML: официальный сайт – URL: http://htmlbook.ru/html (дата обращения 15.10.2021).

4 Руководство по CSS: официальный сайт – URL: https://developer.mozilla.org/ru/docs/Web/CSS/Reference (дата обращения 16.10.2021).

5 Reddit Blog: официальный сайт – URL: https://www.redditinc.com/blog (дата обращения 10.10.2021).

6 Курс по веб-фреймворку Django [сайт] – URL: https://itproger.com/course/django (дата обращения 29.10.2021).

7 Информация о языке программирования Python [сайт] – URL: https://ru.wikipedia.org/wiki/Python (дата обращения 20.11.2021).

8 Информация о веб-фреймворке Django [сайт] – URL: https://ru.wikipedia.org/wiki/Django (дата обращения 20.11.2021).

ПРИЛОЖЕНИЕ А

Представления для генерации страниц

```
from django.shortcuts import render, redirect
from django.http import HttpResponseRedirect
from django.urls import reverse
from django.views.generic import ListView
from django.views import View
from django.contrib.auth import authenticate, login, logout
from django.contrib import messages

from .models import Post, UserSeenPosts, Tag
from .forms import CommentForm, CreateUserForm


class StartingPageView(View):
    def get(self, request):
        popular_posts = Post.objects.order_by('-views')[:3]
        new_posts = Post.objects.order_by('-date_time')[:3]
        context = {
            'popular_posts': popular_posts,
            'new_posts': new_posts
        }

        return render(request, 'blog/index.html', context)


class PostsView(ListView):
    template_name = 'blog/all-posts.html'
    model = Post
    ordering = ['-date_time']
    context_object_name = 'posts'


class PostDetailView(View):
    def is_stored_post(self, request, post_id):
        read_later_post_list = request.session.get('read_later_post_list')
        if read_later_post_list is not None:
            is_saved_for_later = post_id in read_later_post_list
        else:
            is_saved_for_later = False
```

```
        return is_saved_for_later

    def get(self, request, slug):
        post = Post.objects.get(slug=slug)
        related_posts =
Post.objects.filter(tags__in=post.tags.all()).exclude(slug=slug)[:3]

        try:
            if UserSeenPosts.objects.filter(post=post, user=request.user).exists():
                pass
            else:
                post.views += 1
                post.save()
                UserSeenPosts.objects.create(user=request.user, post=post)
        except:
            pass

        context = {
            'post': post,
            'post_tags': post.tags.all(),
            'related_posts': related_posts,
            'comment_form': CommentForm(),
            'all_comments': post.comments.all().order_by('-date_time'),
            'saved_for_later': self.is_stored_post(request, post.id)
        }

        return render(request, 'blog/post-detail.html', context)

    def post(self, request, slug):
        comment_form = CommentForm(request.POST)
        post = Post.objects.get(slug=slug)
        related_posts = Post.objects.filter(tags__in=post.tags.all())[:3]

        if comment_form.is_valid():
            comment_form.instance.user = request.user
            comment = comment_form.save(commit=False)
            comment.post = post
            comment.save()
            return HttpResponseRedirect(reverse('post-detail-page', args=[slug]))

        context = {
            'post': post,
            'post_tags': post.tags.all(),
            'related_posts': related_posts,
            'comment_form': comment_form,
```

```
            'user_name': request.user.username,
            'all_comments': post.comments.all().order_by('-date_time'),
            'saved_for_later': self.is_stored_post(request, post.id)
        }

        return render(request, 'blog/post-detail.html', context)


class ReadLaterView(View):
    def get(self, request):
        read_later_post_list = request.session.get('read_later_post_list')
        context = {}

        if read_later_post_list is None or len(read_later_post_list) == 0:
            context['posts'] = []
            context['has_posts'] = False

        else:
            posts = Post.objects.filter(id__in=read_later_post_list)
            context['posts'] = posts
            context['has_posts'] = True

        return render(request, 'blog/read-later-page.html', context)

    def post(self, request):
        read_later_post_list = request.session.get('read_later_post_list')

        if read_later_post_list is None:
            read_later_post_list = []

        post_id = int(request.POST['post_id'])
        if post_id not in read_later_post_list:
            read_later_post_list.append(post_id)
        else:
            read_later_post_list.remove(post_id)
        request.session['read_later_post_list'] = read_later_post_list

        return redirect('read-later-page')


class CategoriesView(ListView):
    template_name = 'blog/categories.html'
    model = Tag
    context_object_name = 'tags'
```

```
class CategoryView(View):
    def get(self, request, slug):
        posts = Post.objects.filter(tags__slug=slug)
        tag = Tag.objects.get(slug=slug)

        context = {
            'posts': posts,
            'tag': tag
        }

        return render(request, 'blog/category.html', context)


class SearchResultsView(View):
    template_name = 'blog/search_results.html'
    context = {}

    def get(self, request):
        query = request.GET.get('q')
        if query is not None:
            posts = Post.objects.filter(title__icontains=query)
            context = {
                'posts': posts
            }

        return render(request, 'blog/search_results.html', context)


class SignIn(View):
    def get(self, request):
        if request.user.is_authenticated:
            return redirect('starting-page')
        context = {}
        return render(request, 'blog/login.html', context)

    def post(self, request):
        username = request.POST.get('username')
        password = request.POST.get('password2')

        user = authenticate(request, username=username, password=password)

        if user is not None:
            login(request, user)
            return redirect('starting-page')
```

```
        else:
            messages.info(request, 'Username OR password is incorrect')

        context = {}
        return render(request, 'blog/login.html', context)


class SignUp(View):
    def get(self, request):
        if request.user.is_authenticated:
            return redirect('starting-page')
        context = {'form': CreateUserForm()}
        return render(request, 'blog/register.html', context)

    def post(self, request):
        username = request.POST.get('username')
        email = request.POST.get('email')
        password1 = request.POST.get('password1')
        password2 = request.POST.get('password2')
        form = CreateUserForm({
            'username': username,
            'email': email,
            'password1': password1,
            'password2': password2})
        if form.is_valid():
            form.save()
            user = form.cleaned_data.get('username')
            messages.success(request, 'Account was created for ' + user)

            return redirect('login')

        context = {'form': form}
        return render(request, 'blog/register.html', context)


class LogoutUser(View):
    def get(self, request):
        logout(request)
        return redirect('starting-page')
```

