РАЗРАБОТКА ПРОГРАММНЫХ ПРИЛОЖЕНИЙ

Лекция 2. Аутентификация, авторизация и регистрация пользователей

- Введение
- Основы системы пользователей Django
- Приложение авторизации Django

ВВЕДЕНИЕ

Аутентификация и авторизация – это два ключевых термина, связанных с управлением доступом пользователей к ресурсам веб-приложений. Оба понятия имеют существенное значение для безопасности приложений и управления конфиденциальностью информации.

Аутентификация

Аутентификация — это процесс подтверждения личности пользователя, который пытается войти в систему. В Django это реализовано через встроенную модель пользователя (User) и формы аутентификации, такие как AuthenticationForm. При вводе пользователем логина и пароля Django сравнивает введенные данные с хранимыми в базе данных, где пароли надежно хэшируются с использованием алгоритмов вроде PBKDF2 или bcrypt.

Кроме классической аутентификации, Django поддерживает и более сложные схемы, например, аутентификацию через сторонние сервисы (OAuth, OpenID Connect), что упрощает интеграцию с такими платформами, как Google, GitHub или Facebook.

ВВЕДЕНИЕ

Авторизация

После успешной аутентификации наступает этап авторизации.

Авторизация – это процесс определения, какие действия участники могут выполнять после прохождения аутентификации. Django предоставляет гибкую систему разрешений, которая позволяет определять, какие действия пользователь может выполнять в системе.

Каждому пользователю или группе пользователей можно назначить конкретные права: от базового доступа до определённых страниц до сложных правил, которые управляют действиями в админке или доступом к данным. Группы и права легко управляются через Django Admin, что делает процесс настройки ролей простым и гибким.

ВВЕДЕНИЕ

Существует несколько видов аутентификации, которые могут использоваться в бекенд приложениях:

- ✓ **Базовая аутентификация**: это наиболее простой метод аутентификации, который использует логин и пароль пользователя для доступа к ресурсам. Эта информация передается в заголовках запроса HTTP, закодированная в формате Base64. Однако этот метод не считается безопасным, поскольку логин и пароль могут быть перехвачены на пути между клиентом и сервером.

ВВЕДЕНИЕ

Как работает базовая аутентификация:

1. Когда клиент пытается получить доступ к защищенному ресурсу, сервер отправляет HTTP-ответ с кодом статуса 401 Unauthorized и заголовком WWW-Authenticate.
2. Клиент отправляет новый запрос на сервер, включая в заголовке Authorization информацию об аутентификации. Заголовок Authorization имеет вид "Basic base64(username:password)", где username и password - это имя пользователя и пароль в кодировке UTF-8, а base64 - это алгоритм кодирования в Base64.
3. Сервер декодирует информацию об аутентификации из заголовка Authorization и проверяет ее на соответствие с сохраненными учетными данными. Если учетные данные совпадают, то сервер разрешает доступ к защищенному ресурсу, в противном случае - отправляет HTTP-ответ с кодом статуса 401 Unauthorized.

ВВЕДЕНИЕ

Недостатки:

- **Низкая безопасность**: Если данные передаются без шифрования (например, через HTTP вместо HTTPS), логин и пароль могут быть легко перехвачены злоумышленниками.
- **Отсутствие временных ограничений**: В отличие от других методов, базовая аутентификация не предусматривает срок действия сеанса — логин и пароль передаются при каждом запросе.
- **Неудобство при многократных запросах**: Логин и пароль постоянно передаются при каждом запросе, что создает дополнительную нагрузку и увеличивает риск утечки данных.

Когда использовать:

Базовая аутентификация может применяться в простых проектах, где безопасность не является критичным аспектом, либо в защищенных локальных сетях. Однако в большинстве случаев ее рекомендуется избегать в пользу более безопасных методов.

ВВЕДЕНИЕ

- **Token-based аутентификация**: вместо передачи логина и пароля на каждый запрос, этот метод аутентификации использует токен, который клиент получает после успешной аутентификации (уникальная строка, которая подтверждает его подлинность). Токен может быть передан в заголовках или в параметрах запроса. Каждый токен имеет срок действия, и сервер может проверить его наличие и срок действия для каждого запроса.

ВВЕДЕНИЕ

Процесс аутентификации в token-based аутентификации происходит следующим образом:

1. Пользователь отправляет запрос на сервер для входа в систему с использованием своих учетных данных (например, логин и пароль).
2. Сервер проверяет, правильные ли были предоставлены учетные данные, и если они правильны, генерирует токен для пользователя.
3. Токен возвращается пользователю в ответ на запрос на сервер для входа в систему.
4. Пользователь сохраняет токен и использует его для доступа к защищенным ресурсам на сервере.

ВВЕДЕНИЕ

Преимущества:

- **Безопасность**: Логин и пароль передаются только один раз — при первой аутентификации. В дальнейшем используется токен, что снижает риск утечки учетных данных.
- **Гибкость**: Токен может иметь срок действия, что позволяет автоматически завершать сеансы через определенное время. По истечении срока пользователь должен повторно пройти аутентификацию.
- **Удобство для масштабируемых систем**: Токены могут использоваться в распределенных системах без необходимости постоянно обращаться к базе данных для проверки учетных данных.

Недостатки:

- Токены могут быть украдены, если передаются через незашифрованные каналы (например, по HTTP).
- Требуется механизм обновления токенов по истечении их срока действия.

ВВЕДЕНИЕ

- **OAuth аутентификация**: это открытый стандарт аутентификации, который используется для авторизации пользователей в приложениях и социальных сетях. Этот метод позволяет пользователям предоставлять доступ к своим данным без передачи своих учетных данных. OAuth использует токены доступа для доступа к данным пользователя.

ВВЕДЕНИЕ

Процесс авторизации в OAuth происходит следующим образом:

1. Пользователь отправляет запрос на приложение, запрашивающее доступ к его ресурсам.
2. Приложение запрашивает у пользователя разрешение на доступ к его ресурсам.
3. Если пользователь дает согласие, приложение отправляет запрос на сервер авторизации, запрашивая токен доступа.
4. Сервер авторизации проверяет запрос на соответствие настроенным правилам и, если все правильно, выдает приложению токен доступа.
5. Приложение использует полученный токен доступа для доступа к ресурсам пользователя на сервере.

