ТЕХНОАТОМ

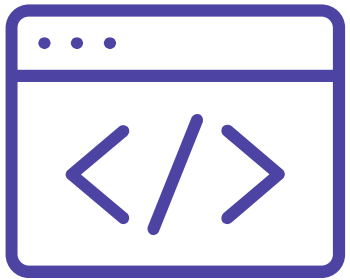
Веб-программирование Python

Лекция 4. Пользователи

Михалев Олег

Сегодня

- Аутентификация
- Cookie
- Сессии
- Пользователи



Однопользовательский режим работы подходит для десктнопных приложений, но совершенно нежизнеспособен в веб-среде

Всемирная паутина по определению распределена и современные веб-приложения обязаны поддерживать многопользовательский режим

Немного о терминологии

Идентификация

Процедура получения
пользователя по идентификатору

Идентификация

Идентификатор пользователя (логин) -> Пользователь
Идентификатор сеанса -> Пользователь

Аутентификация

Процедура проверки подлинности пользователя

Аутентификация

Пользователь, ключ пользователя -> Истина/Ложь

Авторизация

Процедура проверки
прав пользователя

Авторизация

Пользователь, действие -> Истина/Ложь

Вы посещаете закрытую вечеринку, охранник просит показать документы:

- охранник проверяет фотографию в документах на соответствие с оригиналом;
- охранник получает информацию о вашем имени;
- охранник проверяет наличие вашего имени в списке;

ТЕХНОАТОМ

Аутентификация
Идентификация
Авторизация

Вы посещаете бар, охранник просит показать документы :

- охранник проверяет фотографию в документах на соответствие с оригиналом;
- охранник проверяет возраст в документах на соответствие критериям;

Аутентификация
Авторизация

Неявно проходит и идентификация

В многопользовательском веб-приложении описанные процессы встраиваются во все компоненты, взаимодействующие с пользователем

Пользователь совершает первое обращение

Идентификация

Приложение проверяет наличие сеанса пользователя

Приложение создает анонимный сеанс

Пользователь переходит на страницу аутентификации
Пользователь отправляет данные

Аутентификация
Приложение проверяет введенные данные пользователя

Приложение обновляет сеанс

Пользователь совершает обращение к ресурсу

Идентификация

Приложение определяет пользователя из данных сеанса

Авторизация

Приложение определяет права доступа к ресурсу

От абстракции к реальности

Сеанс - это последовательность действий пользователя

Сеанс как любая отдельная сущность может характеризоваться идентификатором

Создание сеанса

- приложение создает идентификатор сеанса
- приложение передает идентификатор сеанса браузеру

Обращение

- браузер передает идентификатор сеанса приложению

Технологии передачи идентификатора сеанса

- URL
- HTTP-заголовки
- Cookie

Добро пожаловать в мир веб-программирования

У нас есть печеньки!

Данные cookie создаются и изменяются веб-приложением
Данные cookie хранятся веб-браузером

Данные cookie передаются при каждом обращении

По своей сути cookie - это все те же HTTP-заголовки

Приложение устанавливает cookie в заголовках ответа

```
1....
2.HTTP/1.1 200 OK
3.Content-Type: text/html
4....
5.Set-Cookie: SessionID=42
6.Set-Cookie: LanguageCode=RU
7...
```

Веб-браузер передает cookie в заголовках запроса

```
1.GET / HTTP/1.1
2.Host: localhost
3....
4.Cookie: SessionID=42; LanguageCode=RU
5....
```

Cookie можно представлять как хранилище "ключ-значение", для их чтения в Django можно воспользоваться обычным словарем

```
1.def language_view(request):
2.    if request.COOKIES.get('LanguageCode') == 'RU':
3.        return HttpResponse('Russian')
4.    else:
5.        return HttpResponse('English (default)')
```

Важно помнить

- изменения словаря **request.COOKIES** не изменят сами cookie
- все ключи/значения словаря **request.COOKIES** являются строками

Для изменения cookie у экземпляра класса **HttpResponse** имеются методы **set_cookie** и **delete_cookie**

```
1.def language_change(request, code):
2.    response = redirect('/')
3.    response.set_cookie('LanguageCode', code)
4.    return response
```

Cookie имеют также атрибуты для более тонкой настройки

Время жизни

max_age - срок жизни cookie в секундах
expires - дата истечения срока жизни

Область видимости

domain - доменное имя веб-сервера
path - путь к ресурсу

Флаги протокола

secure - передача только по HTTPS
httponly - передача только по HTTP

Важно помнить

- размер cookie ограничен 4096 байтами (включая атрибуты)

```
1. def example(request):
2.     session = get_session(request)
3.     ...
4.     response = redirect('/')
5.     response.set_cookie('SessionID', session.id, max_age=300, secure=True)
6.     return response
```

```
1.HTTP/1.1 302 Found
2.Set-Cookie: SessionID=42; Max-Age=300; Secure
3.Location: https://localhost/
4.....
```

