**Lecture 6. Пользовательские фильтры шаблонов и Bootstrap в Django**

Django — это веб-фреймворк на основе Python, который позволяет быстро создавать эффективные веб-приложения. Его также называют фреймворком с включенными батареями, потому что Django предоставляет встроенные функции для всего, включая интерфейс администрирования Django, базу данных по умолчанию — SQLlite3 и т. д.

**Что такое фильтры на языке шаблонов Django (DTL)?**

Прежде чем перейти к тому, как создавать собственные фильтры на языке шаблонов Django, давайте узнаем, что такое фильтры в Django.

* Фильтр также является важной функцией нашего языка шаблонов.
* Фильтры в основном используются для изменения или фильтрации ваших переменных.
* Вы должны использовать этот символ трубы ( | ), чтобы применить фильтр к любым переменным.

Например :-

Таким образом, этот фильтр изменит значение этой переменной в нижнем регистре.

{{имя\_переменной | имя\_фильтра }} или {{ variable\_name | filter\_name }}

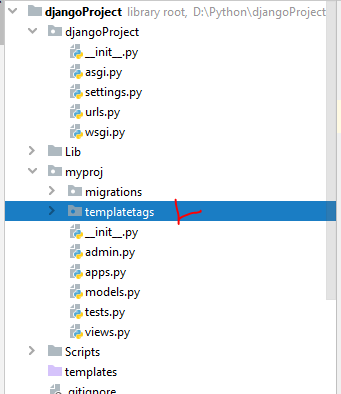
**Как создать собственный фильтр шаблонов в Django?**

Сначала создайте проект django,

Добавьте это представление в свой **views.py**

|  |
| --- |
| **from** django.shortcuts **import** render  *# Create your views here.* **def** home(request):  value = **"HELLO KFU"  return** render(request, **"home.html"**, {**"value"**: value}) |

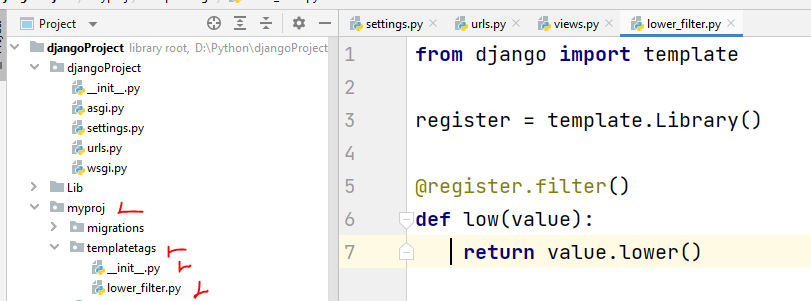
Теперь давайте создадим каталог **templatetags** внутри нашей **основной** папки, т.е. myproj .



и не забудьте создать файл **\_\_init\_\_.py** внутри каталога templatetags

а затем создайте файл **lower\_filter.py**

|  |
| --- |
| **from** django **import** template  register = template.Library()  @register.filter() **def** low(value):  **return** value.lower() |



Создайте каталог в основном каталоге, назовите его как **templates если он не существует.**

Внутри каталога **шаблонов** создайте файл и назовите его **home.html.**

HTML

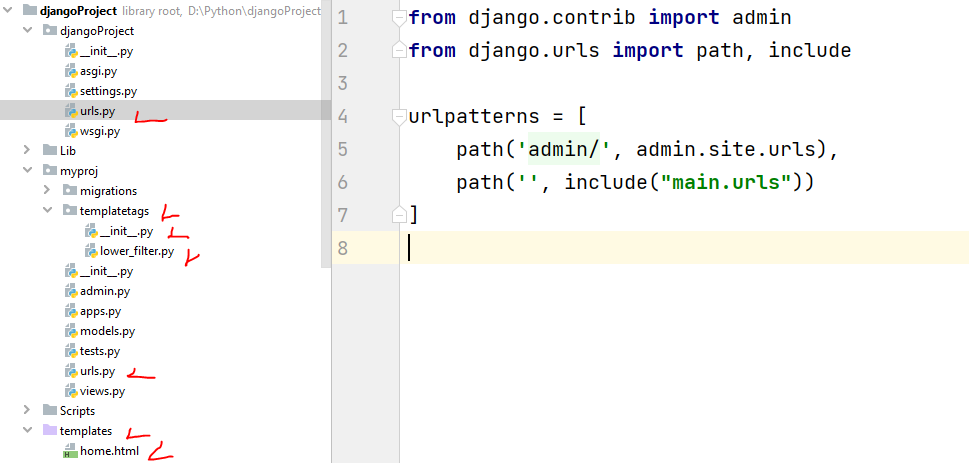
|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**title**>Welcome to KFU</**title**> </**head**> <**body**>  {% **load lower\_filter** %}  <**h1**>{{**value**|**low**}}</**h1**> </**body**> </**html**> |

Создайте файл в myproj и назовите его **urls.py.**

|  |
| --- |
| **from** django.urls **import** path **from** .views **import** \*   urlpatterns = [  path(**''**, home,name=**"home"**), ] |

**В главном urls.py**

|  |
| --- |
| **from** django.contrib **import** admin **from** django.urls **import** path, include  urlpatterns = [  path(**'admin/'**, admin.site.urls),  path(**''**, include(**"main.urls"**)) ] |



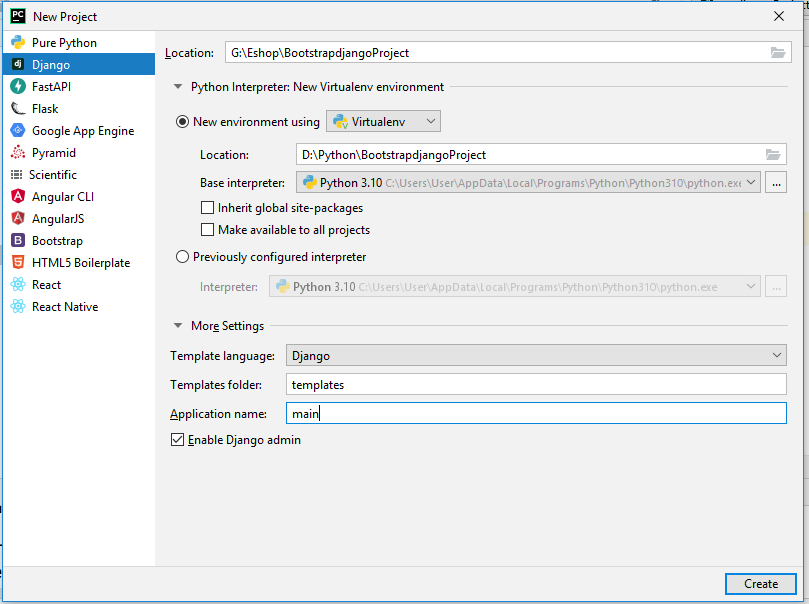
Запустите проект



**Django и формы Bootstrap 4**

Краткое руководство как начать использовать **django-crispy-forms**. Crispy-forms — это замечательное приложение, которое дает контроль над отображением форм **Django** не ломаю их обычную работу. Эта статья оринетирована на использование **Bootstrap** 4-ой версии, но также может быть применена и к старым версиям **Bootstrap**.

Основная причина использования **Crispy-forms** в своих проектах — это простое отображение форм **Django** без написания лишнего кода и с минимальными установками, что очень упрощает работу.



**Установка**

Устанавливаем с использованием pip:

|  |
| --- |
| pip install django-crispy-forms |

Добавляем приложение в INSTALLED\_APPS и выбираем, какой стиль будем использовать:

**settings.py**

|  |
| --- |
| INSTALLED\_APPS = [  **'django.contrib.admin'**,  **'django.contrib.auth'**,  **'django.contrib.contenttypes'**,  **'django.contrib.sessions'**,  **'django.contrib.messages'**,  **'django.contrib.staticfiles'**,  **'application.apps.ApplicationConfig'**,  **'crispy\_forms'**, ] CRISPY\_TEMPLATE\_PACK = **'bootstrap4'** |

**Установка Bootstrap**

Можно загрузить последнюю версию **Bootstrap 4** с [сайта](http://getbootstrap.com/) и использовать готовые скомпилированные CSS и JS файлы.

Или можно использовать CDN, без необходимости устанавливать библиотеку на свой сайт:

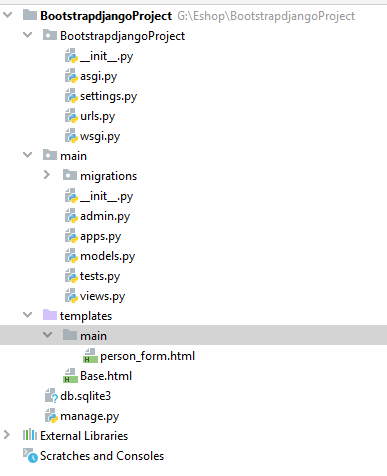
|  |
| --- |
| <**link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous"**> <**script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js" integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEULTy" crossorigin="anonymous"**></**script**> |

Базовый шаблон base.html, который мы будем использовать в этом руководстве:

Templates/base.html

|  |
| --- |
| <!doctype **html**> <**html lang="en"**> <**head**>  <**meta charset="utf-8"**>  <**meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no"**>  <**link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"  integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous"**>  <**title**>Django People</**title**> </**head**> <**body**> <**div class="container"**>  <**div class="row justify-content-center"**>  <**div class="col-8"**>  <**h1 class="mt-2"**>Django People</**h1**>  <**hr class="mt-0 mb-4"**>  {% **block content** %}  {% **endblock** %}  </**div**>  </**div**> </**div**> </**body**> </**html**> |

(В этом примере указан только файл со стилями, т.к. для отображения здесь форм JavaScipt не нужен.)



**Базовый пример**

Предположим, есть следующая модель Person:

**models.py**

|  |
| --- |
| **from** django.db **import** models   **class** Person(models.Model):  name = models.CharField(max\_length=130)  email = models.EmailField(blank=**True**)  job\_title = models.CharField(max\_length=30, blank=**True**)  bio = models.TextField(blank=**True**) |

Теперь создадим отображение для создания объекта Person. Будем использовать встроенный CreateView:

**views.py**

|  |
| --- |
| **from** django.views.generic **import** CreateView **from** .models **import** Person   **class** PersonCreateView(CreateView):  model = Person  fields = (**'name'**, **'email'**, **'job\_title'**, **'bio'**) |

В данном случае **Django** будет искать файл шаблона по пути people/person\_form.html, т.к. наше приложение называется main.