ВВЕДЕНИЕ

Преимущества OAuth:

- **Безопасность**: Логин и пароль пользователя не передаются напрямую стороннему приложению. Приложение получает ограниченный доступ к данным через токен.
- **Удобство для пользователей**: Пользователи могут входить в различные приложения, используя уже существующие учетные записи в популярных сервисах, таких как Google, GitHub или Facebook.
- **Масштабируемость**: OAuth позволяет авторизовать пользователя для доступа к ресурсам на других серверах, что делает его полезным для разработки API.

Недостатки:

- OAuth может быть сложным для настройки, особенно если требуется глубокая интеграция с внешними сервисами.
- Токены OAuth могут быть уязвимы, если их передача не защищена.

ВВЕДЕНИЕ

- **JWT аутентификация**: JSON Web Token (JWT) — это современный метод аутентификации, который использует токены в формате JSON. В отличие от обычных токенов, JWT содержит закодированную информацию о пользователе и его правах. Каждый JWT токен состоит из трех частей:
  - **Header (заголовок)**: Информация о типе токена и алгоритме подписи.
  - **Payload (полезная нагрузка)**: Данные о пользователе, такие как его ID, права доступа, роли и другие метаданные.
  - **Signature (подпись)**: Цифровая подпись, которая используется для проверки подлинности токена.

ВВЕДЕНИЕ

Процесс аутентификации в JWT-аутентификации происходит следующим образом:

1. Пользователь отправляет запрос на сервер для входа в систему с использованием своих учетных данных (например, логин и пароль).
2. Сервер проверяет, правильные ли были предоставлены учетные данные, и если они правильны, создает JWT для пользователя.
3. JWT возвращается пользователю в ответ на запрос на сервер для входа в систему.
4. Пользователь сохраняет JWT и использует его для доступа к защищенным ресурсам на сервере.

ВВЕДЕНИЕ

Преимущества JWT:

- **Самодостаточность и расширяемость**: Токен содержит все необходимые данные о пользователе, что исключает необходимость постоянного обращения к базе данных для проверки прав доступа.
- **Простота передачи**: JWT может передаваться через заголовки HTTP, параметры URL или куки, что делает его универсальным для различных типов приложений (включая SPA и мобильные приложения).
- **Безопасность**: Токен подписан цифровой подписью (например, с использованием HMAC или RSA), что гарантирует его подлинность. Если токен был изменен злоумышленником, подпись перестанет быть валидной, и сервер отклонит запрос.

Недостатки:

- **Токены нельзя отозвать**: Если токен был скомпрометирован, его сложно отозвать до истечения срока действия. Для этого нужно использовать дополнительные механизмы, такие как черные списки (blacklists).
- **Размер токена**: JWT токены могут быть относительно большими из-за их содержимого, что увеличивает объем передаваемых данных.

ОСНОВЫ СИСТЕМЫ ПОЛЬЗОВАТЕЛЕЙ DJANGO

Пользовательская система Django основана на пакете django.contrib.auth, встроенном в фреймворк Django. Этот пакет является пользовательской системой по умолчанию, используемой различными приложениями Django, включая django-admin.

```
# Application definition

INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "users",
    "students",
]
```

ОСНОВЫ СИСТЕМЫ ПОЛЬЗОВАТЕЛЕЙ DJANGO

Типы пользователей, подтипы, группы и разрешения

Существует два основных типа классов пользователей Django: User и AnonymousUser:

Если пользователь аутентифицирует себя (т.е. предоставляет действительное имя пользователя/пароль), Django распознает его как User.

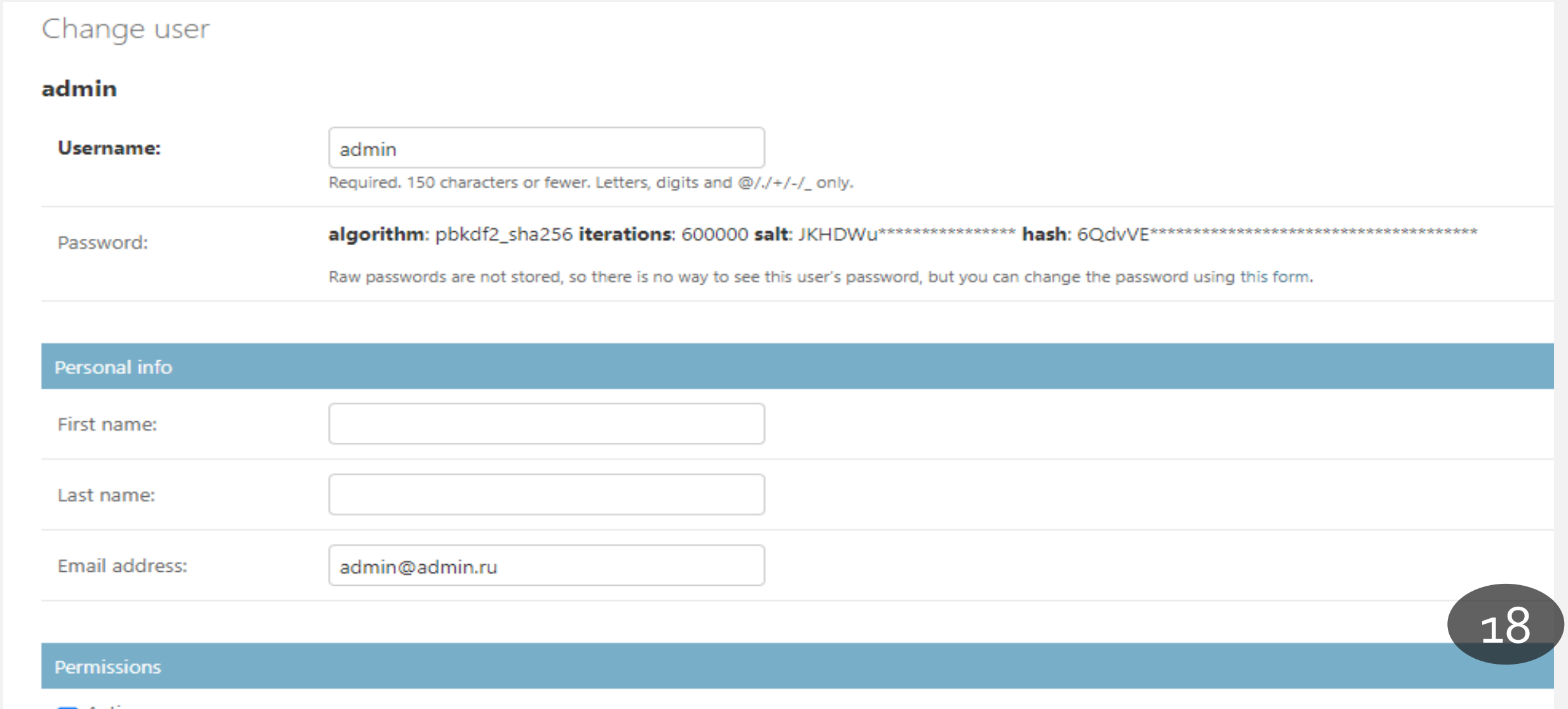
Если пользователь просто просматривает приложение без какой-либо аутентификации, Django распознает его как AnonymousUser.

