*Тема 4.3. Формы в Django* **(2+4=6час)**

***Лекционное занятие 1 (1 час)***

HTML форма – это группа из одного или нескольких полей/виджетов на веб-странице, которая используется для сбора информации от пользователей для последующей отправки на сервер. Формы представляются гибким механизмом сбора данных, поскольку имеют целый набор виджетов для ввода различных типов данных, как то: текстовые поля, флажки, переключатели, установщики дат и т. д.

Работа с формами может показаться достаточно сложной. Разработчикам надо описать ее в HTML, проверить ё валидность, а также, на стороне сервера, проверять введённые данные (а возможно и на стороне клиента), далее, в случае ошибок нужно опять вывести форму и, при этом, указать, что пошло не так, а в случае успеха обработать данные и проинформировать об этом пользователя. Django значительно облегчает работу с формами, беря большую часть действий на себя. Он предоставляет фреймворк, позволяющий определять форму и её поля программно, а затем использовать это для генерации непосредственно кода HTML-формы, и для контроля за процессом валидации и других пользовательский взаимодействий с формой.

В HTML форма - набор элементов внутри <form>...</form>, которые позволяют клиенту вводить текст, выбирать параметры, манипулировать объектами/элементами управления и т.д., а затем отправлять информацию на сервер. Некоторые из элементов интерфейса, такие, как ввод текста или флажки, встроены в HTML. Другие много сложнее, например, интерфейс выбора даты или работы с экранными элементами управления, где использует JavaScript, CSS, а также <input>элементы HTML.

Любая форма содержит как минимум одно поле-тэг input типа type="submit".

Элемент input с type="submit" будет показана как кнопка (по умолчанию), нажав на которую, пользователь отправляет введённые им данные на сервер (в данном случае только значение поля с идентификатором team\_name). Атрибуты формы определяют способ отправки данных на сервер (method) и куда (action):

* action: Это ресурс/URL-адрес куда будут отправлены данные для обработки. Если значение не установлено (то есть, значением поля является пустая строка), тогда данные будут отправлены в отображение (функцию, или класс), которое сформировало текущую страницу.
* method: HTTP-метод, используемый для отправки данных: post, или get.
  + POST используеться если отправка данных приведёт к изменению базы данных сервера. Применение метода повышает уровень защиты от CSRF (подделки запроса).
  + GET применяют для форм, где нет изменения баы данных (поисковых запросов, и т.п.), а также и для создания внешних ссылок на ресурсы сайта.

Django предоставляет класс Form, используемый для HTML. Он описывает форму, ее работу и вид, чем похож на класс ModelForm, создающий форму по модели, но в данном случае не требует ее конкретизации.

**from django import forms**

Главными моментами, которые берут на себя формы Django являются:

* показ формы по умолчанию при первом запросе со стороны пользователя. Форма может содержать пустые поля (например, если вы создаёте новую запись в базе данных), или они (поля) могут иметь начальные значения (например, если вы изменяете запись, или хотите заполнить её каким-либо начальным значением). Форма в данный момент является несвязанной, потому что она не ассоциируется с какими-либо введёнными пользователем данными (хотя и может иметь начальные значения).
* получение данных из HTML-формы со стороны клиента и связывание их с формой (классом формы) на стороне сервера. Связывание данных с формой означает, что данные, введённые пользователем, а также возможные ошибки, при переотрисовке в дальнейшем, будут относиться именно к данной форме, а не к какой-либо ещё.
* очистка и валидация данных. Очистка данных представляет проверку на наличие возможных значений, или вставок в поля ввода (то есть очистка – это удаление неправильных символов, которые потенциально могут использоваться для отправки вредоносного содержимого на сервер), с последующей конвертацией очищенных данных в подходящие типы данных Python. Валидация проверяет, значения полей (например, правильность введённых дат, их диапазон и так далее).
* если какие-либо данные являются неверными, то выполнение перерисовки формы, но на этот раз, с уже введёнными пользователем данными и сообщениями об ошибках, описывающих возникшие проблемы.
* если все данные верны, то исполнение нужных действий (например, сохранение данных, отправка писем, возврат результата поиска, загрузка файла и так далее)
* когда всё завершено, то перенаправление пользователя на другую страницу.