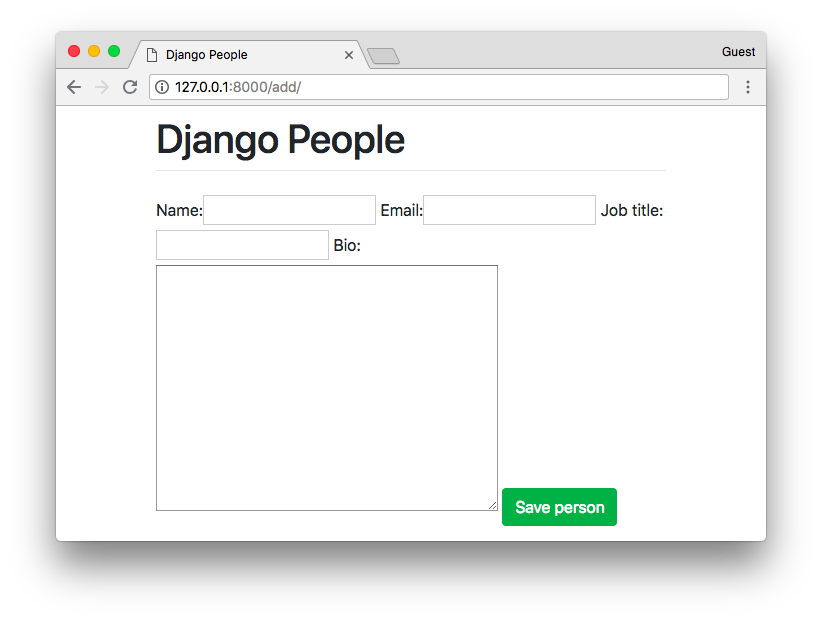
**main/person\_form.html**

|  |
| --- |
| {% **extends 'base.html'** %}  {% **block content** %}  <**form method="post"**>  {% **csrf\_token** %}  {{ **form** }}  <**button type="submit" class="btn btn-success"**>Save person</**button**>  </**form**> {% **endblock** %} |

Изменим файли urls следующим образом:

|  |
| --- |
| **from** django.urls **import** path  **from** application.views **import** PersonCreateView, PersonUpdateView  urlpatterns = [  path(**''**, PersonCreateView.as\_view(), name=**'person\_add'**),  ] |

В этом простейшем варианте, **Django** отобразит форму следующим образом:

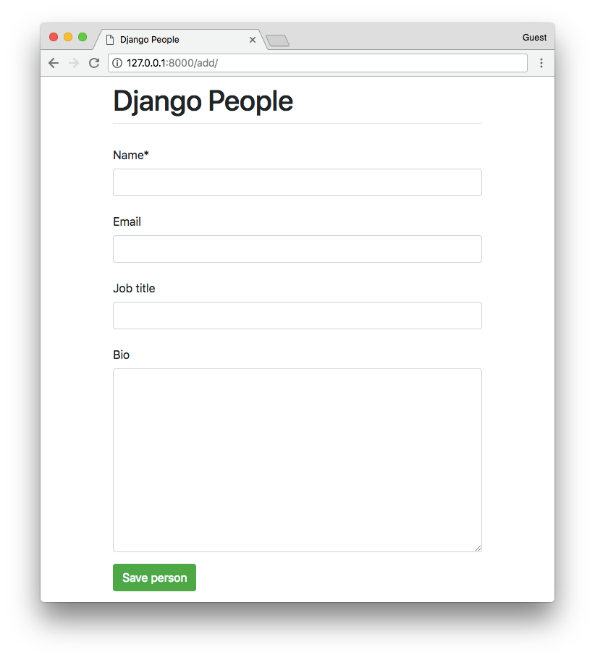


Для отображения этой формы с использованием возможностей **Bootstrap 4** подключаем библиотеку **django-crispy-forms**:

**main/person\_form.html**

|  |
| --- |
| {% **extends 'base.html'** %}  {% **load crispy\_forms\_tags** %}  {% **block content** %}  <**form method="post" novalidate**>  {% **csrf\_token** %}  {{ **form**|**crispy** }}  <**button type="submit" class="btn btn-success"**>Save person</**button**>  </**form**> {% **endblock** %} |

Результат выглядит уже лучше:

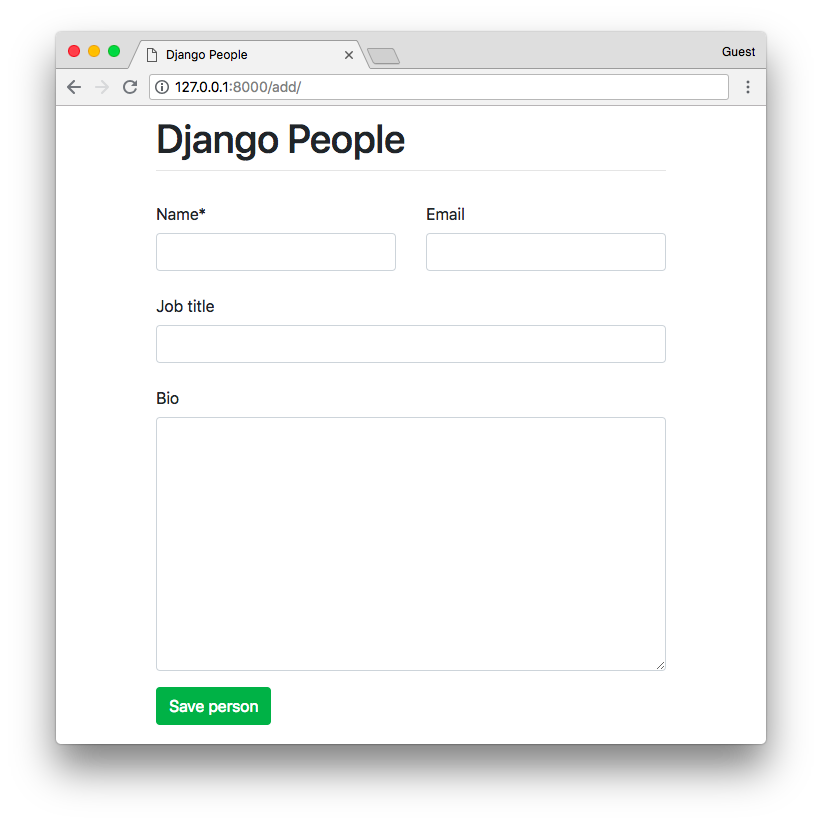


В случае, если необходимо отображать поля "вручную", то можно использовать шаблонный фильтр as\_crispy\_field:

**main/person\_form.html**

|  |
| --- |
| {% **extends 'base.html'** %}  {% **load crispy\_forms\_tags** %}  {% **block content** %}  <**form method="post" novalidate**>  {% **csrf\_token** %}  <**div class="row"**>  <**div class="col-6"**>  {{ **form**.**name**|**as\_crispy\_field** }}  </**div**>  <**div class="col-6"**>  {{ **form**.**email**|**as\_crispy\_field** }}  </**div**>  </**div**>  {{ **form**.**job\_title**|**as\_crispy\_field** }}  {{ **form**.**bio**|**as\_crispy\_field** }}  <**button type="submit" class="btn btn-success"**>Save person</**button**>  </**form**> {% **endblock** %} |

И в результате получим следующую картину:



**Дополнительные возможности**

**django-crispy-forms** имеет специальный класс FormHelper, который дает нам полный контроль над отображением своих форм.

Посмотрим на наш пример и функцию обновления объекта:

**forms.py**

|  |
| --- |
| **from** django **import** forms **from** crispy\_forms.helper **import** FormHelper **from** crispy\_forms.layout **import** Submit **from** people.models **import** Person  **class** PersonForm(forms.ModelForm):  **class** Meta:  model = Person  fields = (**'name'**, **'email'**, **'job\_title'**, **'bio'**)   **def** \_\_init\_\_(self, \*args, \*\*kwargs):  super().\_\_init\_\_(\*args, \*\*kwargs)  self.helper = FormHelper()  self.helper.form\_method = **'post'** self.helper.add\_input(Submit(**'submit'**, **'Save person'**)) |

Вся работа происходит в методе \_\_init\_\_(). остальное — это обычная форма Django на основе модели. Здесь указано, что форма должна использовать для отправки метод POST, а также содержит кнопку отправки 'Save person'.

**Добавим в файле views.py**

|  |
| --- |
| **from** django.views.generic **import** UpdateView **from** .models **import** Person **from** .form **import** PersonForm  **class** PersonUpdateView(UpdateView):  model = Person  form\_class = PersonForm  template\_name = **'people/person\_update\_form.html'** |

В результате получим

|  |
| --- |
| **from** django.views.generic **import** CreateView, UpdateView **from** .models **import** Person **from** main.forms **import** PersonForm  **class** PersonCreateView(CreateView):  model = Person  fields = (**'name'**, **'email'**, **'job\_title'**, **'bio'**)   **class** PersonUpdateView(UpdateView):  model = Person  form\_class = PersonForm  template\_name = **'main/person\_update\_form.html'** |

**main/person\_update\_form.html**

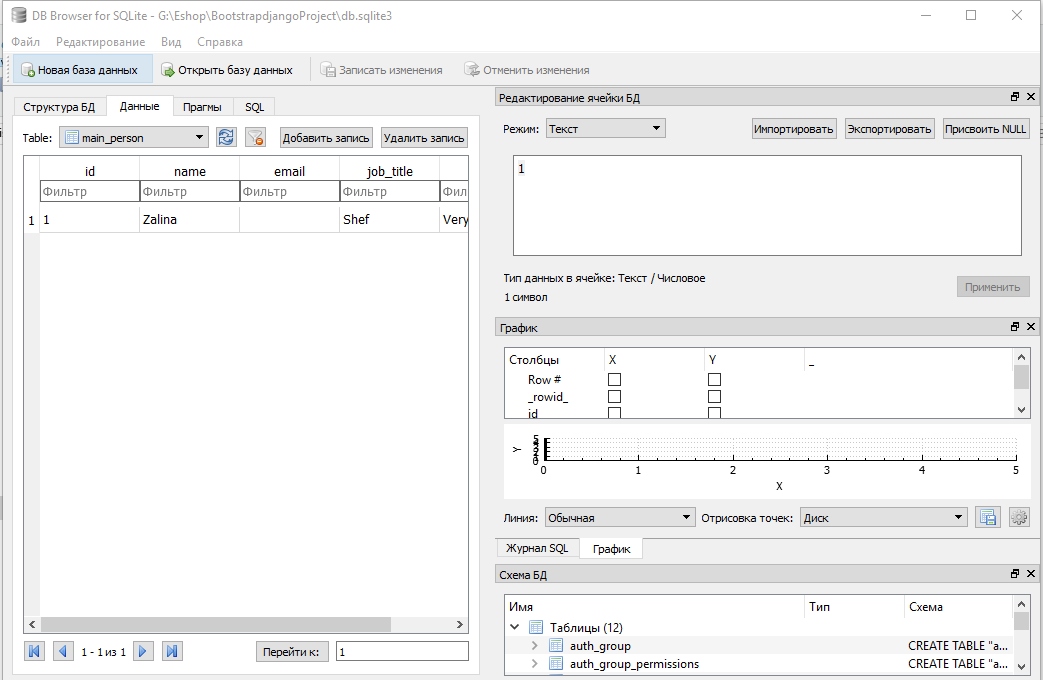
|  |
| --- |
| {% **extends 'base.html'** %}  {% **load crispy\_forms\_tags** %}  {% **block content** %}  {% **crispy form** %} {% **endblock** %} |

Здесь мы просто вызываем шаблонный тег {% crispy %} и передаем экземпляр формы как аргумент.