Любой пользователь может быть далее классифицирован на один из подтипов:

- **superuser**: Самый мощный пользователь с правами на создание, чтение, обновление и удаление данных, в админке Django, которые включают в себя и записи других пользователей, вне зависимости от установленных разрешений на них.
- **staff**: Пользователь, помеченный как staff, может получить доступ к админ-панели. Но разрешения на создание, чтение, обновление и удаление данных в админке Django должны быть предоставлены пользователю явным образом.
- **active**: Пользователи, прошедшие все этапы регистрации, помечаются как активные, администратор может снять эту пометку.

ОСНОВЫ СИСТЕМЫ ПОЛЬЗОВАТЕЛЕЙ DJANGO

Панель администрирования. Профиль superuser



ОСНОВЫ СИСТЕМЫ ПОЛЬЗОВАТЕЛЕЙ DJANGO

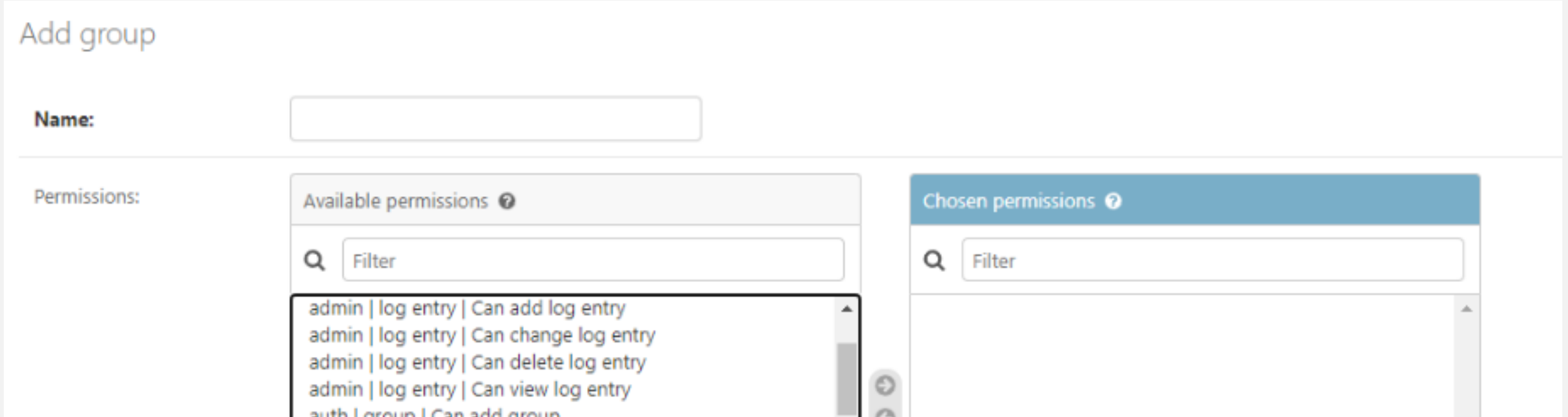
Хранение и использование паролей в Django:

- **Хэширование паролей**: Django использует мощные криптографические алгоритмы для хэширования паролей. По умолчанию, применяется алгоритм PBKDF2 с хэшем SHA256, но поддерживаются и другие алгоритмы, такие как bcrypt, Argon2, и SHA-1.
- **Соль (Salt)**: Вместе с паролем также используется случайная строка, называемая "соль" (salt). Это дополнительный уровень безопасности, который предотвращает использование предварительно вычисленных таблиц хэшей (таблицы радужных значений, "rainbow tables"). Соль генерируется случайным образом для каждого пароля и сохраняется в базе данных вместе с хэшом.
- **Структура хранения**: Пароль в базе данных хранится в поле password модели User.
- **Проверка пароля**: Когда пользователь вводит пароль для входа, Django хэширует введённый пароль с той же солью и сравнивает полученный результат с хранимым хэшем в базе данных. Если они совпадают, пользователь успешно аутентифицируется.

ОСНОВЫ СИСТЕМЫ ПОЛЬЗОВАТЕЛЕЙ DJANGO

Django также предлагает концепцию **класса Group** для предоставления набору пользователей одного и того же набора прав, без необходимости назначать их индивидуально.

Например, вы можете предоставить определённые права группе, а затем назначить пользователей в эту группу, чтобы упростить управление правами. Таким образом, вы можете одним шагом отозвать или добавить разрешения для набора пользователей, а также быстро предоставить новым пользователям те же разрешения:



ОСНОВЫ СИСТЕМЫ ПОЛЬЗОВАТЕЛЕЙ DJANGO

Вы можете назначить права Django на пользователя или группу, чтобы они могли выполнять CRUD(Create-Update-Delete) записи в моделях Django, процесс, который осуществляется через записи модели Permission.

