Министерство цифрового развития, связи и массовых коммуникаций
Российской Федерации
Ордена Трудового Красного Знамени
федеральное государственное бюджетное
образовательное учреждение высшего образования
«Московский технический университет связи и информатики»

Кафедра МКиИТ

Проектирование клиент-серверных приложений

Лабораторная работа №6
“Создание формы, шаблона и представления для регистрации и для авторизации”

Выполнил:
студент 3 курса,
группы БФИ2001
Лушин Е. А.

Москва 2023

Цель работы: научиться создавать формы и представления для регистрации и авторизации.

Задание:

Создание формы, шаблона и представления для авторизации

- Создайте шаблон и настройте адрес для отображения страницы регистрации.
- Создайте представление, которое обрабатывает поступающие запросы и регистрирует новых пользователей. Не забудьте сделать проверку на то, что отправленные поля непусты, а введенное имя пользователя уникально.
- Сверстайте страницу в соответствии с макетом **lab6_registration_form**.
- Добавьте в шапку страниц всех записей и страниц для определенных статей ссылку на регистрацию в верхнем правом углу (стиль ссылке сделать точно такой же, как и ссылки “**Все статьи**” на собственных страницах постов)

Создание формы, шаблона и представления для авторизации

- Создайте шаблон и настройте адрес для отображения страницы авторизации.
- Создайте представление, которое обрабатывает поступающие запросы и авторизует пользователей. Не забудьте сделать проверку на то, что отправленные поля непусты, логин и пароль соответствуют одному из аккаунтов вашего проекта.
- Сверстайте страницу в соответствии с макетом **lab6_authorization_form**.

Краткая теория

Система авторизации и регистрации в Django

Протокол HTTP спроектирован так, что не сохраняет информацию о состоянии соединения, то есть все запросы независимы друг от друга. Между предыдущим и следующим запросом нет никакой связи и не существует такого свойства запроса (IP-адрес, агент пользователя и т. п.), которое позволило бы надежно идентифицировать цепочку последовательных запросов от одного и того же лица.

Разработчики браузеров уже давно поняли, что отсутствие информации о состоянии в протоколе HTTP ставит серьезную проблему перед веб-программистами.

Поэтому на свет появились cookies. Cookie — это небольшой блок информации, который отправляется веб-сервером и сохраняется браузером. Запрашивая любую страницу с некоторого сервера, браузер посылает ему блок информации, который получил от него ранее.

Посмотрите, как действует этот механизм. Когда вы открываете браузер и вводите в адресной строке google.com, браузер посылает серверу Google HTTP-запрос, который начинается так (пример запроса на страницу google.com/):

GET / HTTP/1.1

Host: google.com

. . .

Полученный от Google ответ выглядит приблизительно так (пример ответа на запрос страницы по адресу google.com/):

HTTP/1.1 200 OK

Content-Type: text/html

Set-Cookie:

PREF=ID=5b14f22bdaf1e81c:TM=1167000671:LM=1167000671;

expires=Sun, 17-Jan-2038 19:14:07 GMT;

path=/; domain=.google.com

Server: GWS/2.1

. . .

Обратите внимание на заголовок Set-Cookie. Браузер сохранит значение cookie (PREF=ID=5b14f22bdaf1e81c:TM=1167000671: LM=1167000671) и будет отправлять его Google при каждом обращении к этому сайту. Поэтому при следующем посещении сайта Google запрос, отправленный браузером, будет иметь такой вид (пример следующего запроса на страницу google.com/):

GET / HTTP/1.1

Host: google.com

Cookie:

PREF=ID=5b14f22bdaf1e81c:TM=1167000671:LM=1167000671

. . .

Обнаружив заголовок Cookie, Google понимает, что запрос пришел от человека, уже посещавшего сайт. Значением cookie может быть, например, ключ в таблице базы данных, где хранятся сведения о пользователе. И Google мог бы (да, собственно, так и делает) отобразить на странице имя вашей учетной записи.

Соответственно, если вы пожелаете создать систему авторизации и регистрации пользователей на своем сайте, вам придется с помощью cookie в браузере клиента хранить данные о том, что текущий пользователь — это тот человек, который регистрировался на сайте полгода назад под ником «Vasya». Так что первое пришедшее решение «в лоб»: когда юзер входит в аккаунт (введя логин и пароль), можно просто ему ставить cookie «username=vasya;» и cookie будет прилеплена ко всему запросу. От вас потребуется лишь каждый раз читать значение cookie и радоваться, что такая система авторизации работает.