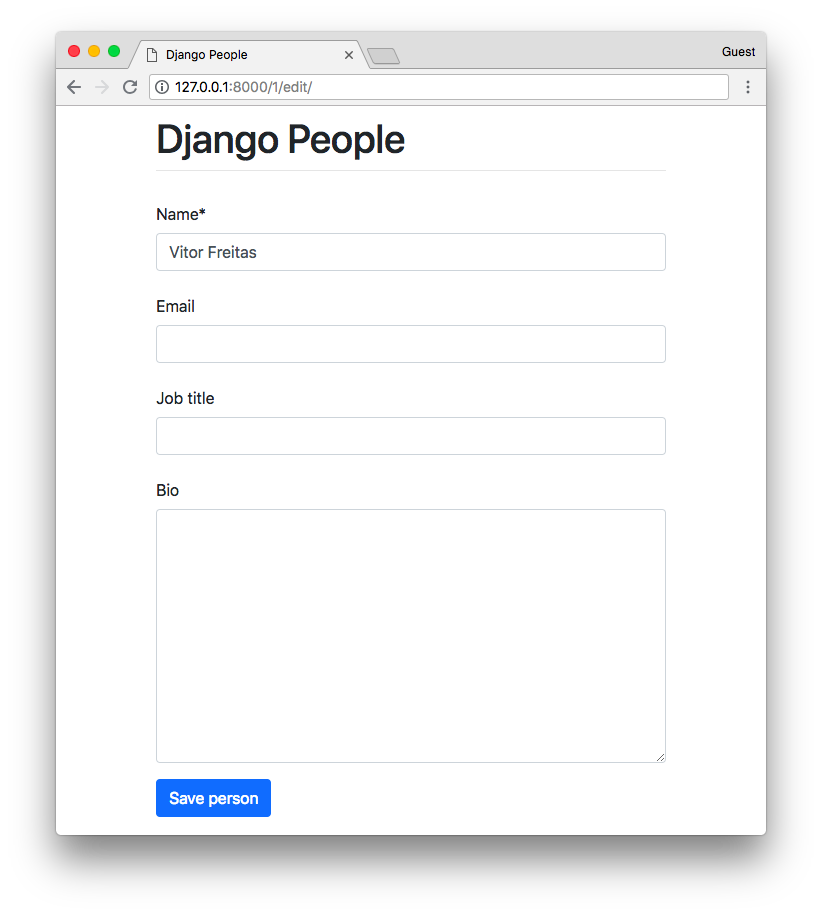
Выполнить миграцию:

|  |
| --- |
| python manage.py makemigrations  python manage.py migrate |

Откроем БД и добавим запись



Это все, что нужно для отображения следующей формы:



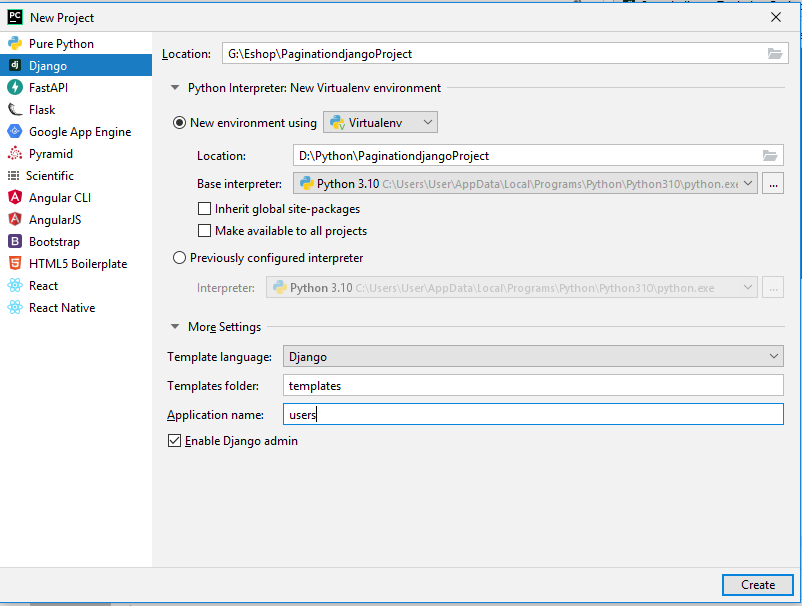
Это были самые простые примеры использования библиотеки **django-crispy-forms**. Если необходим полный контроль над работой с формой и ее отображением, то воспользуйтесь документацией на django-crispy-forms.readthedocs.io [ <https://django-crispy-forms.readthedocs.io/en/latest/>] .

Изменим файле urls следующим образом:

|  |
| --- |
| **from** django.urls **import** path  **from** application.views **import** PersonCreateView, PersonUpdateView  urlpatterns = [  path(**'add/'**, PersonCreateView.as\_view(), name=**'person\_add'**),  path(**'<int:pk>/edit/'**, PersonUpdateView.as\_view(), name=**'person\_edit'**), ] |

**Как разбить на страницы с Django**

Как часть общих инструментов веб-приложений Django, Django предлагает несколько классов для управления данными с разбивкой на страницы. Вы можете передать либо список/кортеж объектов, либо QuerySet. В этом руководстве я покажу, как разбивать данные на страницы с помощью представлений на основе функций и как разбивать на страницы с помощью представлений на основе классов (ListView).



**Пагинатор**

Классы paginator живут в django.core.paginator. Мы будем работать в основном с классами Paginator и Page

Давайте рассмотрим простую примеру.

В файле **models/Person.py** определена модель Person:

|  |
| --- |
| **from** django.db **import** models   **class** Person(models.Model):  name = models.CharField(max\_length=20)  age = models.IntegerField() |

Вначале необходимо создать миграцию с помощью команды

|  |
| --- |
| python manage.py makemigrations  python manage.py migrate |

В файле models/User.py определена модель User:

|  |
| --- |
| **class** User:  **def** \_\_init\_\_(self, name, age):  self.name = name *# имя человека* self.age = age *# возраст человека* UserList = [] UserList.append(User(**"Александрова Оксана Анатольевна"**, 18)) UserList.append(User(**"Бабамурадов Султан Нариманович"**, 18)) UserList.append(User(**"Бариев Ильнар Ирекович"**, 18)) UserList.append(User(**"Богуславская Юлия Александровна"**, 18)) UserList.append(User(**"Гараева Анна Андреевна"**, 18)) UserList.append(User(**"Гимадеева Алсу Айратовна"**, 18)) UserList.append(User(**"Дзеник Мария Дмитриевна"**, 18)) UserList.append(User(**"Зайцева Александра Евгеньевна"**, 18)) UserList.append(User(**"Ильдерова Анна Ивановна"**, 18)) UserList.append(User(**"Курбанмаммедов Курбанмаммет Куватбердиевич"**, 18)) UserList.append(User(**"Мингазова Альфия Ленаровна"**, 18)) UserList.append(User(**"Мухтаров Ибрагим Ильдарович"**, 18)) UserList.append(User(**"Назипова Алиса Фанисовна"**, 18)) UserList.append(User(**"Никашкин Михаил Дмитриевич"**, 18)) UserList.append(User(**"Сагидуллин Артём Эдуардович"**, 18)) UserList.append(User(**"Сабитова Регина Шамилевна"**, 18)) UserList.append(User(**"Сайфуллина Диляра Фаритовна"**, 18)) UserList.append(User(**"Салахова Эльвина Фаниловна"**, 18)) UserList.append(User(**"Фатыхова Энже Булатовна"**, 18)) UserList.append(User(**"Хадеева Зухра Ильдаровна"**, 18)) UserList.append(User(**"Хазеев Рашит Айратович"**, 18)) UserList.append(User(**"Билалетдинова Сабина Айратовна"**, 18)) UserList.append(User(**"Гайнетдинова Мадина Вадимовна"**, 18)) UserList.append(User(**"Галеева Алиса Эдуардовна"**, 18)) UserList.append(User(**"Ганиева Радмила Айратовна"**, 18)) UserList.append(User(**"Гринько Дарья Романовна"**, 18)) UserList.append(User(**"Ибрагимова Залина Маратовна"**, 18)) UserList.append(User(**"Карпова Ксения Сергеевна"**, 18)) UserList.append(User(**"Клюшкарёва Евгения "**, 18)) UserList.append(User(**"Козырева Анастасия Романовна"**, 18)) UserList.append(User(**"Миннигалиев Тимур Уралович"**, 18)) UserList.append(User(**"Мугинова Алина Альбертовна"**, 18)) UserList.append(User(**"Мударисова Аделина Марселовна"**, 18)) UserList.append(User(**"Муллин Айдар Рустямович"**, 18)) UserList.append(User(**"Насибуллина Ришатовна"**, 18)) UserList.append(User(**"Ольховиков Дмитрий Александрович"**, 18)) UserList.append(User(**"Пепеляева Алиса Андреевна"**, 18)) UserList.append(User(**"Репина Полина Александровна"**, 18)) UserList.append(User(**"Садреева Эльвира Ильнуровна"**, 18)) UserList.append(User(**"Садыкова Аделина Руслановна"**, 18)) UserList.append(User(**"Сапачева Алина Сергеевна"**, 18)) UserList.append(User(**"Субботина Аделина Олеговна"**, 18)) UserList.append(User(**"Хайдарова Сулгун Сапаевна"**, 18)) UserList.append(User(**"Хамидуллин Ильназ Гараевич"**, 18)) UserList.append(User(**"Ширяев Александр Владиславович"**, 18)) UserList.append(User(**"Щербакова Арина Витальевна"**, 18)) |

В файле views.py пропишем два представления для получения данных из бд и для сохранения данных:

|  |
| --- |
| **from** django.shortcuts **import** render **from** django.http **import** HttpResponseRedirect  **from** users.Models.Person **import** Person **from** users.Models.User **import** UserList   *# получение данных из бд* **def** index(request):  people = Person.objects.all()  **return** render(request, **"index.html"**, {**"people"**: people})   *# сохранение данных в бд* **def** create(request):  **if** request.method == **"POST"**:  annya = Person()  *# annya.name = request.POST.get("name")  # annya.age = request.POST.get("age")* **for** i **in** range(len(UserList)):  annya = Person()  annya.name=UserList[i].name  annya.age=UserList[i].age  annya.save()   **return** HttpResponseRedirect(**"/"**) |

В функции index() получаем все данные с помощью метода Person.objects.all() и передаем их в шаблон index.html.

В функции create() получаем данные из запроса типа POST, сохраняем данные с помощью метода save() и выполняем переадресацию на корень веб-сайта (то есть на функцию index).