Django Form не включает теги или кнопку отправки. Мы должны сами указать их в шаблоне. Часто используемые поля и их подробности приведены в таблице ниже.

**Таблица 1**

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Класс | HTML-ввод | Пустое значение |
| BooleanField | класс BooleanField(\*\* kwargs) | CheckboxInput | False |
| CharField | класс CharField(\*\* kwargs) | TextInput | Все, что указано как empty\_value. |
| ChoiceField | класс ChoiceField(\*\* kwargs) | Select | ”(пустая строка) |
| DateField | класс DateField(\*\* kwargs) | DateInput | None |
| DateTimeField | класс DateTimeField(\*\* kwargs) | DateTimeInput | None |
| DecimalField | класс DecimalField(\*\* kwargs) | NumberInput | None |
| EmailField | класс EmailField(\*\* kwargs) | EmailInput | ”(пустая строка) |
| FileField | класс FileField(\*\* kwargs) | ClearableFileInput | None |
| ImageField | класс ImageField(\*\* kwargs) | ClearableFileInput | None |

Django предоставляет возможность работать с системой аутентификации и авторизации ("permission") пользователя, реализованную на основе фреймворка работы с сессиями. Это встроенная система django.contrib.auth Система аутентификации и авторизации позволяет владельцу приложения проверять учётные данные пользователей и определять какие действия какой пользователь может выполнять. Данный фреймворк включает в себя встроенные модели для Пользователей и Групп (основной способ применения прав доступа для более чем одного пользователя), непосредственно саму систему прав доступа (permissions)/флаги, которые определяют может ли пользователь выполнить задачу, с какой формой и отображением для авторизованных пользователей, а так же получить доступ к контенту с ограниченным доступом.

Необходимо сделать следующее замечание. В соответствии с идеологией Django система аутентификации является очень общей и, таким образом, не предоставляет некоторые возможности, которые присутствуют в других системах веб-аутентификации. Для решения некоторых общих задач предлагается использовать пакеты сторонних разработчиков.

Приложение auth представляет собой встроенную систему аутентификации в Django, которая позволяет разработчикам добавлять аутентификацию в свои приложения, не изобретая велосипед, пытаясь реализовать базовую функциональность с нуля.

Пример формы регистрации пользователя forms.py

*from django import forms*

*class LoginForm(forms.Form):*

*username = forms.CharField(); password = forms.CharField(widget=forms.PasswordInput)*

В каталоге приложения откроем файл views.py и добавим следующий код:

*from django.http import HttpResponse*

*from django.shortcuts import render*

*from django.contrib.auth import authenticate, login*

*from .forms import LoginForm*

*def user\_login(request):*

*if request.method == 'POST':*

*form = LoginForm(request.POST)*

*if form.is\_valid():*

*cd = form.cleaned\_data*

*user = authenticate(request, username=cd['username'], password=cd['password'])*

*if user is not None:*

*if user.is\_active:*

*login(request, user); return HttpResponse('Authenticated successfully')*

*else: return HttpResponse('Disabled account')*

*else: return HttpResponse('Invalid login')*

*else: form = LoginForm(); return render(request, 'login.html', {'form': form})*

Пример добавления формы в login.html

*<form action="." method="post">*

*{{ form.as\_p }}{% csrf\_token %}<p><input type="submit" value="Вход"></p>*

*</form>*

***Практическое занятие 1 (1 час)***

1. Перейдем в каталог с проектом project\_django.
2. Создадим новое приложение account, для этого в терминале введем команду:

*python manage.py startapp account*

Django создаст каталог приложения. Структура проекта соответствует Рис.1:

|  |  |
| --- | --- |
| 1. В settings.py, добавим код под INSTALLED\_APPS:   *INSTALLED\_APPS = [ 'django.contrib.admin',*  *'django.contrib.auth', 'django.contrib.contenttypes',*  *'django.contrib.sessions', 'django.contrib.messages',*  *'django.contrib.staticfiles', 'hello', 'web', 'account',]*  4) В каталоге приложения account создадим папку templates для шаблонов страниц приложения. Создадим в папке templates файл base.html со следующим кодом: | Рис.1.Структура проекта |

