

## Практическое занятие

**Тема:** Работа с изображениями и файлами в формах Django.

**Цель:** научиться работать с изображениями и файлами в формах Django.

**Оборудование:** персональный компьютер, IDE PyCharm, Visual Studio Code, методические рекомендации к проведению лекционных занятий, Литература: Дронов В. А. Django. Практика создания веб-сайтов на Python - Санкт-Петербург : БХВ-Петербург, 2023. - 800 с., Постолиит А. Python, Django и PyCharm для начинающих – - Санкт-Петербург : БХВ-Петербург, 2021. – 464 с., METANIT.COM: сайт о программировании. - URL: <https://metanit.com/sharp/> – режим доступа: свободный, инструкционная карта для проведения лекционного занятия.

**Загрузка и отображение файлов PDF в формах Django.** Рассмотрим процедуру загрузки на сайт файлов на примере документов в формате PDF. Для этого создадим новую модель для описания структуры БД, форму для загрузки файлов и шаблон HTML-страницы для загрузки документов в формате PDF. Откроем модуль `hello/firstapp/models.py` и создадим новую модель с описанием структуры таблицы БД для хранения файлов.

# Загрузка файлов

```
class File(models.Model):
    title = models.CharField(max_length=100,
                             verbose_name="Описание файла",)
    file = models.FileField(upload_to='files',
                             verbose_name="Файл PDF",
                             null=True, blank=True)

    def __str__(self):
        return self.title
```

Здесь была создана модель `File`, в которой описаны два поля:

- ❑ `title` — заголовок, поясняющий содержание файла;
- ❑ `file` — поле для хранения имени файла.

Для обоих полей заданы метки (`verbose_name`), а также определено, что поле `title` обязательно для заполнения (`null=False`), а поле `file` может оставаться пустыми (`null=True, blank=True`). Параметр `upload_to` указывает расположение файлов, т. е. каталог, в котором будут храниться файлы. Нам не нужно создавать каталог мультимедиа для хранения файлов вручную, он появится автоматически, когда будут загружены файлы. В данном случае файлы будут храниться в папках `media/files/`.

К этому классу добавлен метод `ctef_str_`, который позволит отобразить поле с описанием файлов в административной панели Django.

Все необходимые настройки были выполнены в разделе, описывающем загрузку изображений, и на данном этапе нужно просто запустить следующие команды для миграции созданной модели в БД:

```
python manage.py makemigrations
python manage.py migrate
```

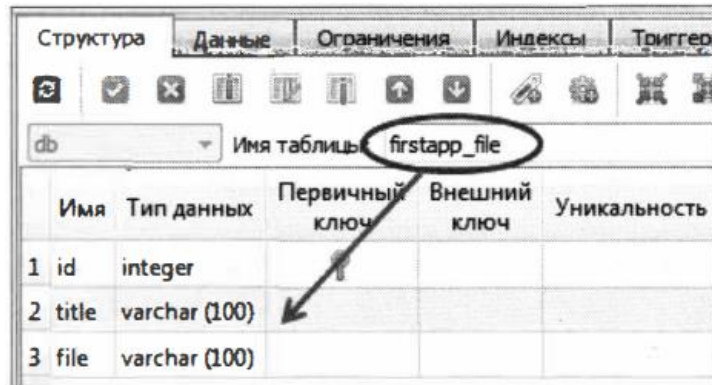
Если теперь открыть Бд, то можно увидеть, что там присутствует таблица `firstapp _ file`.

Как видно из данного рисунка, в БД создана таблица `firstapp _ file` с тремя полями:

- ❑ `id` — идентификатор записи (тип поля `integer` — целое число);
- ❑ `title` — заголовок, поясняющий содержание файла (тип поля `varchar` — текстовое поле переменной длины с максимальным числом символов, равным 100);

- ❑ `file` — поле для хранения имени файла (тип поля `varchar` — текстовое поле переменной длины с максимальным числом символов, равным 100).

Поскольку `file` - это текстовое поле, то в нем будет храниться только имя файла, но не сам файл.



	Имя	Тип данных	Первичный ключ	Внешний ключ	Уникальность
1	id	integer	✓		
2	title	varchar (100)			
3	file	varchar (100)			

**Таблица БД `firstapp_file` для хранения сведений о загруженных файлах**

Теперь создадим форму, которая будет принимать файл в качестве входных данных от пользователя. Откроем файл `hello/firstapp/forms.py` и напишем в нем код.

```
from .models import File # для работы с файлами
```

```
class FileForm(forms.ModelForm):  
    class Meta:  
        model = File  
        fields = '__all__'
```

Здесь из моделей был импортирован класс `File`, затем в классе `meta` на его основе созданы объекты `model` и `fields` (поля). В объект `fields` будут загружены все поля из модели `File`.