ПРИЛОЖЕНИЕ Б

Модели базы данных сайта

```
from django.contrib.auth.models import User
from django.db import models
from django.core.validators import MinLengthValidator


class Author(models.Model):
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    email = models.EmailField(max_length=150)

    def full_name(self):
        return f'{self.first_name} {self.last_name}'

    def __str__(self):
        return self.full_name()


class Tag(models.Model):
    caption = models.CharField(max_length=10)
    slug = models.SlugField(unique=True, null=True)

    def __str__(self):
        return self.caption


class Post(models.Model):
    title = models.CharField(max_length=20)
    views = models.IntegerField(default=0)
    excerpt = models.CharField(max_length=100)
    image = models.ImageField(upload_to='posts', null=True)
    date_time = models.DateTimeField(auto_now_add=True, null=True)
    slug = models.SlugField(unique=True)
    content = models.TextField(validators=[MinLengthValidator(10)])
    author = models.ForeignKey(Author, on_delete=models.SET_NULL,
null=True, related_name="posts")
    tags = models.ManyToManyField(Tag)

    def __str__(self):
        return f'{self.title}, {self.author}, {self.date_time}'
```

```
class Comment(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, null=True,
editable=False)
    user_text = models.TextField(max_length=500)
    post = models.ForeignKey(Post, on_delete=models.CASCADE,
related_name='comments')
    date_time = models.DateTimeField(auto_now_add=True, null=True)

    def __str__(self):
        return 'Comment {} by {}'.format(self.user_text, self.user)


class UserSeenPosts(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='seen_posts')
    post = models.ForeignKey(Post, on_delete=models.CASCADE)

    def __str__(self):
        return f'{self.user}, {self.post}'
```