Однако тут есть маленькая проблема безопасности: любой злоумышленник сможет вручную установить у себя в браузере cookie «username=vasya;» и притвориться Васей! Он сможет читать его сообщения и рассылать спам от его имени! Естественно, это довольно легко решается, чуть-чуть усложнив систему на стороне сервера. Надо лишь устанавливать в куках не значение никнейма пользователя, а лишь какой-то бессмысленный для посторонних идентификатор, символов этак в 20. Тогда вам при каждом запросе достаточно всего лишь на сервере этот уникальный идентификатор проверять, в базе данных отыскивать, какому пользователю соответствует переданный id. В случае такой системы авторизации, чтобы взломать Васю, придется угадать именно тот айди, который сейчас установлен у аккаунта Васи

(к тому же чаще всего id будет меняться время от времени). Такой подход к авторизации называется сессиями, а самому идентификатору внутри cookies присваивается имя наподобие session id.

В принципе, чтобы это всю подобную систему проверить в действии, достаточно сначала зайти в административной панели Django в свой аккаунт, затем на вкладке Resourses панели разработчиков Chrome найти установленные cookies. Среди них как раз должна находиться cookies с именем sessionid. Она-то при каждом, даже пустом, запросе передается серверу, считывается и определяет пользователя, который этому идентификатор соответствует. Ниже на рисунке 1 показан скриншот, который иллюстрирует, как в браузере найти установленные cookies:

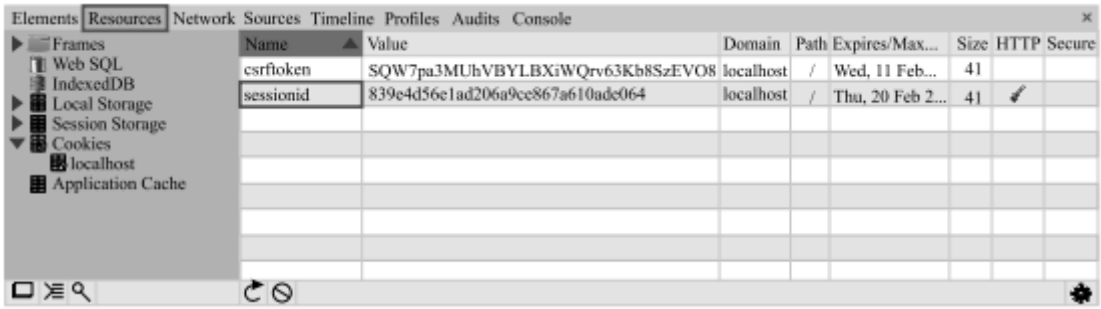

Рисунок 1 – Отображение cookies в панели разработчика Google Chrome

Для непосредственной работы с системой авторизации/регистрации пользователей в Django существует написанное создателями фреймворка приложение django.contrib.auth. В нем хранятся все необходимые функции и модели для создания, редактирования пользователей, а также функции для реализации входа в аккаунт и выхода из аккаунта.

Выполнение

За основу был взят проект из лабораторной работы №5. Добавил в файл urls.py новые пути.

Листинг 1. Содержимое файла urls.py

```
blog > blog > urls.py > ...
  1  from articles import views
  2  from django.contrib import admin
  3  from django.urls import path
  4
  5  urlpatterns = [
  6      path('admin/', admin.site.urls),
  7      path('', views.archive, name= 'home'),
  8      path('article/<int:article_id>', views.get_article, name= 'get_article'),
  9      path('article/new/', views.create_post, name= 'create_post'),
 10      path('registration', views.registred, name= 'regPage'),
 11      path('logIn', views.logIn, name= 'logIn'),
 12      path('logOut', views.logOut, name= 'logOut'),
 13  ]
```

Создадим в файле views.py новую функцию registred, которая должна будет отображать форму с регистрацией пользователя, и переводить его на форму авторизации в случае успешной регистрации.

