Django: пользователи | Я.Шпора

> После развёртывания Django-проекта не спешите выполнять встроенные миграции: сперва продумайте стратегию работы с пользователями и решите, как будет выглядеть модель пользователя.

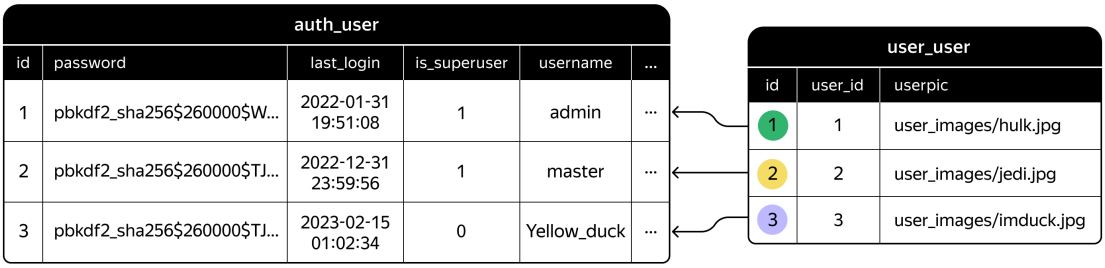
Встроенная модель пользователя `User` готова к работе сразу после установки Django и выполнения встроенных миграций.

Изменение модели пользователя

Изменить встроенную модель пользователя можно двумя способами:

1. **Расширить встроенную модель пользователя.**

   Для расширения модели пользователя создают дополнительную модель, в которую добавляют поля, которых не хватает в основной модели пользователя. Эту модель связывают с основной моделью пользователя через связь «один-к-одному».



2. **Заменить встроенную модель пользователя на кастомную**

   При этом подходе следует создать собственную модель пользователя, унаследовав её от `AbstractUser`, и добавить в модель-наследник необходимые поля.

> Этот способ доступен только в самом начале работы над проектом, до выполнения встроенных миграции: следует сначала создать и настроить модель пользователя, и лишь после этого выполнять встроенные миграции

Импортируем нужные сущности и описываем собственную модель пользователя:

```
from django.contrib.auth.models import AbstractUser
from django.db import models


class MyUser(AbstractUser):
    bio = models.TextField('Биография', blank=True)
```

Необходимо внести правки в *settings.py*: нужно указать, что вместо стандартной модели пользователя будет использоваться кастомная модель (в нашем примере — модель `MyUser` из приложения **users**).

```
AUTH_USER_MODEL = 'users.MyUser'
```

После этого надо создать и применить миграции.

> Разработчики Django рекомендуют всегда обращаться к модели пользователя не напрямую, а через функцию `get_user_model()`, даже если модель пользователя не изменена.

Для более глубокого изменения модели пользователя кастомную модель пользователя надо наследовать не от `AbstractUser`, а от базовой модели `AbstractBaseUser` из модуля `django.contrib.auth.base_user`.

Создание суперпользователя

```
python manage.py createsuperuser
```

Настройка страниц входа и выхода пользователей

В модуле `django.contrib.auth` есть несколько CBV, предназначенных для страниц управления пользователями:

Назначение	URL	CBV	HTML-шаблон
Авторизация	`login/`	**LoginView**	login.html
Выход из аккаунта	`logout/`	**LogoutView**	logged_out.html
Смена пароля: форма **«Придумайте новый пароль»**	`password_change/`	**PasswordChangeView**	password_change_form.html
Смена пароля: страница с уведомлением **«Пароль успешно изменён»**	`password_change/done/`	**PasswordChangeDoneView**	password_change_done.html
Восстановление пароля: форма **«Укажите свой email»**	`password_reset/`	**PasswordResetView**	password_reset_form.html
Восстановление пароля: страница с уведомлением об **«Вам на email отправлена ссылка для восстановления пароля»**	`password_reset/done/`	**PasswordResetDoneView**	password_reset_done.html
Восстановление пароля: страница с формой **«Придумайте новый пароль»**; открывается по ссылке из письма	`reset/<uidb64>/<token>/`	**PasswordResetConfirmView**	password_reset_confirm.html
Восстановление пароля: страница с уведомлением **«Пароль успешно изменён»**	`reset/done/`	**PasswordResetCompleteView**	password_reset_complete.html