Как вы могли заметить, cookie применяются
не только для передачи идентификатора сеанса

Однако в силу ограничений по размеру и
исходя из соображений безопасности в cookie можно
хранить далеко не все данные

Очевидно, что мы можем создать хранилище на стороне веб-приложения, доступное для обращений по идентификатору сеанса

Django предоставляет готовые компоненты сеанса - **django.contrib.sessions**

Для включения компонента пользовательских сессий необходимо сконфигурировать **setting.py**, добавив **django.contrib.sessions** в **INSTALLED_APPS**, а **django.contrib.sessions.middleware.SessionMiddleware** в подключенные **MIDDLEWARE**

Данные сессии могут хранится в разнообразных местах (параметр **SESSION_ENGINE**):

- **django.contrib.sessions.backends.db** - СУБД (по умолчанию)
- **django.contrib.sessions.backends.cache** - Кэш
- **django.contrib.sessions.backends.file** - Файлы

В случае использования кэша мы можем гибко настроить и его (параметр **CACHES**), используя разнообразные реализации
из **django.core.cache.backends**
(например, memcache и кэш веб-приложения)
или сторонние реализации
(например, redis и tarantool)

Работа с сессией похожа на работу с cookie, однако готовые компоненты значительно ее упрощают (cookie выставляется автоматически, экземпляры запроса и ответа связаны общим контекстом)

```
1.def login_view(request):
2.    user = get_user(request)
3.    ...
4.    request.session['user_id'] = user.id
5.    request.session['user_name'] = user.name
6.    return redirect('/')
```

Благодаря тому, что экземпляр **HttpRequest** пробрасывается в шаблоны, мы можем использовать данные сессии при рендеринге страницы

```
1.<div class="profile-link">
2.    Hello,
3.    <a href="{% url 'profile' request.session.user_id %}">
4.        {{ request.session.user_name }}
5.    </a>
6.</div>
```

Важно понимать, что cookie и сессии привязаны к веб-браузеру и ограничены сроком жизни

Для полноценной работы с пользователями Django предоставляет готовые компоненты **django.contrib.auth**

Для включения пользовательских компонент необходимо сконфигурировать **setting.py**

INSTALLED_APPS
django.contrib.sessions
django.contrib.auth
django.contrib.contenttypes

MIDDLEWARE
django.contrib.sessions.middleware.SessionMiddleware
django.contrib.auth.middleware.AuthenticationMiddleware

Компонент **django.contrib.auth** предоставляет три базовые сущности

- Модель **User** - Пользователь
- Модель **Group** - Группа пользователей
- Модель **Permission** - Права доступа

Модель **User** - ядро подсистемы аутентификации Django, основные его поля - **username**, **email** и **password**

```
1.>>> from django.contrib.auth.models import User
2.>>> user = User.objects.create_user('sparrow',
  'sparrow@mail.ru', 'secret')
```

Важно помнить, что пароли могут не хранится в открытом виде

```
1.>>> from django.contrib.auth.models import User
2.>>> user = User.objects.get(username='sparrow')
3.>>> user.set_password('top_secret')
4.>>> user.save()
```

Django Auth, Passwords management
https://docs.djangoproject.com/en/1.10/topics/auth/passwords/

Базовую модель **User** можно переопределить, не забыв при этом изменить конфигурацию **AUTH_USER_MODEL** в **settings.py**

```
1. ...
2. AUTH_USER_MODEL = 'finances.User'
3. ...
```

Пользователи

ТЕХНОАТОМ

```
1. from django.db import models
2. from django.conf.auth.models import AbstractUser

3. class User(AbstractUser):
4.     phone_number = models.CharField(max_length=16)
```

```
1. from django.db import models
2. from django.conf import settings

3. class Account(models.Model):
4.     user = models.ForeignKey(settings.AUTH_USER_MODEL, related_name='accounts')
```

Альтернативой расширению стандартной модели **User** является дополнение ее отдельными моделями

```
1. from django.db import models
2. from django.conf.auth.models import User

3. class UserProfile(models.Model):
4.     phone_number = models.CharField(max_length=16)
5.     user = models.OneToOneField(User, related_name='profile')
```

Как и в случае сессий auth-middleware дополняет экземпляр **HttpRequest**

```
1. from django.core.exceptions import PermissionDenied

2. def example(request):
3.     if request.user.is_authenticated:
4.         ..
5.     else:
6.         raise PermissionDenied
```

Рассмотрим простую аутентификацию

```
1. from django.shortcuts import render, redirect
2. from django.contrib.auth import authenticate, login, logout
3. from django.contrib.messages import error

4. ...
```

Пользователи

ТЕХНОАТОМ

```
1. def login_view(request):
2.     username = request.POST.get('username')
3.     password = request.POST.get('password')
4.     if not (username and password):
5.         return render(request, 'login.html')
6.     user = authenticate(username=username, password=password)
7.     if not user:
8.         error('Wrong credentials!')
9.         return render(request, 'login.html')
10.    login(request, user)
11.    return redirect('/')
```