Теперь создадим представление (`view`) для обработки запросов, связанных с загрузкой файлов. Откроем модуль `hello/firstapp/views.py` и напишем в нем код.

```
# импорт модели и формы для работы с файлами  
from .models import File  
from .forms import FileForm
```

```
# загрузка файлов pdf  
def form_up_pdf(request):  
    if request.method == 'POST':  
        form = FileForm(request.POST, request.FILES)  
        if form.is_valid():  
            form.save()  
  
    my_text = 'Загруженные файлы'  
    form = FileForm()  
    file_obj = File.objects.all()  
    context = {'my_text': my_text, "file_obj": file_obj, "form": form}  
    return render(request, 'firstapp/form_up_pdf.html', context)
```

```
# удаление файлов из БД  
def delete_pdf(request, id):  
    try:  
        pdf = File.objects.get(id=id)  
        pdf.delete()  
        return redirect('form_up_pdf')
```

```
except Person.DoesNotExist:
    return HttpResponseNotFound("<h2>Объект не найден</h2>")
```

В этом модуле на первом шаге выполнен импорт модели и формы для работы с файлами:

```
from .models import File
from .forms import FileForm
```

Затем создана функция для загрузки файлов - `ctef form _ up _pdf (request)`. В этой функции проверяется условие, поступил ли запрос от пользователя на загрузку файла (`if request.method == 'POST'`). Если такой запрос поступил, то на основе класса `FileForm` создаётся объект `form`, который получает запрос от пользователя на сохранение данных о файле (`request.POST`), и сам загружаемый файл (`request.FILES`). Если форма не содержит ошибок, то введённые пользователем данные сохраняются (`form.save`). После этого происходит обновление формы для загрузки файлов.

Если форма вызывается первый раз, т. е. поступил запрос `GET`, то создаются:

- ☐ текстовая переменная `my_text`;
- ☐ объект `form`, который создается на основе класса `FileForm` (т. е. сама форма);
- ☐ объект `file_obj`, который принимает из БД все сведения о загруженных файлах.

Затем создаётся объект `context`, в который в виде словаря передаются объекты `my_text`, `file_obj` и `form`. После этого вызывается шаблон `form_up_pdf.html`, в который передаются все данные через объект `context`.

Чтобы отобразить форму для загрузки файлов, создадим в папке с шаблонами Django HTML-страницу с именем `hello/templates/firstapp/form_up_pdf.html`.

```
{% extends "firstapp/base.html" %}
{% block title %}Формы{% endblock title %}
{% block header %}Изучаем формы Django{% endblock header %}
{% block content %}
<div class="container-fluid text-start my-2
    border border-5 border-warning">
    <h5>Формы - загрузка файлов</h5>
    <form method="POST" enctype="multipart/form-data">
        {% csrf_token %}
        <div class="form-group my-2">
            {{ form.as_p }}
            <button type="submit">Загрузить</button>
        </div>
    </form>
    {% if file_obj.count > 0 %}
    <h3> {{ my_text }}</h3>
    <table class="table table-striped table-bordered
        text-start">
        <thead>
            <tr>
                <th>№</th>
                <th>Описание файла</th>
                <th>Имя файла</th>
                <th>Удаление</th>
                <th>Показать</th>
            </tr>
        </thead>
        <tbody>
```

```

{% for obj in file_obj %}
    <tr>
        <td>{{ obj.id }}</td>
        <td>{{ obj.title }}</td>
        <td>{{ obj.file }}</td>
        <td><a href="delete_pdf/{{ obj.id }}">Удалить</a></td>
        <td><a href="{{ obj.file.url }}" class="btn btn-primary"
            target="_blank">Показать</a></td>
    </tr>
{% endfor %}
</tbody>
</table>
{% endif %}
</div>
{% endblock content %}

```

Здесь шаблон имеет два важных тега:

- ❑ `{{ form.as_p }}` — для отображения полей, обеспечивающих загрузку файлов;
- ❑ `{% if file_obj.count > 0 %}` — для показа файлов, которые пользователь загрузил в БД.

Здесь тег `csrf_token` защитит форму от вредоносных данных, а `form.as_p` отобразит поля, предназначенные для ввода данных, в виде параграфов.

#### **ПРИМЕЧАНИЕ**

Здесь используется опция `enctype = "multipart/form-data"`, которая позволяет отправлять файлы через POST-запросы. Без активации этой опции пользователь не может отправить файл через запрос POST. Эту опцию необходимо обязательно включать в формы работы с файлами и изображениями.

В теге `{% if file_obj.count > 0 %}` делается проверка, есть ли загруженные файлы. Если это условие выполняется, то в теге будет создана таблица, в которой в цикле (тег `{% obj in file_obj %}`) будут показаны все файлы, которые были загружены на сайт и находятся по соответствующим ссылкам. В колонке таблицы Удаление сделана ссылка на функцию удаления файла из БД. В колонке таблицы Показать находится кнопка со ссылкой на URL-адрес документа, по которому он будет показан в полном размере в отдельной вкладке браузера. Иначе говоря, в данной форме можно загружать файлы PDF, просматривать их и удалять с сайта.