В папке templates определим шаблон **index.html**, который будет выводить данные на веб-страницу:

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"** />  <**title**>Модели в Django</**title**> </**head**> <**body class="container"**>  <**form method="POST" action="create/"**>  {% **csrf\_token** %}  <**p**>  <**label**>Введите имя</**label**><**br**>  <**input type="text" name="name"** />  </**p**>  <**p**>  <**label**>Введите возраст</**label**><**br**>  <**input type="number" name="age"** />  </**p**>  <**input type="submit" value="Сохранить"** >  </**form**>  {% **if people**.**count** > 0 %}  <**h2**>Список пользователей</**h2**>  <**table**>  <**tr**><**th**>Id</**th**><**th**>Имя</**th**><**th**>Возраст</**th**></**tr**>  {% **for person in people** %}  <**tr**><**td**>{{ **person**.**id** }}</**td**><**td**>{{ **person**.**name** }}</**td**><**td**>{{ **person**.**age** }}</**td**></**tr**>  {% **endfor** %}  </**table**>  {% **endif** %} </**body**> </**html**> |

В начале шаблона определена форма для добавления данных, которые потом будет получать функция create в POST-запросе. А ниже определена таблица, в которую выводятся данные из переданного из представления набора people.

И также в файле **urls.py** свяжем маршруты с представлениями:

|  |
| --- |
| **from** django.urls **import** path **from** users **import** views  urlpatterns = [  path(**''**, views.index),  path(**'create/'**, views.create), ] |

Запустим проект и обратимся к приложению в браузере

Person таблица имеет 46 пользовательских экземпляра.

|  |
| --- |
| from django.contrib.auth.models import User  from django.core.paginator import Paginator  user\_list = Person.objects.all()  paginator = Paginator(user\_list, 10) |

В приведенном выше примере я предлагаю Paginator разбивать user\_list QuerySet на страницы по 10 страниц. Это создаст 5 - страничный результат. Первые 4 страниц с 10 пользователями на каждой и последняя страница с 6 пользователями.

Отладка объекта Paginator

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Type** |
| paginator.**count** | **46** | <type 'int'> |
| paginator.**num\_pages** | **5** | <type 'int'> |
| paginator.**page\_range** | **xrange(1, 6)** | <type 'xrange'> |
| paginator.**page(2)** | **<Page 2 of 5>** | <class 'django.core.paginator.Page'> |

Метод Paginator.page()вернет заданную страницу результатов с разбивкой на страницы, которая является экземпляром Page. Это то, что мы вернемся к шаблону.

|  |
| --- |
| users = paginator.page(2) |

Отладка объекта страницы

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Type** |
| Users | **<Page 2 of 6>** | <class 'django.core.paginator.Page'> |
| users.**has\_next()** | **True** | <type 'bool'> |
| users.**has\_previous()** | **True** | <type 'bool'> |
| users.**has\_other\_pages()** | **True** | <type 'bool'> |
| users.**next\_page\_number()** | **3** | <type 'int'> |
| users.**previous\_page\_number()** | **1** | <type 'int'> |
| users.**start\_index()** | **11** | <type 'int'> |
| users.**end\_index()** | **20** | <type 'int'> |

Методы Page.next\_page\_number()и Page.previous\_page\_number()вызывают InvalidPage, если следующая/предыдущая страница не существует.

И относятся Page.start\_index()к Page.end\_index()номеру страницы.

|  |
| --- |
| >>> users = paginator.page(6) # last page  <Page 6 of 6>  >>> users.start\_index()  41  >>> users.end\_index()  46 |

Процесс в основном выполняется путем запроса базы данных, затем передачи QuerySet в Paginator, захвата страницы и возврата к шаблону. Остальное делается в шаблоне.

Давайте теперь посмотрим на некоторые практические примеры.

Разбиение на страницы с функциональными представлениями

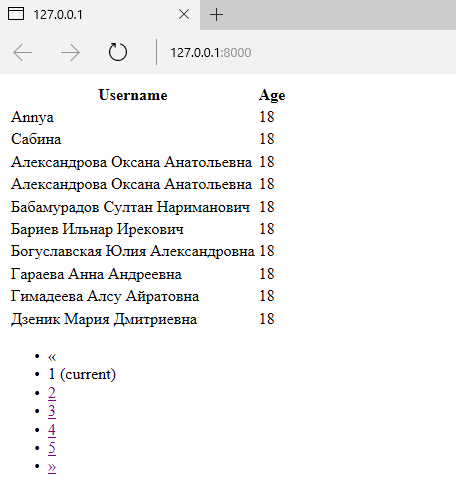
views.py

|  |
| --- |
| **from** django.shortcuts **import** render **from** users.Models.Person **import** Person **from** django.core.paginator **import** Paginator, EmptyPage, PageNotAnInteger   **def** index(request):  user\_list = Person.objects.all()  page = request.GET.get(**'page'**, 1)   paginator = Paginator(user\_list, 10)  **try**:  users = paginator.page(page)  **except** PageNotAnInteger:  users = paginator.page(1)  **except** EmptyPage:  users = paginator.page(paginator.num\_pages)   **return** render(request, **'user\_list.html'**, {**'users'**: users}) |

**user\_list.html**

|  |
| --- |
| <**table class="table table-bordered"**>  <**thead**>  <**tr**>  <**th**>Username</**th**>  <**th**>Age</**th**>  </**tr**>  </**thead**>  <**tbody**>  {% **for user in users** %}  <**tr**>  <**td**>{{ **user**.**name** }}</**td**>  <**td**>{{ **user**.**age** }}</**td**>  </**tr**>  {% **endfor** %}  </**tbody**> </**table**>  {% **if users**.**has\_other\_pages** %}  <**ul class="pagination"**>  {% **if users**.**has\_previous** %}  <**li**><**a href="?page=**{{ **users**.**previous\_page\_number** }}**"**>**&laquo;**</**a**></**li**>  {% **else** %}  <**li class="disabled"**><**span**>**&laquo;**</**span**></**li**>  {% **endif** %}  {% **for i in users**.**paginator**.**page\_range** %}  {% **if users**.**number** == **i** %}  <**li class="active"**><**span**>{{ **i** }} <**span class="sr-only"**>(current)</**span**></**span**></**li**>  {% **else** %}  <**li**><**a href="?page=**{{ **i** }}**"**>{{ **i** }}</**a**></**li**>  {% **endif** %}  {% **endfor** %}  {% **if users**.**has\_next** %}  <**li**><**a href="?page=**{{ **users**.**next\_page\_number** }}**"**>**&raquo;**</**a**></**li**>  {% **else** %}  <**li class="disabled"**><**span**>**&raquo;**</**span**></**li**>  {% **endif** %}  </**ul**> {% **endif** %} |

Результат примерно такой:



В приведенном выше примере используется Bootstrap 4.

С начало в templates создаем файла base.html

|  |
| --- |
| <!doctype **html**> <**html lang="en"**> <**head**>  <**meta charset="utf-8"**>  <**meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no"**>  <**link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"  integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous"**>  <**title**>Django People</**title**> </**head**> <**body**> <**div class="container"**>  <**div class="row justify-content-center"**>  <**div class="col-8"**>  <**h1 class="mt-2"**>List of Students</**h1**>  <**hr class="mt-0 mb-4"**>  {% **block content** %}  {% **endblock** %}  </**div**>  </**div**> </**div**> </**body**> </**html**> |

Устанавливаем Bootstrap4 с использованием pip:

|  |
| --- |
| pip install django-crispy-forms |

Добавляем приложение в INSTALLED\_APPS и выбираем, какой стиль будем использовать:

**settings.py**

|  |
| --- |
| INSTALLED\_APPS = [  **'django.contrib.admin'**,  **'django.contrib.auth'**,  **'django.contrib.contenttypes'**,  **'django.contrib.sessions'**,  **'django.contrib.messages'**,  **'django.contrib.staticfiles'**,  **'application.apps.ApplicationConfig'**,  **'crispy\_forms'**, ] CRISPY\_TEMPLATE\_PACK = **'bootstrap4'** |

Разбиение на страницы с представлениями на основе классов

views.py

|  |
| --- |
| **from** users.Models.Person **import** Person **from** django.views.generic **import** ListView   **class** UserListView(ListView):  model = Person  template\_name = **'user\_list.html'** *# Default: <app\_label>/<model\_name>\_list.html* context\_object\_name = **'users'** *# Default: object\_list* paginate\_by = 10  queryset = Person.objects.all() *# Default: Model.objects.all()* |

**Urls.py**

|  |
| --- |
| **from** django.urls **import** path **from** users.views **import** UserListView  urlpatterns = [  path(**'users/'**, UserListView.as\_view(), name=**'users'**), ] |

**user\_list.html**

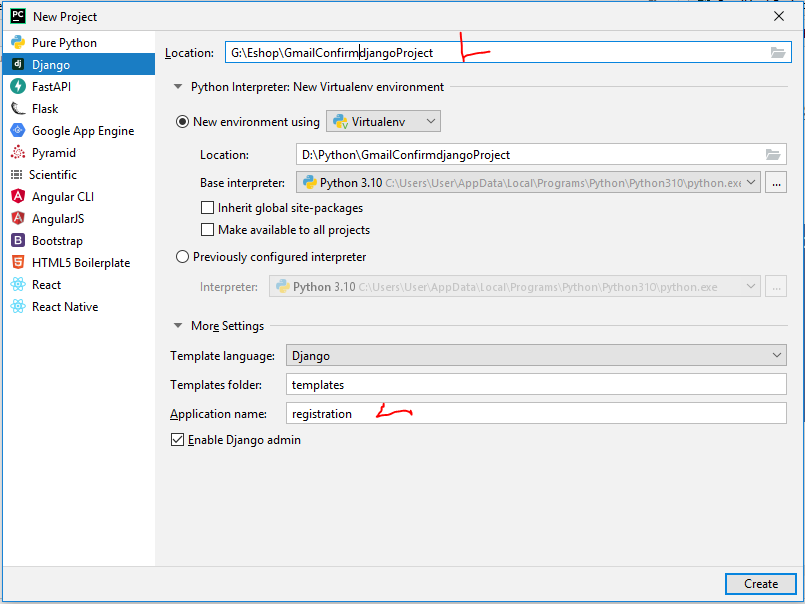
|  |
| --- |
| {% **extends 'base.html'** %} {% **block content** %} <**table class="table table-bordered"**>  <**thead**>  <**tr**>  <**th**>Username</**th**>  <**th**>Age</**th**>  </**tr**>  </**thead**>  <**tbody**>  {% **for user in users** %}  <**tr**>  <**td**>{{ **user**.**name** }}</**td**>  <**td**>{{ **user**.**age** }}</**td**>  </**tr**>  {% **endfor** %}  </**tbody**> </**table**>  {% **if is\_paginated** %}  <**ul class="pagination"**>  {% **if page\_obj**.**has\_previous** %}  <**li**><**a href="?page=**{{ **page\_obj**.**previous\_page\_number** }}**"**>**&laquo;**</**a**></**li**>  {% **else** %}  <**li class="disabled"**><**span**>**&laquo;**</**span**></**li**>  {% **endif** %}  {% **for i in paginator**.**page\_range** %}  {% **if page\_obj**.**number** == **i** %}  <**li class="active"**><**span**>{{ **i** }} <**span class="sr-only"**>(current)</**span**></**span**></**li**>  {% **else** %}  <**li**><**a href="?page=**{{ **i** }}**"**>{{ **i** }}</**a**></**li**>  {% **endif** %}  {% **endfor** %}  {% **if page\_obj**.**has\_next** %}  <**li**><**a href="?page=**{{ **page\_obj**.**next\_page\_number** }}**"**>**&raquo;**</**a**></**li**>  {% **else** %}  <**li class="disabled"**><**span**>**&raquo;**</**span**></**li**>  {% **endif** %}  </**ul**> {% **endif** %}  {% **endblock** %} |

**Регистрация пользователя Django с подтверждением по электронной почте**

Как правило, мы регистрируемся на любом веб-сайте, он отправляет ссылку подтверждения для активации учетной записи. Или иногда просит написать по электронной почте, чтобы изменить адрес электронной почты или сбросить пароль учетной записи.

