Лабораторная работа № 5

Создание формы и представления для нового поста

В директорию **lab5** скопируйте ваш прошлый проект **blog**. В файле **urls.py** создайте ещё один адрес: article/new/, который будет обрабатываться функцией-представлением create_post. Создайте в файле представлений наряду с функциями archive и get_article новую функцию create_post (на которую и ссылается только что написанный url). Эта функция должна будет при первом обращении к странице просто возвращать html-код с формой создания новой записи (html-шаблон для этого вы напишите чуть позже), а в случае, если к функции пользователь обратился во второй и следующий разы, при этом заполнив форму с названием и текстом нового поста, представление должно сохранять отправленные через форму данные в базу данных. Иными словами, сейчас вам необходимо создать представление, которое выполняет ту же роль, что и страница добавления статьи в административном интерфейса (эта страница при запущенном проекте должна быть доступна по ссылке http://localhost:8000/admin/articles/article/add/).

Сначала в представлении проверьте, **авторизован ли пользователь на сайте**, потому что у каждого поста есть поле «**автор**», которое не может быть корректно установлено, если автор не авторизован. Соответственно, всю работу необходимо проводить, только если данная проверка была успешно пройдена:

```
def create_post(request):
     if not request.user.is_anonymous():
          # Здесь будет основной код представления
     else:
          raise Http404
```

Если пользователь авторизован, то сначала определитесь, что программе необходимо сделать на данный момент: просто вернуть страницу с формой или пользователь в свое время уже получил форму, заполнил её и сейчас от программы требуется сохранить новую запись. Для этого нужно знать метод, с помощью которого отправлен запрос, и если это метод POST, то необходимо проверить полученные данные (если не заполнено одно из полей, вернуть

обратно страницу с формой и уведомить об ошибке) и, если проверка пройдена, создать в базе данных новую запись.

```
if request.method == "POST":
        # обработать данные формы, если метод POST
            form = {
                'text': request.POST["text"], 'title':
request.POST["title"]
            }
        # в словаре form будет храниться информация, введенная
пользователем
            if form["text"] and form["title"]:
        # если поля заполнены без ошибок
                Article.objects.create(text=form["text"],
title=form["title"], author=request.user)
                return redirect('get_article',
article_id=article.id)
            # перейти на страницу поста
            else:
        # если введенные данные некорректны
                form['errors'] = u"Не все поля заполнены"
                return render(request, 'create_post.html',
{'form': form})
        else:
        # просто вернуть страницу с формой, если метод GET
            return render(request, 'create_post.html', {})
```

Для того, чтобы перейти на главную страницу, в Django из представления можно вернуть результат функции redirect(), которой в качестве параметра достаточно передать имя url (имя для каждого url определяется в файле urls.py в параметре name) и указать, как заполнить именованные группы в регулярном выражении. Теперь перейдем к реализации шаблона с формой:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Django Public Blog - создать статью</title>
    </head>
    <body>
        <div class="content">
            <h1>Написать статью</h1>
            <form method="POST">{% csrf_token %}
<input type="text" name="title" placeholder="Название статьи"
  value="{{ form.title }}">
<textarea name="text" placeholder="Текст
  статьи">{{ form.text }}</textarea>
                <input type="submit" value="Сохранить">
            </form>
            {{ form.errors }}
```

```
        </div>
    </body>
</html>
```

Тег <form> используется для объединения нескольких полей ввода, чтобы при отправлении, в запрос добавить все данные, которые ввел пользователь. Это особенно важно, когда на одной странице присутствуют сразу несколько полей ввода, которые нужно отправлять независимо (например, если на странице сразу присутствуют и поля авторизации (поля логина и пароля) и поля, например, добавления комментария). Если пользователь оставляет комментарий, на сервер не нужно отправлять данные об авторизации и, наоборот, если пользователь входит в свою учетную запись, не нужно отправлять содержимое поля «комментарий».

Внутри тега <form> находится специальный шаблонный тег `{% csrf_token %}`, который добавляет специальное скрытое поле в форму, это поле легко увидеть, если посмотреть содержимое формы через консоль отладки. Затем следуют два главных элемента: <input> (для ввода названия статьи, всегда имеет однострочную форму) и <textarea> (для ввода самой записи, тег позволяет вводить текст на нескольких строках). Значение по умолчанию для тега <input> необходимо указывать в атрибуте value, а для тега <textarea> достаточно написать любой текст между открывающим тегом и закрывающим. Также в форму добавлена кнопка для отправки данных на сервер, а после перечисления всех полей указываются все допущенные пользователем ошибки (если эти ошибки имели место быть).

Задание:

- Создайте несколько статей через созданную форму;
- Создайте стили, подключив CSS-файл к шаблону;
- Добавьте проверку на то, что введенное название статьи уникально;
- Загрузите ваш проект на любой гит-репозиторий (GitHub, GitLab, Google Code, Bitbucket и т.п.).