Пользователи

ТЕХНОАТОМ

```
1. def logout_view(request):
2.     if request.method == 'POST':
3.         logout(request)
4.     return redirect('/')
```

Для определения факта прохождения пользователем аутентификации удобно использовать декоратор

```
1. from django.contrib.auth.decorators import login_required
2. from django.shortcuts import render

3. @login_required
4. def profile_view(request):
5.     context = {'profile': request.user.profile}
6.     return render(request, 'profile.html', context)
```

Параметр **LOGIN_URL** в **setting.py**
позволяет указать, куда будет перенаправлен
пользователь для прохождения аутентификации

Django может использовать собственные формы и представления для аутентификации

```
1. urlpatterns = [
2.     url('^', include('django.contrib.auth.urls')),
3. ]
```

Аутентифицировались, что дальше?

Модели **Group** и **Permission** - ядро подсистемы авторизации Django

Они связаны также с моделью **User**

Группы объединяют пользователей с одинаковыми правами

```
1. from django.conf.auth.models import Group
2. from django.core.exceptions import PermissionDenied

3. @login_required
4. def news_edit(request):
5.     editors_group = Group.objects.get(name='editors')
6.     if editors_group not in request.user.groups:
7.         raise PermissionDenied
8.     ...
```

Для добавления/удаления групп используется экземпляр **User**

```
1. def group_add(request, group_name):
2.     group = get_object_or_404(Group, name=group_name)
3.     request.user.groups.add(group)
4.     return redirect('/')

5. def group_remove(request, group_name):
6.     group = get_object_or_404(Group, name=group_name)
7.     request.user.groups.remove(group)
8.     return redirect('/')
```

Права доступа могут быть определены в мета-данных моделей

```
1. class Account(models.Model):

2.     number = models.CharField(max_length=20)
3.     user = models.ForeignKey(User, related_name='account')

4.     class Meta:
5.         permissions = (
6.             ('can_view_account', 'Can view account'),
7.             ('can_edit_account', 'Can edit account₩'s data'),
8.         )
```

Или созданы вручную

```
1. from finances.models import Account
2. from django.contrib.contenttypes.models import ContentType
3. from django.contrib.auth.models import Permission

4. content_type = ContentType.objects.get_for_model(Account)
5. permission = Permission.objects.create(
6.     codename='can_edit_account',
7.     name='Can edit account₩'s data',
8.     content_type=content_type
9. )
```

Права могут быть разрешены на группу или конкретного пользователя

```
1. def grant_for_group(request):
2.     perm = Permission.objects.get(codename='can_edit_account')
3.     editors_group = Group.objects.get(name='editors')
4.     editors_group.permissions.add(perm)
5.     return redirect('/')

6. def grant_for_user(request):
7.     perm = Permission.objects.get(codename='can_edit_account')
8.     request.user.user_permissions.add(perm)
9.     return redirect('/')
```

Проверить права доступа можно методом **has_perm** на экземпляре **User**

```
1.@login_required
2.def account_edit(request):
3.    if not request.user.has_perm('can_edit_account'):
4.        raise PermissionDenied
5.    ...
```

По традиции, удобнее использовать декораторы

```
1.from django.contrib.auth.decorators import (
2.    login_required, permission_required
3.)

4.@login_required
5.@permission_required('can_edit_account')
6.def account_edit(request):
7.    ...
```

ТЕХНОАТОМ

Спасибо за внимание!

Михалев Олег
mailto:mhalairt@gmail.com

Домашнее задание

ТЕХНОАТОМ

Подключить компоненты **django.contrib.auth** к проекту. Реализовать представления входа в систему (аутентификации) и выхода из нее (для одиноких разработчиков возможно подключение готовых представлений Django, командам необходимо реализовать свои). Реализовать модель "Профиль пользователя" **UserProfile** с полями "Номер телефона" и "Адрес" (командам необходимо расширить абстрактную модель **AbstractUser**) и реализовать страницу просмотра профиля для пользователей, прошедших аутентификацию.

Домашнее задание

ТЕХНОАТОМ

Реализовать связь один-ко-многим сущностей **User** и **Account**. В представлении "Список счетов" выводить только счета, принадлежащие текущему пользователю. Доступ к счетам и их денежным транзакциям для неаутентифицированных пользователей или пользователям, которым счет не принадлежит, должен быть закрыт.

Полезные ссылки

ТЕХНОАТОМ

Django Sessions
https://docs.djangoproject.com/en/1.10/topics/http/sessions/

Django Auth
https://docs.djangoproject.com/en/1.10/topics/auth/
https://docs.djangoproject.com/en/1.10/ref/contrib/auth/

Django Auth, Customizing
https://docs.djangoproject.com/en/1.10/topics/auth/customizing/