В этом руководстве мы узнаем, как отправить письмо с подтверждением с помощью Django, когда кто-то регистрируется в нашем веб-приложении. Мы обсудим простой способ построить это. Но Django предоставляет множество других опций, таких как приложение django allauth, django-registration, django-registration-redux. Эти приложения делают эту задачу очень простой.

Прежде чем начать, убедитесь, что вы знаете языки программирования Django и Python. Создайте базовую настройку Django и настройте параметры.



**Настроить параметры**

Сначала нам нужно настроить хост-сервер электронной почты в settings.py для письма с подтверждением. Добавьте приведенную ниже конфигурацию в файл **settings.py.**

|  |
| --- |
| EMAIL\_BACKEND = **'django.core.mail.backends.smtp.EmailBackend'** EMAIL\_USE\_TLS = **True** EMAIL\_HOST = **'smtp.gmail.com'** EMAIL\_HOST\_USER = **'examplekfustudents@gmail.com'** EMAIL\_HOST\_PASSWORD = **'MtrPxe5ACNapNiE'** EMAIL\_PORT = 587 DEFAULT\_FROM\_EMAIL = **'examplekfustudents@gmail.com'** |

Мы использовали идентификатор электронной почты вместе с паролем и хост-сервером SMTP gmail. Вы также можете использовать другой SMTP-сервер.

**Создать токен**

Нам нужно создать токен, который будет использоваться в URL-адресе подтверждения электронной почты. В пакете registration Создайте файл tokens.py и добавьте приведенный ниже код.

registration/tokens.py

|  |
| --- |
| **from** django.contrib.auth.tokens **import** PasswordResetTokenGenerator **from** django.utils **import** six   **class** TokenGenerator(PasswordResetTokenGenerator):  **def** \_make\_hash\_value(self, user, timestamp):  **return** (  six.text\_type(user.pk) + six.text\_type(timestamp) +  six.text\_type(user.is\_active)  )   account\_activation\_token = TokenGenerator() |

Мы использовали класс **PasswordTokenGenerator**, который используется для сброса пароля. В приведенном выше коде мы сгенерировали уникальный токен для подтверждения.

Примечание. Откроете терминал и установите django-utils-six.

|  |
| --- |
| pip install django-utils-six |

Вариант 2

Прежде всего, вам нужно установить шесть с помощью этой команды.

pip3 install six

Затем вам нужно обновить пакет заголовков cors с помощью этой команды.

pip3 install --upgrade django-cors-headers

[six](http://pythonhosted.org/six/) является канонической библиотекой для одновременной поддержки Python 2 и 3!

[Измененная версия six](https://djbook.ru/rel1.7/topics/python3.html#module-django.utils.six) входит в поставку Django начиная с версии 1.4.2. Вы можете импортировать его как django.utils.six.

В нём учитываются наиболее распространенные изменения, что позволит написать совместимый код.

**Создать регистрационную форму**

Django предоставляет встроенный **UserCreationForm** , который является отличным классом для создания формы. Мы создаем файл forms.py в приложении и импортируем класс UserCreationForm. Давайте разберем следующий код.

registration/**forms.py**

|  |
| --- |
| **from** django **import** forms **from** django.contrib.auth.forms **import** UserCreationForm **from** django.contrib.auth.models **import** User   **class** SignupForm(UserCreationForm):  email = forms.EmailField(max\_length=200, help\_text=**'Required'**)   **class** Meta:  model = User  fields = (**'username'**, **'email'**, **'password1'**, **'password2'**) |

В приведенном выше коде мы импортировали форму UserCreationForm и встроенный файл User. Затем мы создали **SignupForm, включив в SignupForm** дополнительное поле электронной почты .

**registration/view.py**

|  |
| --- |
| **from** django.http **import** HttpResponse **from** django.shortcuts **import** render, redirect **from** django.contrib.auth **import** login, authenticate **from** .forms **import** SignupForm **from** django.contrib.sites.shortcuts **import** get\_current\_site **from** django.utils.encoding **import** force\_bytes, force\_str **from** django.utils.http **import** urlsafe\_base64\_encode, urlsafe\_base64\_decode **from** django.template.loader **import** render\_to\_string **from** .tokens **import** account\_activation\_token **from** django.contrib.auth.models **import** User **from** django.core.mail **import** EmailMessage   **def** signup(request):  **if** request.method == **'POST'**:  form = SignupForm(request.POST)  **if** form.is\_valid():  *# save form in the memory not in database* user = form.save(commit=**False**)  user.is\_active = **False** user.save()  *# to get the domain of the current site* current\_site = get\_current\_site(request)  mail\_subject = **'Activation link has been sent to your email id'** message = render\_to\_string(**'acc\_active\_email.html'**, {  **'user'**: user,  **'domain'**: current\_site.domain,  **'uid'**: urlsafe\_base64\_encode(force\_bytes(user.pk)),  **'token'**: account\_activation\_token.make\_token(user),  })  to\_email = form.cleaned\_data.get(**'email'**)  email = EmailMessage(  mail\_subject, message, to=[to\_email]  )  email.send()  **return** HttpResponse(**'Please confirm your email address to complete the registration'**)  **else**:  form = SignupForm()  **return** render(request, **'signup.html'**, {**'form'**: form}) |

Здесь мы создаем представление регистрации, оно получает информацию с помощью метода POST и проверяет ее. Мы использовали commit = False, потому что это позволяет нам получить объект модели и добавить дополнительный атрибут. Здесь мы сделали **user.is\_active = False** , что означает, что пользователь не может войти в систему, пока адрес электронной почты не будет проверен.

Затем мы использовали **функцию EmailMessage()** для отправки почты вместе с темой сообщения. Сообщение электронной почты, созданное по шаблону.

**templates/acc\_active\_email.html**

|  |
| --- |
| {% **autoescape off** %}  Hi {{ **user**.**username** }},  Please click on the link to confirm your registration,  http://{{ **domain** }}{% **url 'activate' uidb64**=**uid token**=**token** %} {% **endautoescape** %} |

Этот шаблон создает тело электронной почты со ссылкой активации, которая будет отправлена ​​для приложения.

Теперь нам нужно создать представление для ссылки активации.

**Создать представление активации**

После того, как пользователь нажмет на ссылку активации, нам нужно активировать его учетную запись с помощью ссылки активации. Этот процесс обрабатывается представлением **активации**.

**В конце файла registration/views.py добавим следующего кода.**

|  |
| --- |
| **def** activate(request, uidb64, token):  User = get\_user\_model()  **try**:  uid = force\_str(urlsafe\_base64\_decode(uidb64))  user = User.objects.get(pk=uid)  **except**(TypeError, ValueError, OverflowError, User.DoesNotExist):  user = **None  if** user **is not None and** account\_activation\_token.check\_token(user, token):  user.is\_active = **True** user.save()  **return** HttpResponse(**'Thank you for your email confirmation. Now you can login your account.'**)  **else**:  **return** HttpResponse(**'Activation link is invalid!'**) |

В результате он примет следующий вид:

|  |
| --- |
| **from** django.http **import** HttpResponse **from** django.shortcuts **import** render, redirect **from** django.contrib.auth **import** login, authenticate, get\_user\_model **from** .forms **import** SignupForm **from** django.contrib.sites.shortcuts **import** get\_current\_site **from** django.utils.encoding **import** force\_bytes, force\_str **from** django.utils.http **import** urlsafe\_base64\_encode, urlsafe\_base64\_decode **from** django.template.loader **import** render\_to\_string **from** .tokens **import** account\_activation\_token **from** django.contrib.auth.models **import** User **from** django.core.mail **import** EmailMessage   **def** signup(request):  **if** request.method == **'POST'**:  form = SignupForm(request.POST)  **if** form.is\_valid():  *# save form in the memory not in database* user = form.save(commit=**False**)  user.is\_active = **False** user.save()  *# to get the domain of the current site* current\_site = get\_current\_site(request)  mail\_subject = **'Activation link has been sent to your email id'** message = render\_to\_string(**'acc\_active\_email.html'**, {  **'user'**: user,  **'domain'**: current\_site.domain,  **'uid'**: urlsafe\_base64\_encode(force\_bytes(user.pk)),  **'token'**: account\_activation\_token.make\_token(user),  })  to\_email = form.cleaned\_data.get(**'email'**)  email = EmailMessage(  mail\_subject, message, to=[to\_email]  )  email.send()  **return** HttpResponse(**'Please confirm your email address to complete the registration'**)  **else**:  form = SignupForm()  **return** render(request, **'signup.html'**, {**'form'**: form})   **def** activate(request, uidb64, token):  User = get\_user\_model()  **try**:  uid = force\_str(urlsafe\_base64\_decode(uidb64))  user = User.objects.get(pk=uid)  **except**(TypeError, ValueError, OverflowError, User.DoesNotExist):  user = **None  if** user **is not None and** account\_activation\_token.check\_token(user, token):  user.is\_active = **True** user.save()  **return** HttpResponse(**'Thank you for your email confirmation. Now you can login your account.'**)  **else**:  **return** HttpResponse(**'Activation link is invalid!'**) |

Мы добавили функцию активации после функции регистрации. Это представление проверит токен на действительность, после чего пользователь активирует и войдет в систему. Мы устанавливаем **user.is\_active = True** , что означает, что пользователь может войти в систему.

Теперь мы сопоставим представления с URL-адресами.