Подключение путей из модуля `django.contrib.auth`:

```
# <project_dir>/urls.py
...

urlpatterns = [
    ...
    path('auth/', include('django.contrib.auth.urls')),
    ...
]
```

Кастомизация шаблонов

Шаблоны для страниц управления пользователем в Django удобно хранить в каталоге *templates/registration/.*

```
└──templates/
    ├── ...
    └── registration/
        ├── logged_out.html
        ├── login.html
        ├── password_change_done.html
        ├── password_change_form.html
        ├── password_reset_complete.html
        ├── password_reset_confirm.html
        ├── password_reset_done.html
        └── password_reset_form.html
```

Кастомный шаблон для страницы входа: CBV LoginView

Класс `LoginView` применяет шаблон по умолчанию, этот шаблон должен называться *login.html*. Разработчик должен создать этот шаблон самостоятельно. Пример кода можно взять в документации.

В файле *settings.py* в константе `LOGIN_REDIRECT_URL` нужно указать адрес, куда после входа будет переадресован пользователь. Это может быть относительный путь (например `/birthday/`), либо имя URL-паттерна (`namespace` и `name` пути страницы в *urls.py;* например `pages:homepage`).

```
LOGIN_REDIRECT_URL = 'pages:homepage'
```

Кастомный шаблон страницы выхода: CBV LogoutView

CBV `LogoutView` использует HTML-шаблон *logged_out.html.*

В шаблоне можно получить объект пользователя, запросившего страницу выхода: этот объект передаётся в словаре контекста под ключом `user`.

Для авторизованного пользователя можно получить `user.email`, `user.is_superuser` и другие свойства; можно определить, авторизован ли пользователь: свойство `user.is_authenticated` возвращает `True` или `False`.

Настройка страницы регистрации

В `django.contrib.auth` нет готового CBV для регистрации нового пользователя, нет шаблона страницы регистрации, нет маршрута. Для создания страницы регистрации можно применить CBV `CreateView`.

В качестве формы будем использовать встроенный в Django класс формы создания пользователя `UserCreationForm`.

Этот класс применяется только в том случае, если в проекте используется встроенная модель пользователя.

```
# urls.py

from django.contrib.auth.forms import
from django.views.generic.edit import CreateView
...
# К импортам из django.urls добавьте импорт функции reverse_lazy
from django.urls import include, path, reverse_lazy

urlpatterns = [
    ...
    path(
        'auth/registration/',
        CreateView.as_view(
            template_name='registration/registration_form.html',
            form_class=UserCreationForm,
            success_url=reverse_lazy('pages:homepage'),
        ),
        name='registration',
    ),
    ...
]
```

Регистрация пользователя при переопределении модели

Класс формы регистрации `UserCreationForm` рассчитан только на работу со встроенной моделью пользователя. Если модель пользователя переопределена — нужно переопределить и класс формы регистрации.

Для создания кастомной формы регистрации нужно создать класс — наследник класса `UserCreationForm`; в нём нужно будет указать новую модель пользователя. Указывать модель пользователя лучше через `get_user_model()`.

```
# users/forms.py
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth import get_user_model

# Получаем модель пользователя:
User = get_user_model()


class CustomUserCreationForm(UserCreationForm):

    # Наследуем класс Meta от соответствующего класса родительской формы.
    # Так этот класс будет не перезаписан, а расширен.
    class Meta(UserCreationForm.Meta):
        model = User
```

Остальные действия аналогичны работе со встроенным классом формы `UserCreationForm`.

Ограничение доступа для view-функций: декоратор @login_required

Декоратор `@login_required` в Django используется для защиты представлений (views) от неавторизованного доступа.

```
...
from django.contrib.auth.decorators import login_required
from django.http import HttpResponse


@login_required
def simple_view(request):
    return HttpResponse('Страница только для залогиненных пользователей!'
```

Этот декоратор применяется только для view-функций; использовать его для CBV не получится.

Ограничение доступа для CBV: миксин LoginRequiredMixin

Для Class Based View применяют миксин LoginRequiredMixin. Он добавляет в CBV проверку — аутентифицирован ли пользователь, сделавший запрос.

```
from django.contrib.auth.mixins import LoginRequiredMixin


class BirthdayCreateView(LoginRequiredMixin, CreateView):
    model = Birthday
    form_class = BirthdayForm
```

Миксин `LoginRequredMixin` должен быть на самой левой позиции, первым в списке наследования.

Практикум