Листинг 2. Содержимое файла views.py

```
blog > articles > views.py > ...
  1  from .models import Article
  2  from django.shortcuts import render, redirect
  3
  4  from django.http import Http404
  5
  6  # Create your views here.
  7
  8  def archive(request):
  9      return render(request, 'archive.html', {"posts": Article.objects.all()})
 10
 11  def get_article(request, article_id):
 12      try:
 13          post = Article.objects.get(id=article_id)
 14          return render(request, 'article.html', {"post": post})
 15      except Article.DoesNotExist:
 16          raise Http404
 17
 18  def create_post(request):
 19      if request.user.is_authenticated:
 20          if request.method == "POST":
 21          # обработать данные форны, если метод POST
 22              form = {
 23                  'text': request.POST["text"], 'title': request.POST["title"]
 24              }
 25              # в словаре form будет храниться информация, введенная пользователен
 26              if form["text"] and form["title"]:
 27                  # если поля заполнены без ошибок
 28                  if not Article.objects.filter(title=form['title']).exists():
 29                      Article.objects.create(text=form["text"], title=form["title"], author=request.user)
 30                      return redirect('get_article', article_id=Article.objects.count())
```

```
                else:
                    form['errors'] = u"Статья с таким названием уже существует"
                    return render(request, 'create_post.html', {'form': form})
            # перейти на страницу поста
            else:
                # если введённые данные некорректны
                form['errors'] = u"Не все поля запонены"
                return render(request, 'create_post.html', {'form': form})
        else:
        #просто вернуть страницу с формой, если метод GET
            return render(request, 'create_post.html', {})
    else:
        raise Http404

def registred(request):
    if request.method == "POST":
        form = {
            'username': request.POST["username"],
            'email': request.POST["email"],
            'password': request.POST["password"]
        }
        if form["username"] and form['email'] and form['password']:
            try:
                User.objects.get(username=request.POST["username"])
                form['errors'] = u"Пользователь с таким именем уже существует"
                return render(request, 'registrationpage.html', {'form': form})
            except User.DoesNotExist:
                User.objects.create_user(username=request.POST["username"],
                                         email=request.POST["username"],
                                         password=request.POST['password'])
                return redirect('logIn')
        else:
            form['errors'] = u"Не все поля заполнены"
            return render(request, 'registrationpage.html', {'form': form})
    else:
        return render(request, 'registrationpage.html', {})
```

Далее создал файл registrationpage.html.

Листинг 3. Содержимое файла registrationpage.html

```
blog > articles > templates > <> registrationpage.html > ...
{% load static %}

<!DOCTYPE html>
<html>

<head>
    <title>Регистрация</title>
    <link rel="stylesheet" type="text/css" href="{% static 'css/article.css' %}">
</head>

<body>
    <div class="content">
        <h1>Страница регистрации</h1>
        <form method="POST">{% csrf_token %}
            <input type="text" name="username" placeholder="Имя пользователя" value="{{form.title}}" />
            <input type="password" name="password" placeholder="Пароль" value="{{form.title}}" />
            <input type="submit" value="Сохранить" />
        </form>
        {{form.errors}}
    </div>
</body>

</html>
```

Затем добавил функцию logIn, которая будет выполнять авторизацию пользователя. Кусок кода продемонстрирован на рисунке 2 на странице 8.

```python
def logIn(request):
    if request.method == "POST":
        form = {
            'username': request.POST["username"],
            'password': request.POST["password"]
        }
        if form["username"] and form['password']:
            user = authenticate(request, username=request.POST["username"], password=request.POST["password"])
            if user is not None:
                login(request, user)
                return redirect('home')
            else:
                form['errors'] = u"Введённый пользователь не существует"
                return render(request, 'singInPage.html', {'form': form})
        else:
            form['errors'] = u"Не все поля заполнены"
            return render(request, 'singInPage.html', {'form': form})
    else:
        return render(request, 'singInPage.html', {})

def logoutFunc(request):
    logout(request)
    return redirect('home')
```

Рисунок 2 – Функция logIn

Далее создал файл signinpage.html.

Листинг 4. Содержимое файла signinpage.html

```
blog > articles > templates > <> signinpage.html > ...
{% load static %}

<!DOCTYPE html>
<html>

<head>
    <title>Авторизация</title>
    <link rel="stylesheet" type="text/css" href="{% static 'css/article.css' %}">
</head>

<body>
    <div class="content">
        <h1>Страница Авторизации</h1>
        <form method="POST">{% csrf_token %}
            <input type="text" name="username" placeholder="Имя пользователя" value="{{form.title}}" />
            <input type="password" name="password" placeholder="Пароль" value="{{form.title}}" />
            <input type="submit" value="Сохранить" />
        </form>
        {{form.errors}}
    </div>
</body>

</html>
```

Ниже представлен программный код файла article.css со стилями для страниц.