**URL-адреса**

**urls.py**

|  |
| --- |
| **from** django.urls **import** path **from** authorization.views **import** activate, signup urlpatterns = [  path(**''**, signup, name = **'home'**),  path(**'activate/(?P<uidb64>[0-9A-Za-z\_\-]+)/(?P<token>[0-9A-Za-z]{1,13}-[0-9A-Za-z]{1,20})/'**,  activate, name=**'activate'**), ] |

Создайте форму регистрации в **template/signup.html**.

|  |
| --- |
| {% **extends 'base.html'** %}  {% **block content** %}  <**div class="container"**>  <**h2**>Sign up</**h2**>  <**form method="post"**>  {% **csrf\_token** %}  {% **for field in form** %}  <**p**>  {{ **field**.**label\_tag** }}<**br**>  {{ **field** }}  {% **if field**.**help\_text** %}  <**small style="display**: **none "**>{{ **field**.**help\_text** }}</**small**>  {% **endif** %}  {% **for error in field**.**errors** %}  <**p style="color**: **red"**>{{ **error** }}</**p**>  {% **endfor** %}  </**p**>  {% **endfor** %}  <**button type="submit"**>Sign up</**button**>  </**form**>  </**div**> {% **endblock** %} |

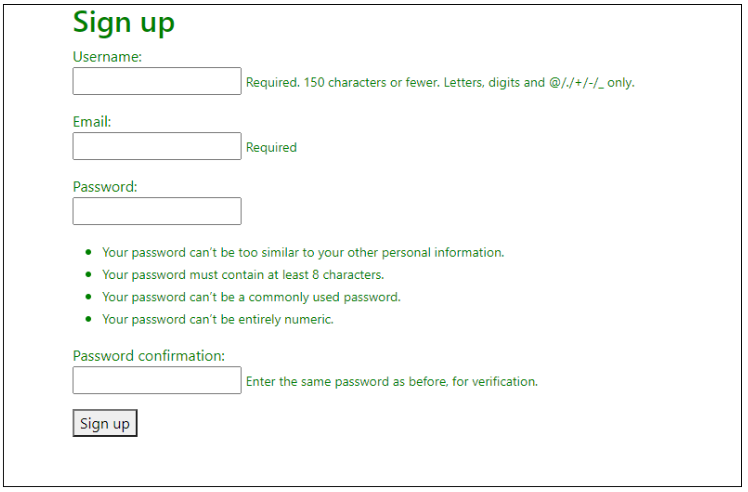
Где Base.html

|  |
| --- |
| *{#{% load staticfiles %}#}* <!DOCTYPE **html**> <**html**> <**head**>  <**title**>{% **block title** %}{% **endblock** %}</**title**>  *{# <link href="{% static "css/base.css" %}" rel="stylesheet">#}* </**head**> <**body**> <**div id="header"**>  <**span class="logo"**>GmailConfirm</**span**> </**div**> <**div id="content"**>  {% **block content** %}  {% **endblock** %} </**div**> </**body**> </**html**> |

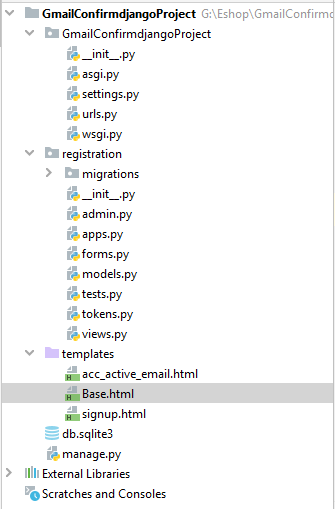
Сделаем миграцию

|  |
| --- |
| python manage.py makemigrations  python manage.py migrate |

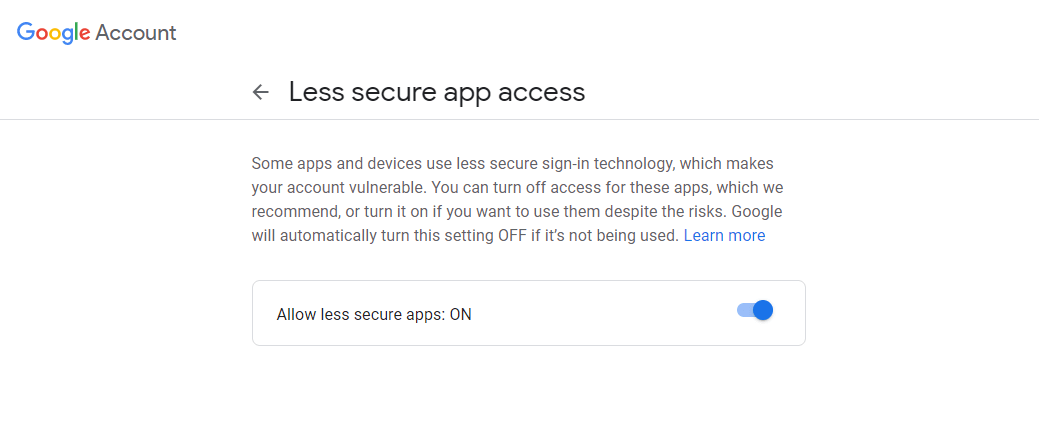
Форма будет выглядеть так, как показано ниже. Когда пользователь нажимает кнопку отправки, ссылка активации отправляет введенный им адрес электронной почты.



Общая структура проект имеет вид:



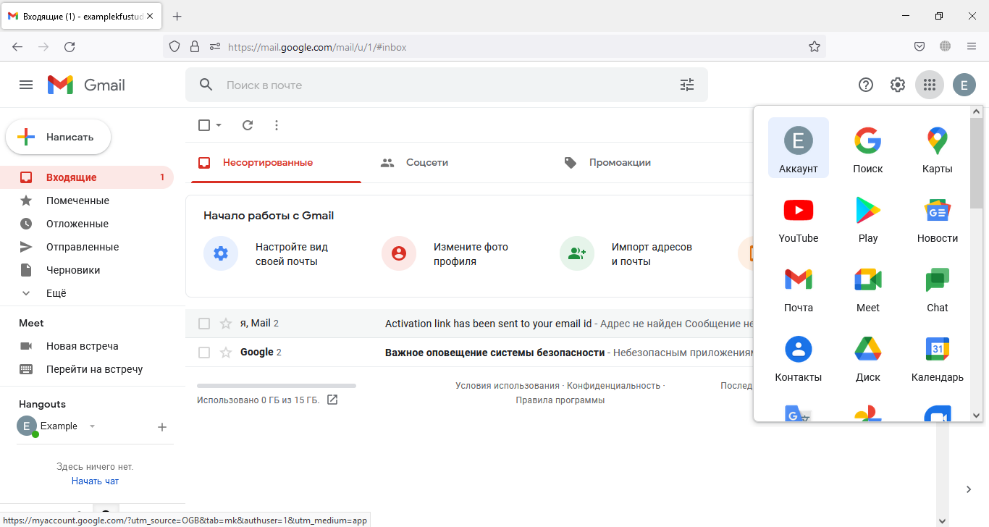
Когда вы нажимаете кнопку регистрации, электронное письмо с подтверждением отправляется на указанный адрес электронной почты.



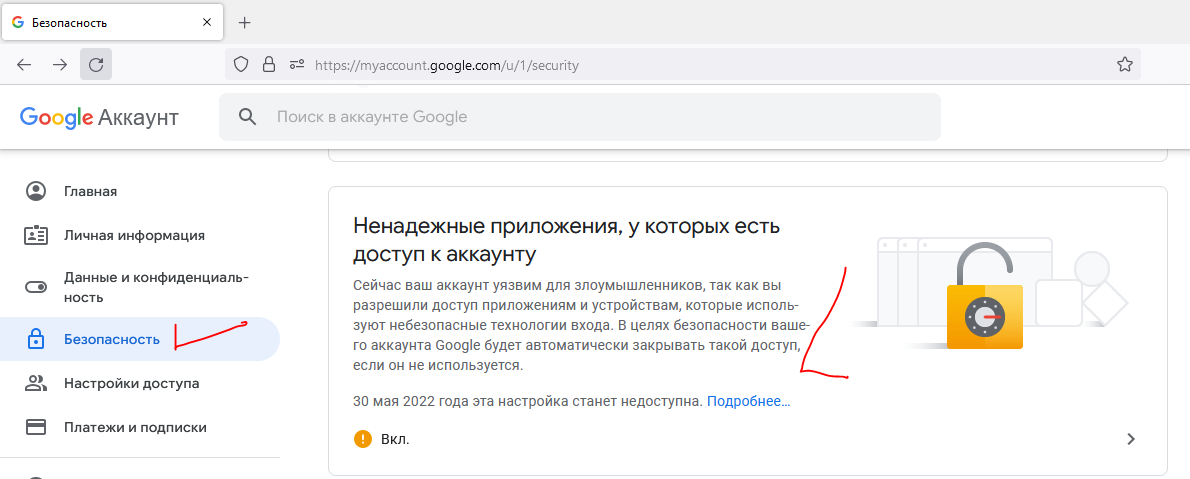
Нажмите на полученную ссылку, и теперь вы готовы войти в систему.

**Примечание. Убедитесь, что параметр менее безопасного доступа к приложению включен. В противном случае вы столкнетесь с ошибкой smtp.**

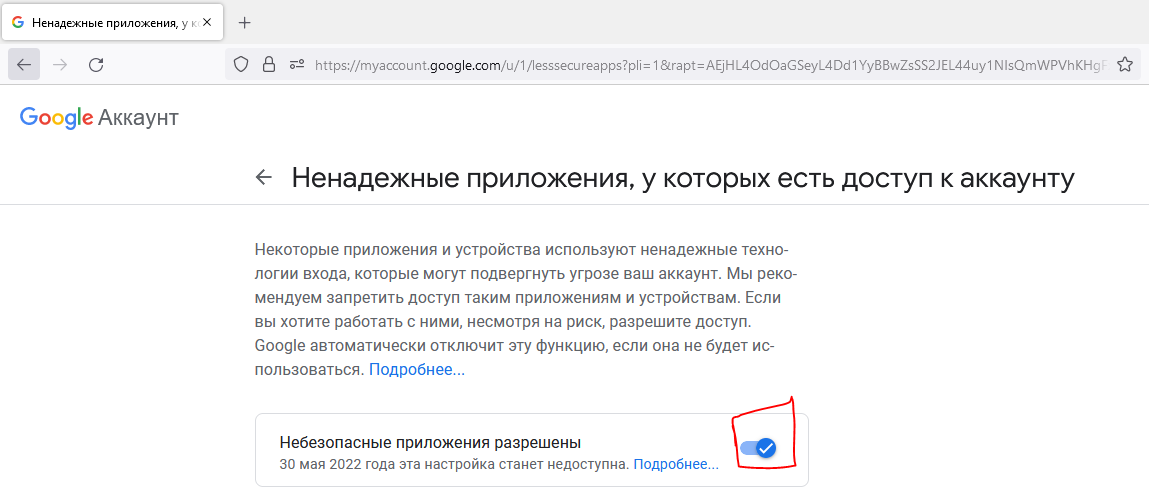
1



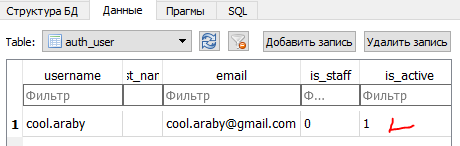
2



3



4



**Аутентификация в Django: полный пример входа, выхода и смены пароля**

В этом руководстве вы узнаете, как легко добавить полноценную систему аутентификации приложение Django с функциями входа в систему, выхода из системы, смены пароля и сброса.

Мы будем использовать django-crispy-forms и Bootstrap 4 [ <https://django.fun/tutorials/django-i-formy-bootstrap-4/>] для стилизации пользовательского интерфейса приложения.