*<!doctype html> <html lang="en"><head> <meta charset="utf-8">*

*<title>{% block title %}{% endblock %}</title> {% load static %}*

*<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw7cmDA6j6gD" crossorigin="anonymous">*

*</head><body> <div id="header"> <span class="logo"></span> </div>*

*<div id="content"> {% block content %}{% endblock %} </div></body> </html>*

1. Создадим в папке templates файл login.html со следующим кодом:

*{% extends "base.html" %}{% block title %}Страница входа{% endblock %}*

*{% block content %}<h1>Вход в систему</h1><p>Используйте эту форму для входа:*

*<form action="." method="post"> {{ form.as\_p }} {% csrf\_token %}*

*<p><input type="submit" value="Вход"></p></form>{% endblock %}*

1. Далее необходимо создать файл forms.py

*from django import forms*

*class LoginForm(forms.Form):*

*username = forms.CharField(); password = forms.CharField(widget=forms.PasswordInput)*

1. Далее необходимо задать представления в файле account/views.py:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | *from django.http import HttpResponse*  *from django.shortcuts import render*  *from django.contrib.auth import authenticate, login*  *from .forms import LoginForm*  *def user\_login(request):*  *if request.method == 'POST':*  *form = LoginForm(request.POST)*  *if form.is\_valid():*  *cd = form.cleaned\_data*  *user=authenticate(request,username=cd['username'],password=cd['password'])*  *if user is not None:*  *if user.is\_active:*  *login(request, user); return HttpResponse('Authenticated successfully')*  *else: return HttpResponse('Disabled account')*  *else: return HttpResponse('Invalid login')*  *else: form = LoginForm(); return render(request, 'login.html', {'form': form})* |

1. Откроем urls.py в папке проекта my\_project и добавим следующий код:

*from django.contrib import admin*

*from django.urls import path, include*

*urlpatterns = [ path('admin/', admin.site.urls), path('', include('hello.urls')),*

*path('web/', include('web.urls')), path('account/', include('account.urls')),]*

Далее создадим urls.py в папке account, который содержит следующий код:

|  |  |  |
| --- | --- | --- |
| 1  2  3 | *from django.urls import path*  *from account import views*  *urlpatterns=[path('login/',views.user\_login, name='login'),]* | |
| 1. Запустим локальный сервер командой в терминале: *python manage.py runserver.* Перейдя на адрес 127.0.0.1:8000/account/login, увидим: | | Рис. 2. Страница входа login.html |

***Лекционное занятие 2 (1 час)***

Функциональность форм Django может упростить и автоматизировать огромную часть этой работы, а также сделать это более безопасно, чем большинство программистов могли бы сделать в коде, который они написали сами.

Django обрабатывает три различные части работы, связанной с формами:

* подготовка и реструктуризация данных для последующей визуализации;
* создание HTML-форм для данных;
* получение и обработка представленных форм и данных от клиента.

Можно написать код, который будет делать все это вручную, но Django значительно облегчает работу программисту.

В основе лежит класс Django Form. Точно так же, как модель Django описывает логическую структуру объекта, его поведение и то, как его части представлены нам, класс Form описывает форму и определяет, как она работает и отображается.

Подобно тому, как поля класса модели отображаются на поля базы данных, поля класса формы отображаются на элементы HTML-формы <input>. (Поле ModelForm отображает поля класса модели на элементы HTML-формы <input> через Form; на этом основана система администрирования Django).

Поля формы сами по себе являются классами; они управляют данными формы и выполняют проверку при отправке формы. DateField и FileField обрабатывают совершенно разные типы данных и должны делать с ними разные вещи.

Поле формы наблюдается в браузере как HTML «виджет» – часть интерфейса. Каждый тип поля имеет соответствующее значение по умолчанию Widget class, но оно может быть переопределено по мере необходимости.

Рендеринг формы в шаблоне включает в себя практически ту же работу, что и рендеринг любого другого типа объекта, но есть несколько ключевых различий.