Также вы можете назначить более грубые разрешения Django на методы url/view или содержимое шаблона, чтобы предоставить доступ пользователю, группе или даже получателю разрешения.

ОСНОВЫ СИСТЕМЫ ПОЛЬЗОВАТЕЛЕЙ DJANGO

Поля модели **django.contrib.auth.models.User**

Поле	Описание
username	Имя пользователя, не более 150 символов, может содержать буквенно-цифровые и _@+.- символы.
first_name	Имя, не более 150 символов, не обязательное значение.
last_name	Фамилия, не более 150 символов, не обязательное значение.
email	Адрес электронной почты, не обязательное значение.
password	Хэш и метаданные пароля. Обратите внимание, что Django не хранит необработанный пароль.
groups	Связь «многие-ко-многим» с django.contrib.auth.models.Group
user_permissions	Связь «многие-ко-многим» с django.contrib.auth.models.Permission
is_staff	Определяет, может ли пользователь получить доступ к сайту администратора.(булево значение)

ОСНОВЫ СИСТЕМЫ ПОЛЬЗОВАТЕЛЕЙ DJANGO

Атрибуты **django.contrib.auth.models.User**

Атрибут	Описание
is_anonymous	Для пользователя этот атрибут всегда возвращает False, он используется только как способ различать пользователя и анонимного пользователя.
is_authenticated	Для пользователя этот атрибут всегда возвращает True, он используется только для того, чтобы узнать, прошел ли пользователь AuthenticationMiddleware(представляющий текущего пользователя, вошедшего в систему).

ОСНОВЫ СИСТЕМЫ ПОЛЬЗОВАТЕЛЕЙ DJANGO

Методы django.contrib.auth.models.User

Метод	Описание
get_username()	Возвращает имя пользователя для пользователя. Поскольку модель пользователя можно изменить на другую, этот метод является рекомендуемым подходом вместо прямой ссылки на атрибут имени пользователя.
get_full_name()	Возвращает поля first_name и last_name с пробелом между ними.
get_short_name()	Возвращает имя.
set_password (raw_password**)**	Устанавливает пароль пользователя в заданную необработанную строку, заботясь о хешировании пароля. Обратите внимание, что если для параметра raw_password установлено значение None, в качестве пароля устанавливается непригодный для использования пароль, как если бы использовался

ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO

Чтобы использовать **встроенное приложение auth**, нам нужно подключить его маршруты в URL-файле проекта urls.py.

В моем примере для приложения auth URL- используется адрес accounts/, но вы можете использовать любой другой URL, который хотите.