Листинг 5. Содержимое файла article.css

```css
body {
    background: #1abc9c;
    font-family: Tahoma, Arial, sans-serif;
    color: #ffffff
}
```

```css
img {
    display: block;
    width: 318px;
    margin-left: auto;
    margin-right: auto;
}

.archive {
    width: 960px;
    margin-left: auto;
    margin-right: auto;
}

post-title a {
    color: #ffffff;
}

.article-author {
    width: 50%;
    float: Left;
}

.article-created-date {
    text-align: right;
}

.article-image {
    display: block;
    width: 318px;
    margin-left: 0;
}

.link {
    color: white;
    font-weight: bold;
    position: absolute;
    right: 470px;
    top: 180px;
}

.article-border p {
    text-align: right;
}

.article-text {
    width: 960px;
    text-align: justify;
}

.article-created-data {
    text-align: right;
```

```css
}

.content {
    text-align: center;
    padding-top: 70px;
}

input[name="title"] {
    padding: 5px;
    margin-bottom: 10px;
    border: 1px solid #888;
    outline: none;
    -moz-appearance: none;
    width: 200px;
    text-align: center;
    border-radius: 40px;
}

textarea[name="text"] {
    padding: 25px;
    margin-bottom: 10px;
    border: 1px solid #888;
    outline: none;
    -moz-appearance: none;
    width: 650px;
    height: 350px;
    resize: none;
    border-radius: 40px;
    scrollbar-width: thin;
}

.create {
    display: flex;
    flex-direction: column;
    align-items: center;
}

.save_button {
    padding: 10px;
    width: 150px;
    background-color: white;
    border: none;
    border-radius: 40px;
    color: #1abc9c;
    font-weight: bold;
    letter-spacing: 0.06em;
    margin-top: 10px;
}

.save_button:hover {
    color: white;
```

```
    background-color: #1abc9c;
    box-shadow: 1px 1px 10px 10px;
    transition-duration: 0.3s;
}
```

На рисунках 3-4 представлены страницы авторизации и регистрации/

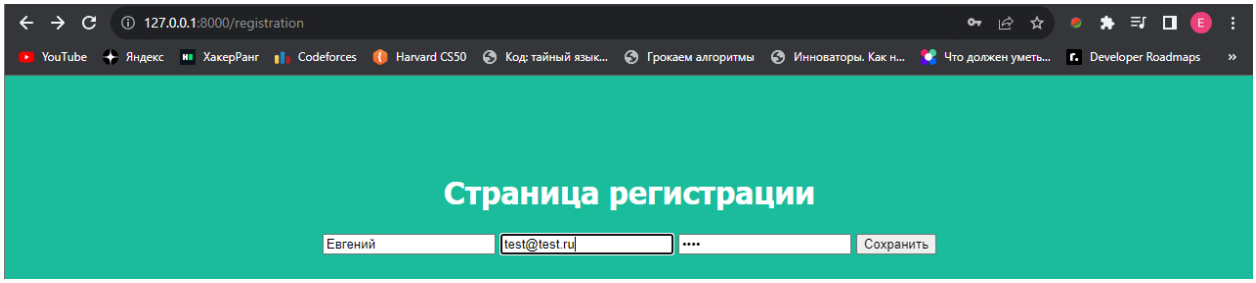

Рисунок 3 – Страница регистрации

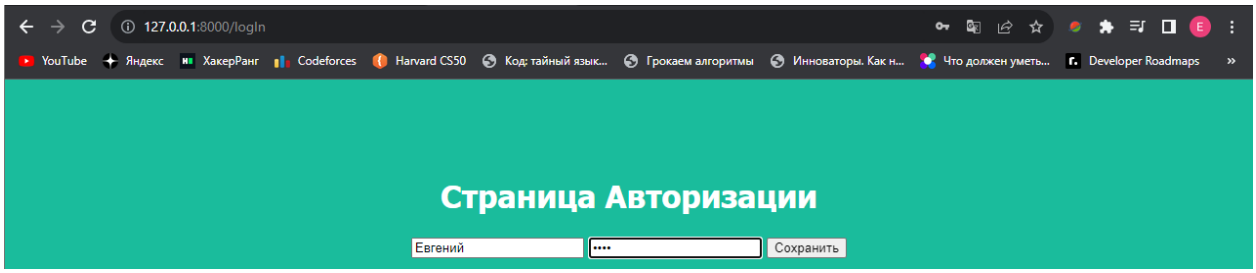

Рисунок 4 – Страница авторизации

Вывод: в данной лабораторной работе я научился создавать страницу регистрации и входа.