В случае экземпляра модели, не содержащего данных, редко, если вообще когда-либо, будет полезно что-то делать с ним в шаблоне. С другой стороны, имеет смысл отобразить пустую форму, когда хотим, чтобы пользователь заполнил ее.

Поэтому, когда мы работаем с экземпляром модели в представлении, то обычно получаем его из базы данных. Работая с формой мы создаем ее в представлении.

Мы можем создать форму пустой или заполнив, например, с помощью:

* данных из сохраненного экземпляра модели (как в случае с админскими формами);
* данных, которые мы собрали из других источников;
* данных, полученных от предыдущей отправки HTML-формы.

Последний из этих случаев наиболее интересен, поскольку именно он позволяет пользователям не только читать сайт, но и отправлять на него информацию.

Все классы форм создаются как подклассы либо django.forms.Form, либо django.forms.ModelForm. Можно считать ModelForm подклассом Form. Form и ModelForm фактически наследуют общую функциональность от (частного) класса BaseForm, но эта деталь редко бывает важной. Начало создания формы в Django:

*forms.py*

*from django import forms*

*class NameForm(forms.Form):*

*your\_name = forms.CharField(label='Your name', max\_length=100)*

Это определяет класс Form с одним полем (your\_name). Применяется к полю удобная для человека метка, которая появится в <label> при его отображении (хотя в данном случае указанная нами метка label фактически является той же самой, которая была бы сгенерирована автоматически, если бы мы ее опустили).

Максимальная длина поля задается max\_length. Это накладывает условие maxlength="100" на HTML <input> (таким образом, браузер должен предотвратить ввод пользователем более чем этого количества символов), а также означает, что когда Django получит форму обратно от браузера, то проверит размер данных.

У Form есть метод is\_valid(), запускающий процедуры валидации для всех его полей. При вызове, если все поля содержат достоверные данные, он:

* возвращает True
* помещает данные формы в ее атрибут cleaned\_data.

Вся форма при первом отображении будет выглядеть следующим образом:

*<label for="your\_name">Your name: </label>*

*<input id="your\_name" type="text" name="your\_name" maxlength="100" required>*

Обратите внимание, что не включает в себя теги <form> или кнопку отправки. Необходимо самим предусмотреть их в шаблоне.

Каждое поле формы имеет соответствующий Widget class, который в свою очередь соответствует виджету HTML-формы, например <input type="text">.

Виджет – представление элемента ввода HTML в Django. Виджет обрабатывает рендеринг HTML-кода и извлечение данных из словаря GET/POST, который соответствует виджету. Всякий раз, когда программист указывает поле в форме, Django использует виджет по умолчанию, соответствующий типу отображаемых данных. У каждого поля есть предопределенный виджет, например, у IntegerField по умолчанию есть виджет NumberInput. Виджеты не следует путать с полями формы.

Программист может переопределить виджет по умолчанию для каждого поля.

Каждый раз, когда программист указывает поле в форме, Django будет использовать виджет по умолчанию, соответствующий типу отображаемых данных.

Однако, если вы хотите использовать другой виджет для поля, вы можете использовать widget аргумент в определении поля. Например:

*from django import forms*

*class CommentForm(forms.Form):*

*name=forms.CharField(); url=forms.URLField();*

*comment=forms.CharField(widget=forms.Textarea)*

В данном примере определена форма с комментарием, которая использует более крупный Textarea виджет, а не виджет по TextInput умолчанию.

Многие виджеты имеют необязательные дополнительные аргументы; их можно задать при определении виджета в поле.

Когда Django отображает виджет в формате HTML, он отображает только очень минимальную разметку, то есть Django не добавляет имена классов или любые другие атрибуты, специфичные для виджета. Это означает, например, что все TextInput виджеты будут отображаться одинаково на ваших веб-страницах.

Есть 2 способа настройки виджетов: для экземпляра для класса.

**Оформление экземпляров виджетов.** Если программист хочет, чтобы один экземпляр виджета отличался от другого, вам нужно будет указать дополнительные атрибуты во время создания экземпляра объекта widget и присвоения его полю формы (и, возможно, добавить некоторые правила в ваши CSS-файлы).