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path("admin/", admin.site.urls),
    path('users/', include("django.contrib.auth.urls")),
    path('', include('students.urls')),
]
```

ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO

Приложение auth, которое мы сейчас включили, предоставляет нам несколько представлений аутентификации и URL-адресов для управления входом в систему, выходом из системы и управлением паролями.

URL-адреса, предоставленные приложением auth:

accounts/login/ [name='login']

accounts/logout/ [name='logout']

accounts/password_change/ [name='password_change']

accounts/password_change/done/ [name='password_change_done']

accounts/password_reset/ [name='password_reset']

accounts/password_reset/done/ [name='password_reset_done']

accounts/reset/<uidb64>/<token>/ [name='password_reset_confirm']

accounts/reset/done/ [name='password_reset_complete']

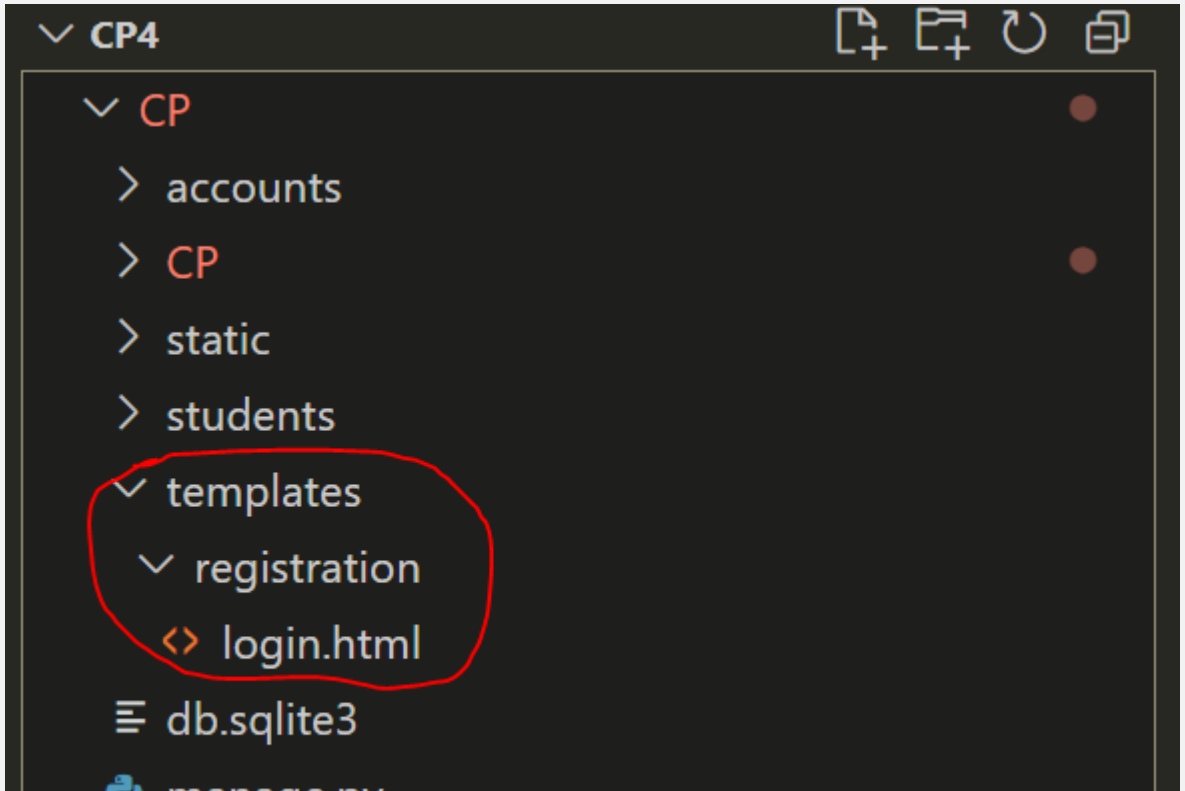
Для каждого шаблона URL также есть связанные представления аутентификации. Это означает, что нам нужно только создать шаблон для использования каждого!

ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO

Страница авторизации

Давайте сделаем нашу страницу входа! **Django, по умолчанию, будет искать шаблоны для приложения auth, в директории шаблонов, с именем registration.**

Создайте новую директорию с именем registration в директории шаблонов templates, которая расположена в корневой папке, а затем создайте в директории registration файл шаблона login.html.



ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO

Затем добавьте файл шаблона templates/registration/login.html следующий код:

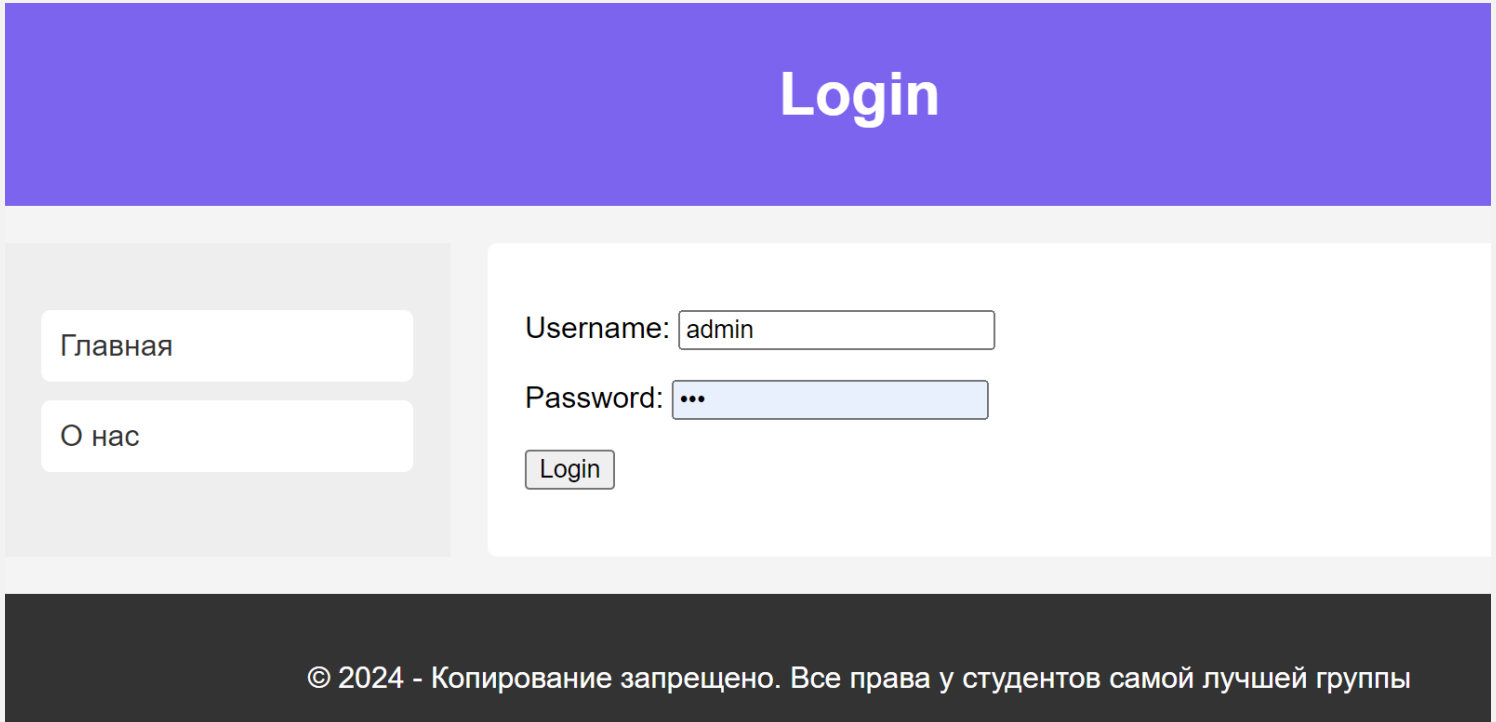
```
{% extends 'students/base.html' %}
{% block title %}Login{% endblock %}
{% block header %}Login{% endblock %}
{% block content %}
    <form method="post">
        {{ form.as_p }}
        {% csrf_token %}
        <p><input type="submit" value="Войти"></p>
    </form>
{% endblock %}
```

ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO

Это стандартная форма Django, использует **метод POST** для отправки данных формы и тег **{% csrf_token %}**, который добавит скрытое поле CSRF-токена, из соображений безопасности, а именно для предотвращения атак CSRF.

Содержимое формы выводится между тегами абзаца благодаря **{{ form.as_p }}**, а затем мы добавляется кнопка LOGIN:

Перейдём по адресу **http://127.0.0.1:8000/accounts/login/** и проверим работу:



ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO

Наша функция входа в систему теперь работает, но мы должны указать, куда перенаправлять пользователя после успешного входа в систему.

Для этого мы будем использовать параметр LOGIN_REDIRECT_URL в файле настроек settings.py, чтобы указать этот URL.

Добавим данный параметр со значением '/', в низ файла settings.py, чтобы пользователи перенаправлялись на главную страницу сайта:

```
127
128   DEFAULT_AUTO_FIELD = "django.db.models.BigAutoField"
129
130   LOGIN_REDIRECT_URL = '/'
131
```

ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO

Выход

Давайте добавим на нашу страницу ссылку для выхода, чтобы пользователи могли легко переключаться между двумя состояниями.

Возможности приложения auth уже предоставляют нам встроенный URL-адрес и представление для этого. Нам не нужно ничего отображать при выходе из системы, поэтому нет необходимости в шаблоне. Все, что мы действительно делаем после успешного запроса на «выход из системы», — это перенаправление на другую страницу.

Давайте отредактируем наш базовый шаблон base.html, и изменим часть кода нашего меню:

ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO

```
{% block nav %}
<li><a href="{% url 'index' %}">Главная</a></li>
<li><a href="{% url 'about' %}">О нас</a></li>
<hr>
{% if user.is_authenticated %}
    Привет, {{ user.username }}!
<form action="{% url 'logout' %}" method="post">
    {% csrf_token %}
    <a href="#" onclick="parentNode.submit();">Выйти</a>
</form>
{% else %}
    <p>Вы не авторизованы!</p>
    <a href="{% url 'login' %}">Войти</a>
{% endif %}
<hr>
{% endblock %}
```

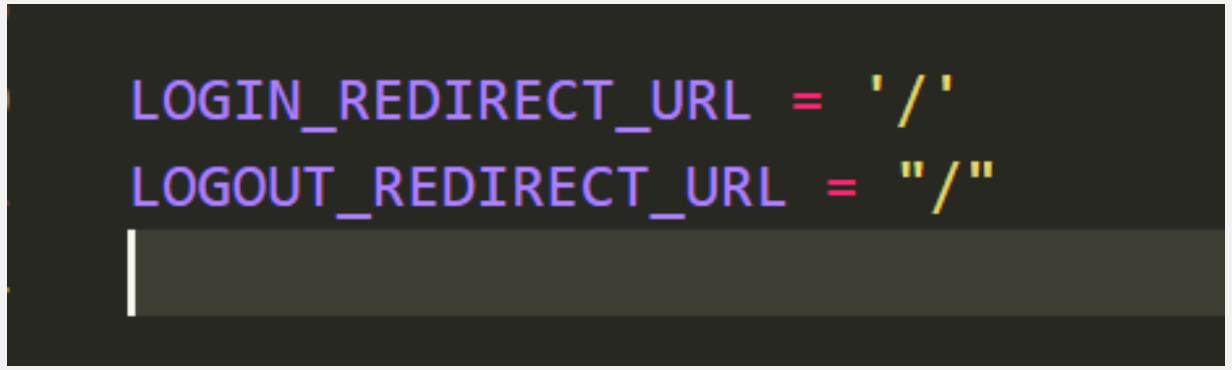
ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO

Затем обновите в файле настроек settings.py нашу ссылку перенаправления, которая называется LOGOUT_REDIRECT_URL.

Добавьте его прямо рядом с нашей переадресацией входа, чтобы нижняя часть файла settings.py выглядела следующим образом:

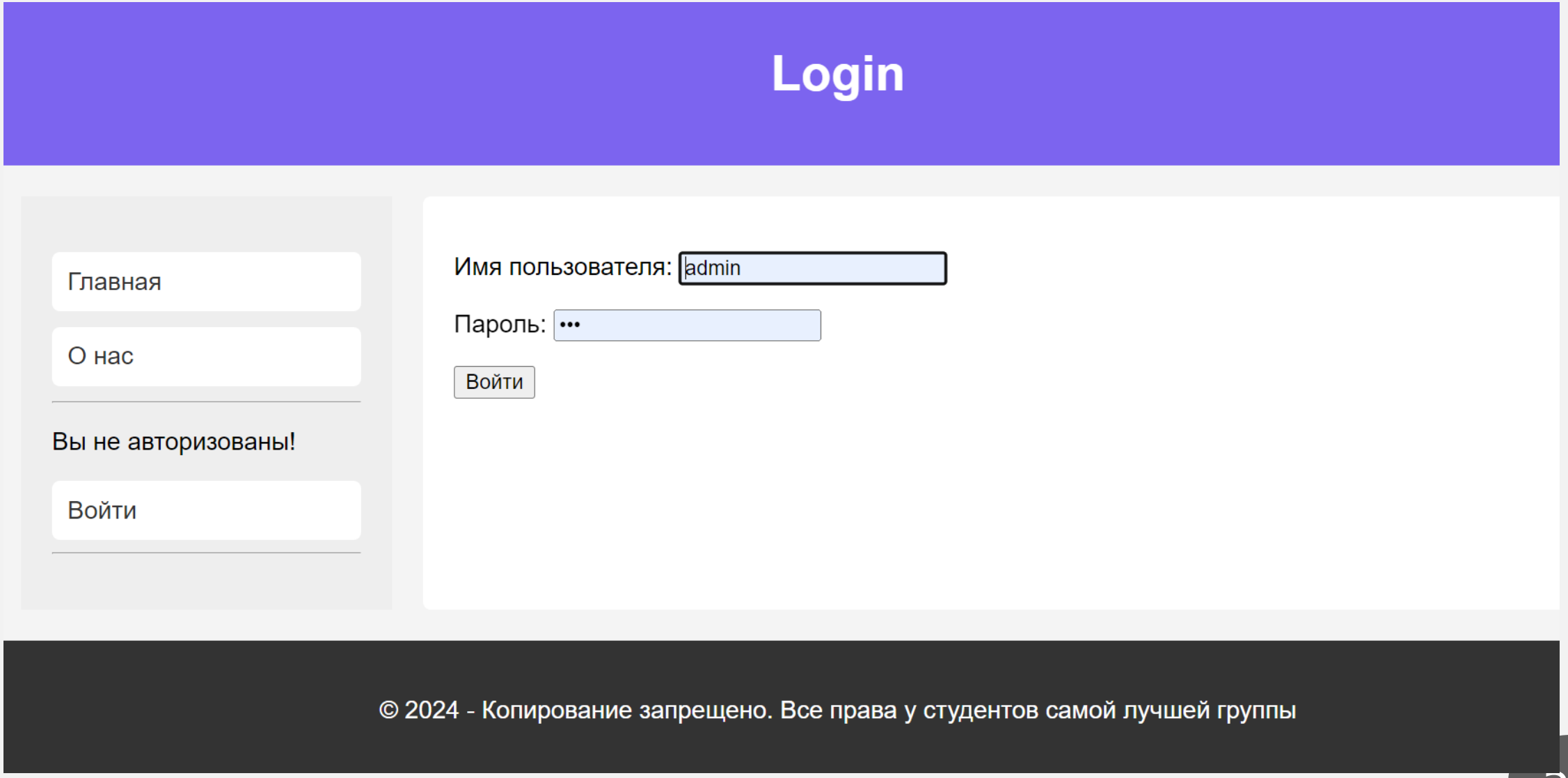