**Что нужно знать**

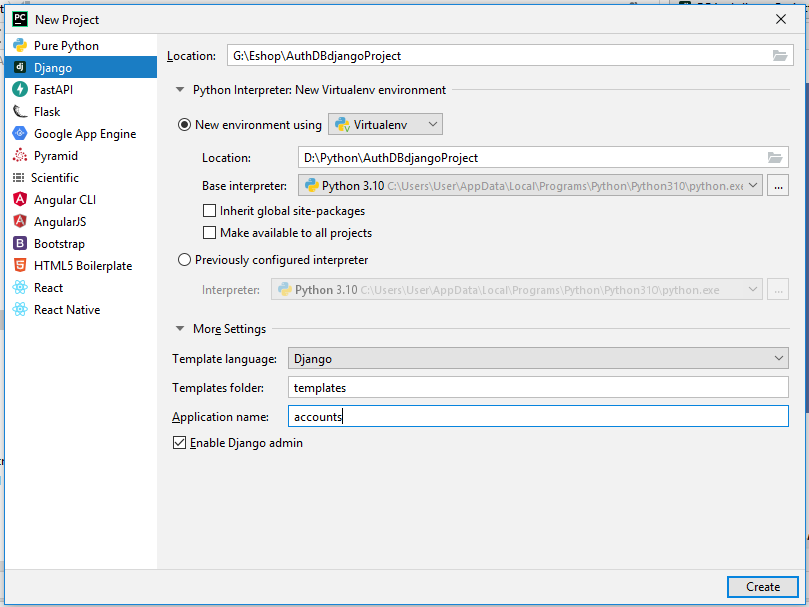
Давайте начнем с определением знаний для использования этого урока. Чтобы выполнить пошаговое руководство, вам необходимо выполнить несколько требований, таких как:

* базовые знания Python,
* знания Django (django-admin.py and manage.py),
* должна быть установлена последняя версия Python 3 (сейчас это 3.7),
* база данных SQLite установлена в вашей системе.

Мы будем использовать pip и venv, которые связаны в виде модулей в последних версиях Python, поэтому вам не нужно устанавливать их, если вы не работаете со старыми версиями.

Если вы готовы, давайте начнем!

**Создание проекта Django**



Здесь мы будем использовать Crispy-forms. Crispy-forms — это замечательное приложение, которое дает контроль над отображением форм **Django** не ломаю их обычную работу. Эта статья оринетирована на использование **Bootstrap** 4-ой версии, но также может быть применена и к старым версиям **Bootstrap**.

Основная причина использования **Crispy-forms** в своих проектах — это простое отображение форм **Django** без написания лишнего кода и с минимальными установками, что очень упрощает работу.

**Установка**

Устанавливаем с использованием pip:

|  |
| --- |
| pip install django-crispy-forms |

Добавляем приложение в INSTALLED\_APPS и выбираем, какой стиль будем использовать:

**settings.py**

|  |
| --- |
| INSTALLED\_APPS = [  ...  'crispy\_forms',  ]  CRISPY\_TEMPLATE\_PACK = **'bootstrap4'** |

В итоге получим:

|  |
| --- |
| INSTALLED\_APPS = [  **'django.contrib.admin'**,  **'django.contrib.auth'**,  **'django.contrib.contenttypes'**,  **'django.contrib.sessions'**,  **'django.contrib.messages'**,  **'django.contrib.staticfiles'**,  **'accounts.apps.AccountsConfig'**,  **'crispy\_forms'**, ] CRISPY\_TEMPLATE\_PACK = **'bootstrap4'** |

Теперь вы можете создать свою базу данных и запустить сервер разработки Django, используя следующие команды:

|  |
| --- |
| python manage.py migrate  python manage.py runserver |

Вы можете использовать свой браузер, чтобы перейти по адресу localhost:8000 и увидеть, как работает ваше веб-приложение.

**Встроенное приложение auth**

Приложение auth - это встроенная система аутентификации в Django, которая позволяет разработчикам добавлять аутентификацию в свои приложения, не изобретая велосипед, пытаясь реализовать базовую функциональность с нуля.

Приложение аутентификации Django предоставляет следующие функциональные возможности из «коробки»:

* Класс авторизации [LoginView](https://django.fun/cbv/Django/latest/django.contrib.auth.views/LoginView/),
* Класс выхода [LogoutView](https://django.fun/cbv/Django/latest/django.contrib.auth.views/LogoutView/),
* Сброс пароля [PasswordResetView](https://django.fun/cbv/Django/latest/django.contrib.auth.views/PasswordResetView/),
* Смена пароля [PasswordChangeView](https://django.fun/cbv/Django/latest/django.contrib.auth.views/PasswordChangeView/),

Вам нужно только предоставить шаблоны для реализации этих функций в вашем приложении.

Для регистрации пользователей необходимо создать свой вид и шаблон.

Вам нужно проверить, приложение django.contrib.auth указано в INSTALLED\_APPS файла settings.py, который используется по умолчанию.

Затем в accounts создайте файл urls.py в приложении учетных записей и добавьте следующий код:

|  |
| --- |
| **from** django.contrib.auth **import** views **from** django.urls **import** path  urlpatterns = [  ] |

**Авторизация пользователей с помощью LoginView**

Вы можете войти в систему в приложении Django, используя представление на основе классов [LoginView](https://django.fun/cbv/Django/latest/django.contrib.auth.views/LoginView/). В ваш accounts/urls.py добавьте следующий путь:

|  |
| --- |
| **from** django.contrib.auth **import** views **from** django.urls **import** path  urlpatterns = [  path(**'login/'**, views.LoginView.as\_view(), name=**'login'**),  ] |

Вы просто используете метод as\_view() класса LoginView, чтобы вернуть объект обратного вызова, который может быть назначен в качестве функции представления функции path().

Далее вам нужно предоставить шаблон для вашего входа в систему. Создайте папку с шаблонами в корне приложения ваших учетных записей и добавьте файл base.html со следующим кодом:

|  |
| --- |
| <!doctype **html**> <**html lang="en"**> <**head**>  <**link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"  integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous"**>  <**title**>Django Authentication Example</**title**> </**head**> <**body**> <**div class="container"**>  <**div class="row justify-content-center"**>  <**div class="col-4"**>  <**h1 class="text-center"**>Django Authentication Example</**h1**>  {% **block main** %}  {% **endblock** %}  </**div**>  </**div**> </**div**> </**body**> </**html**> |

Переходим к файлу urls.py приложений и изменим его содержание:

|  |
| --- |
| **from** django.urls **import** path, include urlpatterns = [  path(**'accounts/'**, include(**'accounts.urls'**)) ] |

Сначала мы импортируем Bootstrap 4 в наш базовый HTML-шаблон. Затем мы создаем контейнер <div> заголовок и основной блок main, где Django отображает другие части наших шаблонов.

Затем создайте папку templates/registration и шаблон login.html со следующим кодом:

|  |
| --- |
| {% **extends 'base.html'** %} {% **load crispy\_forms\_tags** %}  {% **block main** %}  <**div class="card"**>  <**div class="card-body"**>  <**h4 class="card-title"**>Log in to your account</**h4**>   <**form method="post"**>  {% **csrf\_token** %}  <**input type="hidden" name="next" value="**{{ **next** }}**"**>   {{ **form**|**crispy** }}   <**button type="submit" class="btn btn-primary btn-block"**>Log in</**button**>  </**form**>  </**div**>  </**div**> {% **endblock** %} |

Мы расширяем предыдущий базовый шаблон, загружаем crispy\_forms\_tags и затем перезаписываем основной блок main, чтобы добавить нашу форму входа.

Затем мы создаем HTML-форму с помощью метода POST и визуализируем поля формы с помощью фильтра crispy, который применяет стили Bootstrap к отдельным полям.

csrf\_token добавляет поле для защиты CSRF в форму входа в систему.

Мы также добавляем скрытое поле формы, содержащее следующий URL-адрес, который будет использоваться Django для перенаправления пользователя на следующую страницу после успешного входа в систему. По умолчанию он перенаправляет на URL-адрес accounts/profile.

**Настройка URL для редиректа после авторизации**

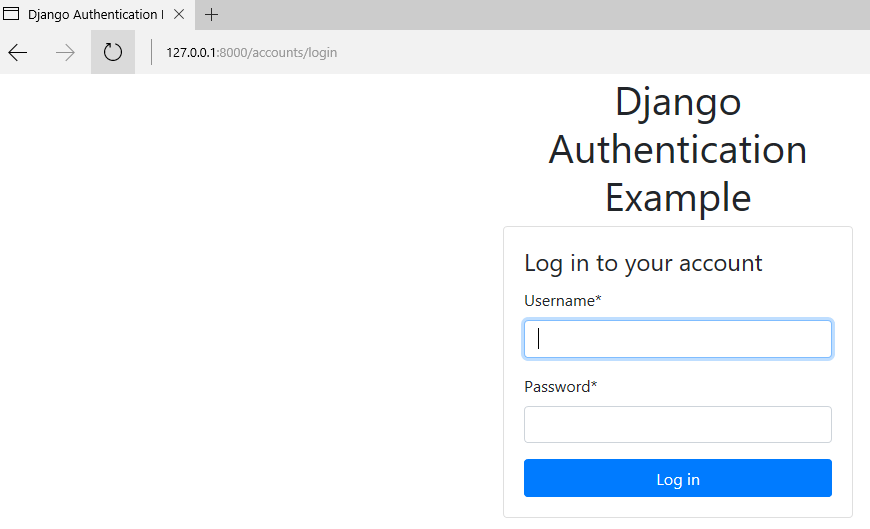
Вы можете установить URL перенаправления с помощью параметра LOGIN\_REDIRECT\_URL. Откройте файл settings.py и в конца его добавьте:

|  |
| --- |
| LOGIN\_REDIRECT\_URL = **'/'** |

Для проверки представления входа в систему вы можете создать пользователя с помощью команды manage.py createsuperuser из своего терминала.