**Оформление классов виджетов.** С помощью виджетов можно добавлять ресурсы (css и javascript) и более глубоко настраивать их внешний вид и поведение.

В двух словах, вам нужно будет создать подкласс виджета и либо определить внутренний класс “Media”, либо создать свойство “media”. Эти методы включают в себя несколько продвинутое программирование на Python.

Для быстрого создания регистрации пользователя, можно импортировать модули аутентификации, например, Django UserCreationForm. Это встроенный модуль, наследуемый от ModelForm. Пользовательские объекты являются основным компонентом аутентификации. Их используют для доступа, регистрации профилей пользователей, сопоставления контента с создателем и т.п. UserCreationForm можно использовать, если доступны веб-приложения Django. Он имеет три поля: имя, пароль 1 и пароль 2 (обычно - для подтверждения пароля). Чтобы использовать UserCreationForm, нужно импортировать его из django.contrib.auth.forms:

*from django.contrib.auth.forms import UserCreationForm*

*from django.urls import reverse\_lazy*

*from django.views import generic*

*class Register(generic.CreateView):*

*form\_class = UserCreationForm; success\_url = reverse\_lazy('login'); template\_name = 'register.html'*

**Использование обработчиков аутентификации Django**

Django реализует несколько форм и обработчиков в подсистеме аутентификации, которые легко использовать. В большинстве случаев достаточно использовать стандартные средства, уже реализованные в Django.

Для работы с аутентификацией Django предоставляет обработчики-классы django.contrib.auth.views:

* LoginView – обработчик входа пользователя в его аккаунт;
* LogoutView – обработчик выхода пользователя из-под его учетной записи.

В Django реализованы обработчики смены пароля пользователем:

* PasswordChangeView – обрабатывает форму смены пароля;
* PasswordChangeDoneView – обработчик, на который будет перенаправлен пользователь после успешной смены пароля.

Также реализованы обработчики для восстановления пароля пользователя:

* PasswordResetView – обработчик восстановления пароля. Он генерирует временную ссылку с токеном и отправляет ее на электронную почту пользователя;
* PasswordResetDoneView – отображает страницу с сообщением о том, что ссылка восстановления пароля была отправлена на электронную почту;
* PasswordResetConfirmView – позволяет пользователю указать новый пароль;
* PasswordResetCompleteView – отображает сообщение об успешной смене пароля.

Вышеописанные обработчики экономят время, когда на сайте нужно реализовать аутентификацию. Все они используют классы форм и HTML-шаблоны по умолчанию, но их можно переопределить и воспользоваться собственными.

***Практическое занятие 2 (1 час)***

1. Продолжим работу с приложением *account*.
2. Определим наше приложение в настройках проекта settings.py:

*INSTALLED\_APPS = [ 'django.contrib.admin', 'django.contrib.auth', 'django.contrib.contenttypes',*

*'django.contrib.sessions', 'django.contrib.messages', 'django.contrib.staticfiles', 'hello', 'web', 'account',]*

1. Продолжим работу в папке account/templates файл base.html со следующим кодом:

*<!doctype html><html lang="en"><head><meta charset="utf-8"><title>*

*{% block title %}{% endblock %}</title>{%load static%}*

*<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw7cmDA6j6gD" crossorigin="anonymous">*

*</head><body> <div id="header"> <span class="logo"></span> </div>*

*<div id="content"> {% block content %}{% endblock %} </div></body></html>*

Создадим в папке account/templates файл register.html со следующим кодом:

*{% extends "base.html" %}{% block title %}Регистрация{% endblock %}{% block content %}*

*<h1>Регистрация нового пользователя</h1> <form action="." method="post"> {{ form.as\_p }}*

*{% csrf\_token %}<p><input type="submit" value="Cоздать аккаунт"></p></form>{% endblock %}*

1. Далее необходимо добавить в файл account/forms.py следующий код:

*from django import forms*

*class LoginForm(forms.Form):*

*username = forms.CharField(); password = forms.CharField(widget=forms.PasswordInput)*

1. Далее добавим к уже существующему коду файла account/views.py представления:

*from django.contrib.auth.forms import UserCreationForm*

*from django.urls import reverse\_lazy*

*from django.views import generic*

*class Register(generic.CreateView):*

*form\_class = UserCreationForm; success\_url = reverse\_lazy('login'); template\_name = 'register.html'*

**Итоговый код примет вид:**

*from django.http import HttpResponse*

*from django.shortcuts import render*

*from django.contrib.auth import authenticate, login*

*from .forms import LoginForm*

*from django.contrib.auth.forms import UserCreationForm*

*from django.urls import reverse\_lazy*

*from django.views import generic*

*def user\_login(request):*

*if request.method == 'POST':*

*form = LoginForm(request.POST)*

*if form.is\_valid():*

*cd = form.cleaned\_data*

*user = authenticate(request, username=cd['user'], password=cd['password'])*

*if user is not None:*

*if user.is\_active: login(request, user); return HttpResponse('Authenticated successfully')*

*else: return HttpResponse('Disabled account')*

*else: return HttpResponse('Invalid login')*

*else: form = LoginForm(); return render(request, 'login.html', {'form': form})*

*class Register(generic.CreateView):*

*form\_class = UserCreationForm; success\_url = reverse\_lazy('login'); template\_name = 'register.html'*

1. Откроем urls.py в папке проекта my\_project и добавим следующий код:

*from django.contrib import admin*

*from django.urls import path, include*

*urlpatterns = [ path('admin/', admin.site.urls), path('', include('hello.urls')),*

*path('web/', include('web.urls')), path('account/', include('account.urls')),]*

|  |  |
| --- | --- |
| 7) Далее создадим urls.py в папке account, который содержит следующий код: | *from django.urls import path*  *from account import views*  *from account.views import Register*  *urlpatterns=[path('login/',views.user\_login,name='login'),*  *path('register/',Register.as\_view(),name='register'),]* |
| 8) Запустив сервер и перейдя в 127.0.0.1:8000/account/register, увидим следующее:  Рис. 3. Страница регистрации |  |

***Практическое занятие 3 (2 часа)***

1. Перейдем в каталог с проектом project\_django.
2. Продолжим работу с приложением *account*.
3. При создании проекта Django автоматически создаёт базу данных sqlite3. Работа с проектом Django это – простота подключения базы данных и возможность работать с любыми базами с помощью одного и того же кода. Django автоматически создал для нас файл models.py. Каждая модель представляет отдельную таблицу в базе.

Структура проекта с файлом базы данных db.sqlite3:

|  |  |
| --- | --- |
| В Django встроена подсистема миграций, отслеживающая изменения моделей и транслирующая их в базу данных. Команда migrate применяет все миграции для всех приложений из списка INSTALLED\_APPS. Она изменяет базу данных с учетом моделей и созданных миграций.  В корневом каталоге проекта выполните следующую команду: *python manage.py migrate* | Риc. 4. Дерево проекта |
| Рис. 5 Окно терминала с успешной миграцией  4) Создадим пользователя для управления сайтом администрирования. Выполните следующую команду:  *python manage.py createsuperuser*  Видим запрос логина, электронной почты и пароля для |  |

нового пользователя. Создадим нового пользователя - администратора.

1. Запустим локальный сервер с терминала: *python manage.py runserver*

|  |  |  |
| --- | --- | --- |
| Перейдя в адрес 127.0.0.1:8000/admin, увидим:  Рис. 6 Вход в систему администрирования: | |  |
| Введем логин и пароль:  Рис. 7. Меню администратора |  | |
| Рис. 8. Создание нового пользователя с помощью панели администратора |  | |
| Рис. 9. Пользователь добавлен |  | |
| 6)Проведем проверку авторизации нового пользователя в браузере  Рис. 10. Вид приложения |  | |