LOGIN_REDIRECT_URL = "/"

LOGOUT_REDIRECT_URL = "/"

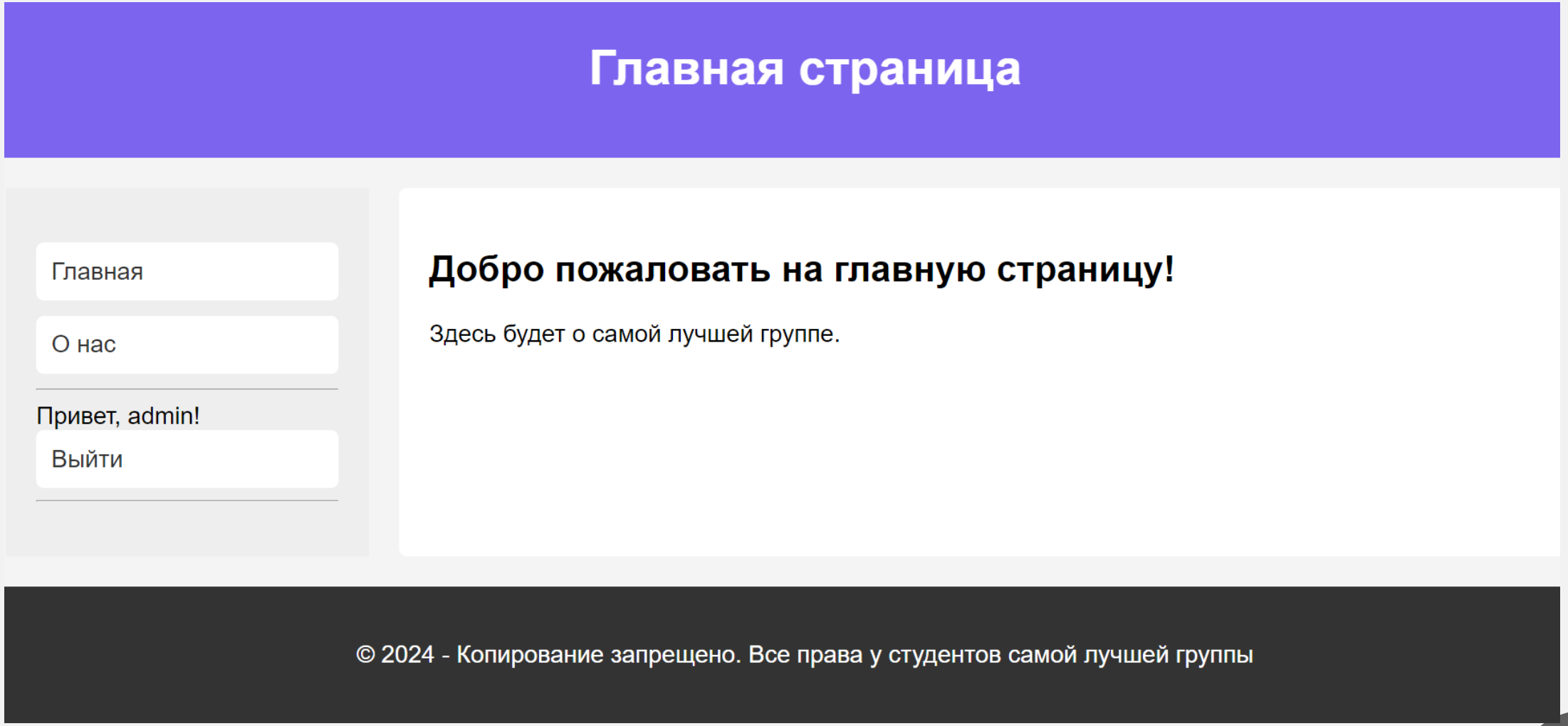


Теперь все работает корректно.

ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO



ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO



ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO

Чтобы неавторизованный пользователь видел только страницу авторизации и не мог получить доступ к другим страницам сайта, можно использовать **декоратор @login_required** для всех представлений, которые должны быть доступны только авторизованным пользователям.

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

@login_required
def index(request):
  return render(request, 'students/index.html')

@login_required
def about(request):
  return render(request, 'students/about.html')
```

ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO

Регистрация пользователей

Создадим новое приложение accounts (имя вы выбираете сами).

Обязательно добавьте новое приложение в список установленных приложений INSTALLED_APPS в файле настроек mysite/settings.py.

Затем добавьте URL-адрес уровня проекта для приложения accounts над включенным приложением Django auth.

Django будет искать шаблоны URL-адресов сверху вниз, поэтому, когда он увидит маршрут URL-адреса http://127.0.0.1:8000/accounts/, то он будет следовать сначала в наше приложение accounts, и только потом в auth:

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path("accounts/", include("accounts.urls")),
    path('accounts/', include("django.contrib.auth.urls")),
    path('', include('students.urls')),
]
```

ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO

Создайте новый файл urls.py в нашем приложении accounts и добавьте в него следующий код:

```
from django.urls import path
from .views import SignUpView

urlpatterns = [
    path("signup/", SignUpView.as_view(), name="signup"),
]
```

ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO

Теперь добавим в файл представлений views.py нашего приложения accounts следующий код:

```
from django.contrib.auth.forms import UserCreationForm
from django.urls import reverse_lazy
from django.views import generic

class SignUpView(generic.CreateView):
    form_class = UserCreationForm
    success_url = reverse_lazy("login")
    template_name = "registration/signup.html"
```

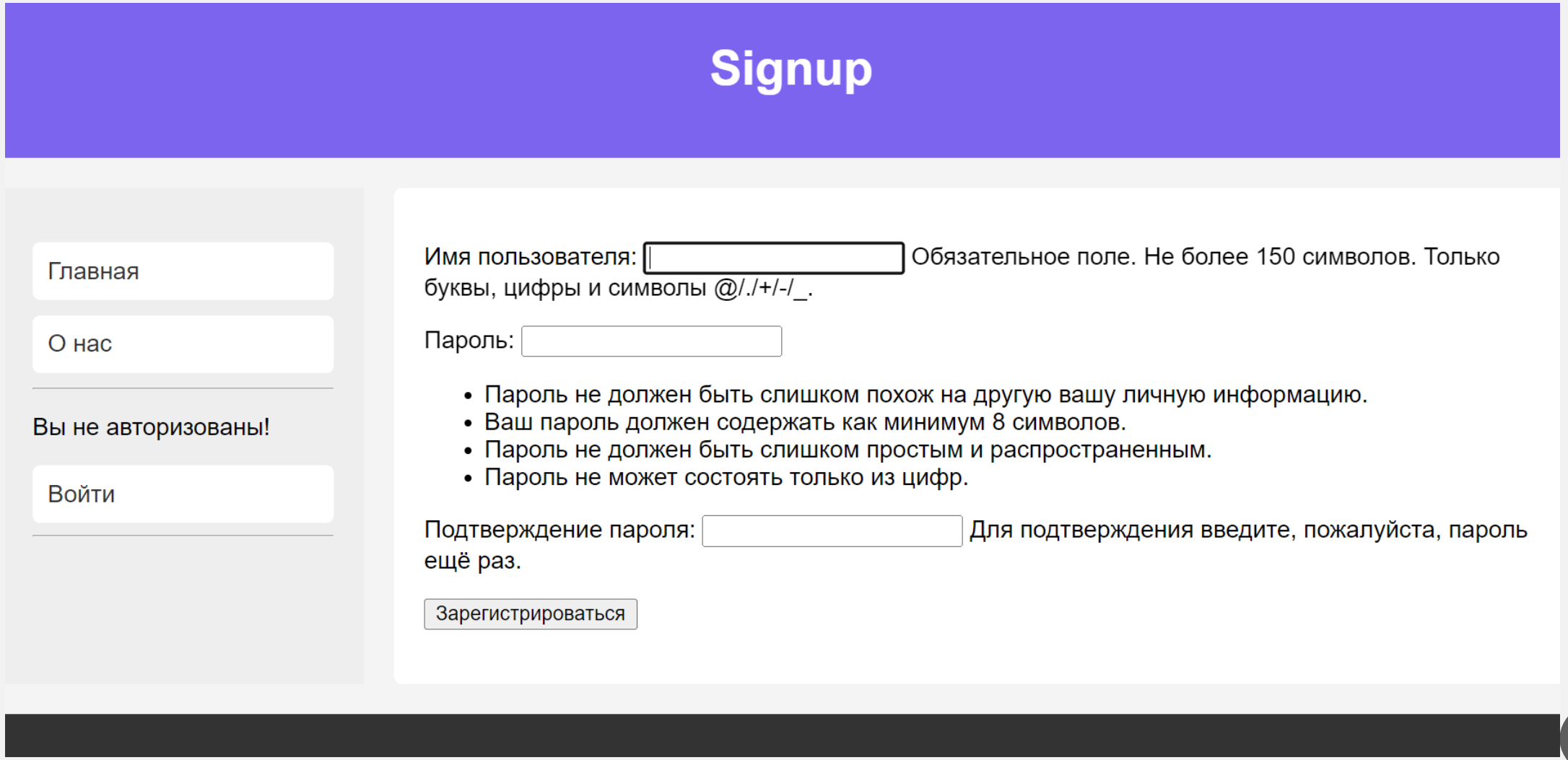
ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO

Создайте новый файл шаблона templates/registration/signup.html и заполните его этим кодом, который выглядит почти точно так же, как мы использовали для шаблона login.html:

```
{% extends 'students/base.html' %}
{% block title %}Signup{% endblock %}
{% block header %}Signup{% endblock %}
{% block content %}
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <p><input type="submit" value="Зарегистрироваться"></p>
    </form>
{% endblock %}
```

ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO

Перейдите к странице регистрации http://127.0.0.1:8000/accounts/signup/:



ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO

Как вы уже поняли UserCreationForm() в django.contrib.auth.forms предоставляет ограниченное количество полей.

Предположим, мы хотим отправить письмо с подтверждением пользователя, но мы не можем этого сделать, потому что при регистрации у него нет поля электронной почты.

Для этого либо мы модифицируем форму UserCreationForm, включая поле электронной почты и средство проверки, либо создаем новую форму регистрации пользователя с нуля.

Мы создадим новую форму регистрации пользователя, потому что она обеспечит полный контроль над формой.

Сначала мы создадим файл forms.py и новый класс SignUpForm в файле forms.py приложения accounts:

ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO

```
from django import forms
from django.contrib.auth.models import User
from django.contrib.auth.forms import UserCreationForm


class SignUpForm(UserCreationForm):
    username = forms.CharField(max_length=30)
    email = forms.EmailField(max_length=200)

    class Meta:
        model = User
        fields = ['username', 'email', 'password1', 'password2']
```

ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO

Изменим views.py:

```
from .forms import SignUpForm
from django.urls import reverse_lazy
from django.views import generic


class SignUpView(generic.CreateView):
    form_class = SignUpForm
    success_url = reverse_lazy("login")
    template_name = "registration/signup.html"
```

То есть мы просто импортировали из форм новую форму и переопределили наш form_class.

ПРИЛОЖЕНИЕ АВТОРИЗАЦИИ DJANGO

Signup

Главная

О нас

Вы не авторизованы!

Войти

Username:

Email:

Пароль:

- Пароль не должен быть слишком похож на другую вашу личную информацию.
- Ваш пароль должен содержать как минимум 8 символов.
- Пароль не должен быть слишком простым и распространенным.
- Пароль не может состоять только из цифр.

Подтверждение пароля: Для подтверждения введите, пожалуйста, пароль ещё раз.

Зарегистрироваться