**Примечание.** После входа в систему вы будете перенаправлены на URL-адрес /accounts/profile.

Это скриншот формы входа в систему с Bootstrap 4:



Переходим к папку template и создаём html файл с названием Index.html

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**> </**head**> <**body**> {% **block main** %}  <**ul class="sidebar-nav"**>  {% **if user**.**is\_authenticated** %}  <**li**>Пользователь: {{ **user**.**get\_username** }}</**li**>  <**li**><**a href="**{% **url 'logout'** %}**?next=**{{ **request**.**path** }}**"**>Logout</**a**></**li**>  {% **else** %}  <**li**><**a href="**{% **url 'login'** %}**?next=**{{ **request**.**path** }}**"**>Login</**a**></**li**>  {% **endif** %}  </**ul**> {% **endblock** %} </**body**> </**html**> |

Переходим к файлу urls.py приложений и изменим его содержание:

|  |
| --- |
| **from** django.urls **import** path, include **from** django.views.generic **import** TemplateView  urlpatterns = [  path(**'accounts/'**, include(**'accounts.urls'**)),  path(**''**, TemplateView.as\_view(template\_name=**"index.html"**)), ] |

**Выход пользователя с помощью LogoutView**

Вы можете выйти из системы в вашем приложении, используя представление на основе классов [LogoutView](https://django.fun/cbv/Django/latest/django.contrib.auth.views/LogoutView/). В своем файле accounts.py добавьте путь logout/ и свяжите его с вызываемым представлением LogoutView:

|  |
| --- |
| path(**'logout/'**, views.LogoutView.as\_view(), name=**'logout'**), |

В результате получим

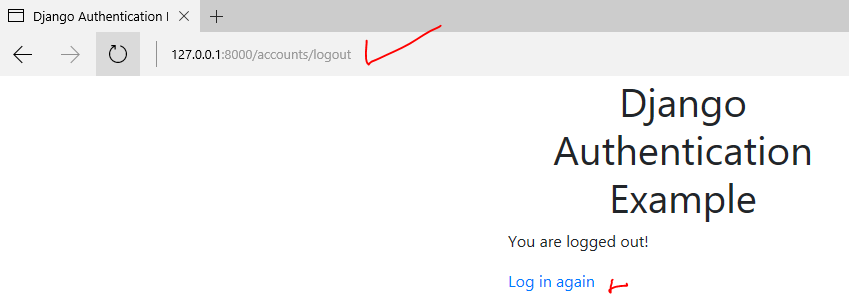
|  |
| --- |
| **from** django.contrib.auth **import** views **from** django.urls **import** path  urlpatterns = [  path(**'login/'**, views.LoginView.as\_view(), name=**'login'**),  path(**'logout/'**, views.LogoutView.as\_view(), name=**'logout'**),  ] |

Мы снова используем метод as\_view() для возврата вызываемого объекта из класса LogoutView.

Далее вам нужно создать файл registration/logged\_out.html со следующим кодом:

|  |
| --- |
| {% **extends 'base.html'** %} {% **block main** %}  <**p**>You are logged out!</**p**>  <**a href="**{% **url 'login'** %}**"**>Log in again</**a**> {% **endblock** %} |

Это снимок экрана с завершенным выходом:



PasswordResetView, PasswordResetDoneView, PasswordResetConfirmView and PasswordResetCompleteView

Вы можете разрешить своим пользователям сбрасывать свои пароли, используя множество представлений:

* [PasswordResetView](https://django.fun/cbv/Django/latest/django.contrib.auth.views/PasswordResetView/),
* [PasswordResetDoneView](https://django.fun/cbv/Django/latest/django.contrib.auth.views/PasswordResetDoneView/),
* [PasswordResetConfirmView](https://django.fun/cbv/Django/latest/django.contrib.auth.views/PasswordResetConfirmView/)
* [PasswordResetCompleteView](https://django.fun/cbv/Django/latest/django.contrib.auth.views/PasswordResetCompleteView/)

В своем файле accounts/urls.py добавьте следующие пути:

|  |
| --- |
| **from** django.contrib.auth **import** views **from** django.urls **import** path  urlpatterns = [  path(**'login/'**, views.LoginView.as\_view(), name=**'login'**),  path(**'logout/'**, views.LogoutView.as\_view(), name=**'logout'**),  path(**'password-change/'**, views.PasswordChangeView.as\_view(), name=**'password\_change'**),  path(**'password-change/done/'**, views.PasswordChangeDoneView.as\_view(), name=**'password\_change\_done'**),  ] |

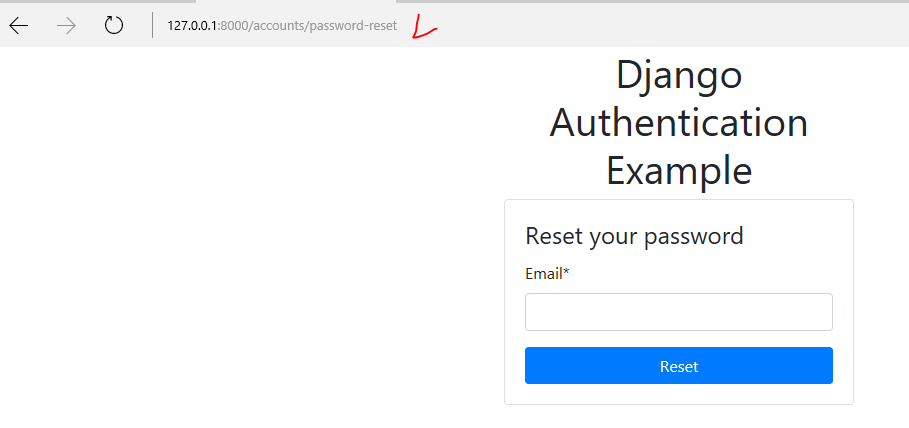
Затем вам нужно добавить шаблон registration/password\_reset\_form.html со следующим кодом:

|  |
| --- |
| {% **extends 'base.html'** %} {% **load crispy\_forms\_tags** %}  {% **block main** %}  <**div class="card"**>  <**div class="card-body"**>  <**h4 class="card-title"**>Reset your password</**h4**>  <**form method="post"**>  {% **csrf\_token** %}  <**input type="hidden" name="next" value="**{{ **next** }}**"**>  {{ **form**|**crispy** }}   <**button type="submit" class="btn btn-primary btn-block"**>Reset</**button**>  </**form**>  </**div**>  </**div**>  </**div**> {% **endblock** %} |

Таким же образом, вам нужно добавить шаблоны password\_reset\_confirm.html, password\_reset\_done.html, password\_reset\_email.html и password\_reset\_complete.html.

Это скриншот формы сброса пароля со стилями Bootstrap 4:

<http://127.0.0.1:8000/accounts/password-reset/>



**Смена пароля: PasswordChangeView и PasswordChangeDoneView**

Вы можете разрешить своим пользователям изменять свои пароли с помощью представлений на основе классов PasswordChangeView и PasswordChangeDoneView.

В своем файле accounts/urls.py добавьте следующие пути:

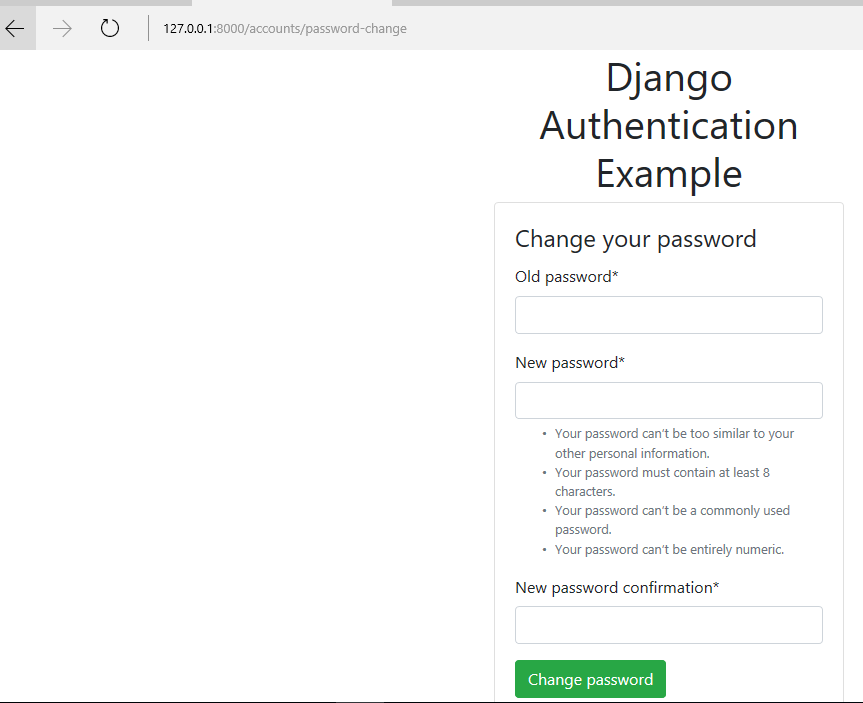
|  |
| --- |
| path(**'password-reset/'**, views.PasswordResetView.as\_view(), name=**'password\_reset'**), path(**'password-reset/done/'**, views.PasswordResetDoneView.as\_view(), name=**'password\_reset\_done'**), |

Затем создайте шаблон registration/password\_change\_form.html и добавьте следующий код:

|  |
| --- |
| {% **extends 'base.html'** %} {% **load crispy\_forms\_tags** %}  {% **block main** %}  <**div class="card"**>  <**div class="card-body"**>  <**h4 class="card-title"**> Change your password</**h4**>   <**form method="post"**>  {% **csrf\_token** %}  {{ **form**|**crispy** }}   <**button type="submit" class="btn btn-success"**>Change password</**button**>  </**form**>  </**div**>  </**div**> {% **endblock** %} |

Вам также необходимо добавить шаблон password\_change\_done.html.

Это скриншот формы смены пароля:



**Регистрация пользователей**

Для регистрации пользователей встроенное приложение аутентификации Django не предоставляет готовую функцию или представление на основе классов, поэтому вам необходимо создать собственную реализацию регистрации.

Импорт URL в основой urls.py вашего проекта

Вы добавили различные URL-адреса для реализации аутентификации в вашем приложении учетных записей, но их нельзя использовать, пока вы не добавите их в файл уровня проекта urls.py.

Во-первых, вот полный файл accounts/urls.py:

|  |
| --- |
| **from** django.contrib.auth **import** views **from** django.urls **import** path  urlpatterns = [  path(**'login/'**, views.LoginView.as\_view(), name=**'login'**),  path(**'logout/'**, views.LogoutView.as\_view(), name=**'logout'**),  path(**'password-change/'**, views.PasswordChangeView.as\_view(), name=**'password\_change'**),  path(**'password-change/done/'**, views.PasswordChangeDoneView.as\_view(), name=**'password\_change\_done'**),  path(**'password-reset/'**, views.PasswordResetView.as\_view(), name=**'password\_reset'**),  path(**'password-reset/done/'**, views.PasswordResetDoneView.as\_view(), name=**'password\_reset\_done'**),  path(**'reset/<uidb64>/<token>/'**, views.PasswordResetConfirmView.as\_view(), name=**'password\_reset\_confirm'**),  path(**'reset/done/'**, views.PasswordResetCompleteView.as\_view(), name=**'password\_reset\_complete'**), ] |

Затем откройте файл urls.py, который находится в базовом каталоге вашего проекта, и используйте функцию include() для импорта URL-адресов учетных записей:

|  |
| --- |
| **from** django.urls **import** path, include **from** django.views.generic **import** TemplateView  urlpatterns = [  path(**'accounts/'**, include(**'accounts.urls'**)),  path(**''**, TemplateView.as\_view(template\_name=**"index.html"**)), ] |

Теперь запустите сервер разработки. Вы можете использовать аутентификационные URL-адреса по адресу /accounts:

<http://127.0.0.1:8000/accounts/login/> для входа,

<http://127.0.0.1:8000/accounts/logout/> для выхода,

<http://127.0.0.1:8000/accounts/password-change/> для смены пароля,

<http://127.0.0.1:8000/accounts/password-reset/> для сброс пароля.