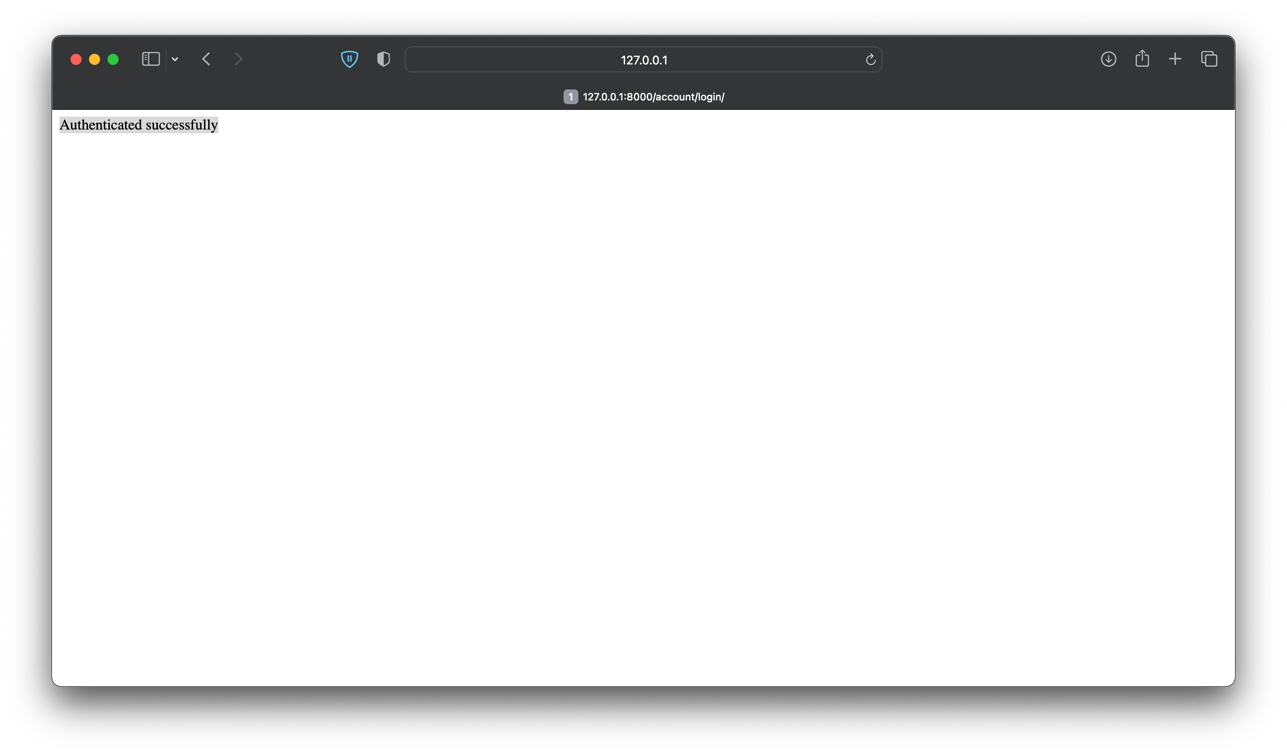


Рис. 11. Успешная авторизация пользователя

7)Проведем проверку регистрации нового пользователя

|  |  |
| --- | --- |
| Рис. 12 Регистрация нового пользователя  Проверим панель администрирования: http://localhost:8000/admin/auth/user/ – учетная запись user2 появилась, значит, форма работает правильно. |  |
| Рис. 13. Вид приложения | |

8)Добавим новому пользователю дополнительную информацию имя и почту

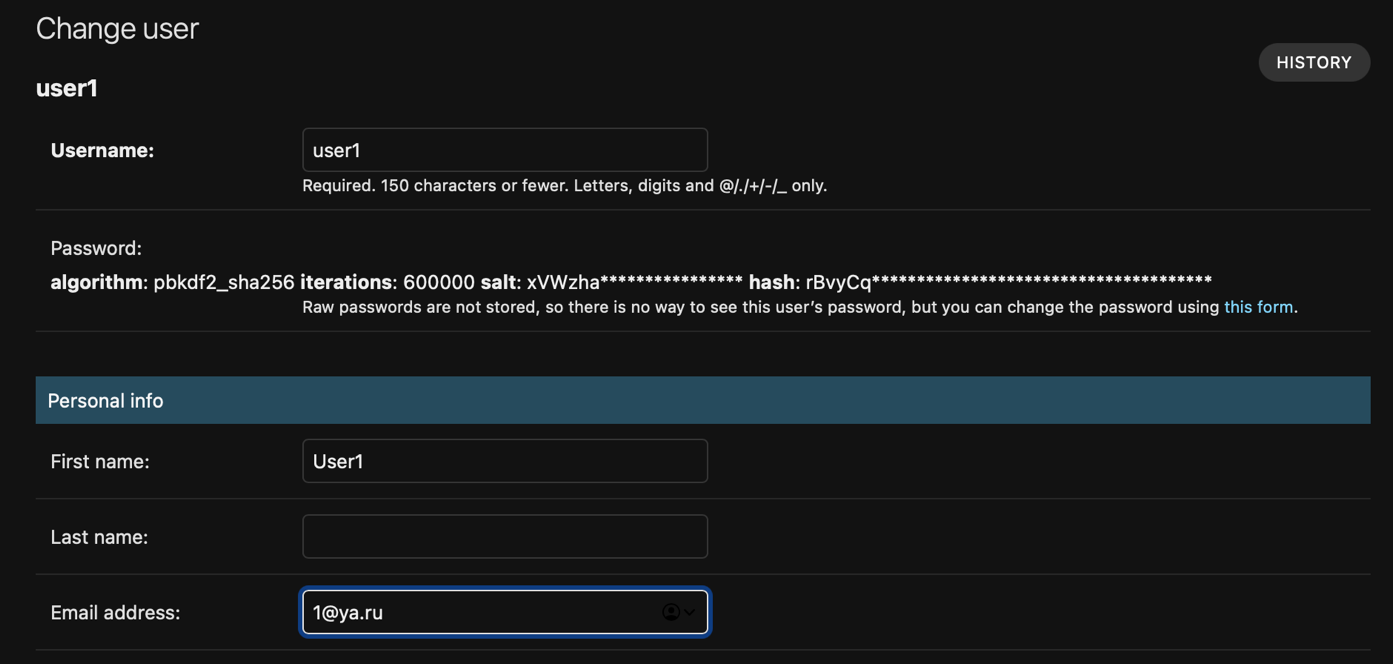


Рис. 14. Добавление нового пользователя

Одно из главных преимуществ системы авторизации – возможность показывать определенный контент только зарегистрированным пользователям.

1. Создадим html страницу profile.html. Для отображения профиля пользователя.

*{% extends "base.html" %}{% block content %} <div class="content-section">*

*<div class="media-body"><h2 class="account-heading">{{ user.username }}</h2>*

*<p class="text-secondary">{{user.email}}</div></div></div>{%endblock content%}*

1. Добавим в urls.py импорт модулей и пути к шаблонам входа, выхода и профиля

*from django.urls import path*

*from account import views*

*from account.views import Register*

*from django.contrib.auth import views as auth\_views*

*from . import views as user\_views*

*urlpatterns = [ path('login/', views.user\_login, name='login'),*

*path('login/',auth\_views.LoginView.as\_view(template\_name='login.html'), name='login'),*

*path('register/', Register.as\_view(), name='register'), path('profile/', user\_views.profile, name='profile'),]*

1. Далее необходимо добавить представления в файле account/views.py:

*from django.contrib.auth.decorators import login\_required*

*@login\_required*

*def profile(request): return render(request, 'profile.html')*

1. Откроем файл settings.py и добавим следующие строки кода

*LOGIN\_REDIRECT\_URL = 'profile'; LOGIN\_URL = 'login'*

1. Запустим локальный сервер с терминала: *python manage.py runserver*

|  |  |  |
| --- | --- | --- |
| Перейдя в 127.0.0.1:8000/account/login, увидим:  Рис. 15. Вид приложения |  | |
| После входа Django перенаправит нас на страницу профиля.  Рис. 16. Вид приложения | |  |

**Примеры работы приложений**

|  |  |  |
| --- | --- | --- |
| Рис. 17. Консоль админа. | Рис. 18. Добавим пользователя | Рис. 18. Редактируем профиль |

Эти примеры можно заархивировать и использовать как отчет по работе.