Теперь, когда и форма, и обработчик формы готовы, сопоставим форму с её URL-адресом в `urls.py`. Открываем модуль `hello/firstapp/urls.py` и добавляем в него две строки.

```

from django.urls import path
from . import views

urlpatterns = [
    path("", views.index, name='index'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
    path('my_form/', views.my_form, name='my_form'),
    path('my_form/edit_form/<int:id>/', views.edit_form, name='edit_form'),
    path('my_form/delete/<int:id>/', views.delete),
    path('form_up_img/', views.form_up_img, name='form_up_img'),
    path('form_up_img/delete_img/<int:id>/', views.delete_img),
    path('form_up_pdf/', views.form_up_pdf, name='form_up_pdf'),
    path('form_up_pdf/delete_pdf/<int:id>/', views.delete_pdf),
]

```

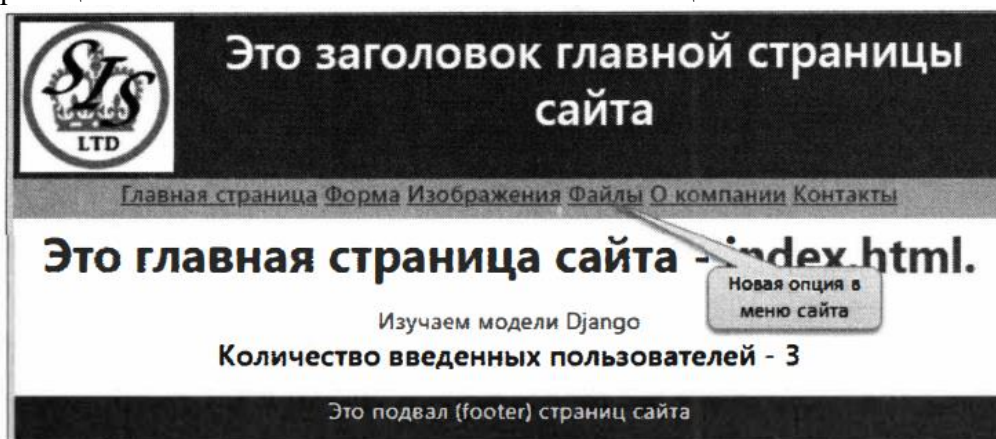
В этом программном коде была сопоставлена функция загрузки файлов ( views. form \_ up ydf) с URL-адресом шаблона HTML-страницы, которая отображает саму форму ('form\_up\_pdf/'), и функция удаления файла (views.delete\_pdf) с её URL-адресом (form\_upydf/delete\_pdf/). В этой инструкции в функцию delete\_pdf передаётся идентификатор той записи, которую нужно удалить из БД.

Остался последний шаг - необходимо включить вызов формы form\_up\_pdf.html из меню главной страницы сайта. Для этого в базовый шаблон сайта hello/templates/firstapp/base.html добавим одну строку - ссылку на страницу form\_up\_pdf.html. Ниже приведён фрагмент кода модуля base.html.

```
<div class="row bg-warning text-center">
  <h6>
    <a href="{% url 'index' %}">Главная страница</a>
    <a href="{% url 'my_form' %}">Форма</a>
    <a href="{% url 'form_up_img' %}">Изображения</a>
    <a href="{% url 'form_up_pdf' %}">Файлы</a>
    <a href="{% url 'about' %}">О компании</a>
    <a href="{% url 'contact' %}">Контакты</a>
  </h6>
</div>
```

Итак, у нас все готово для того, чтобы продемонстрировать загрузку файлов PDF в БД нашего учебного сайта. Запускаем наш веб-сервер командой python manage.py runserver

На главной странице сайта в главном меню появилась новая опция Файлы.



Опция в меню сайта вызова формы для загрузки файлов

Если теперь щёлкнуть мышью на ссылке Файлы, то откроется форма для загрузки файлов.

Форма для загрузки файлов

Как видно из данного рисунка, на форме имеется поле для ввода текста с описанием файла и две кнопки: Выберите файл - для выбора файла и Загрузить - для загрузки файла в БД сайта.



Если теперь выбрать какой-нибудь файл (документ в формате PDF), ввести его описание и нажать на кнопку Загрузить, то данный файл будет загружен на сайт, а имя загруженного сайта будет сохранено в БД. После этих действий страница будет выглядеть так.

Это заголовок главной страницы сайта

Главная страница Форма Иллюстрации Файлы О компании Контакты

### Изучаем формы Django

Формы - загрузка файлов

Описание файла:

Файл PDF:  Файл не выбран

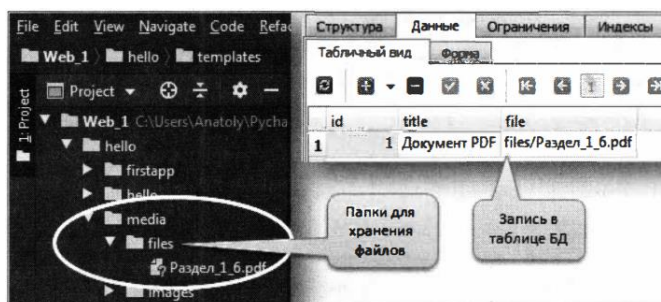
Загруженные файлы

№	Описание файла	Имя файла	Удаление	Показать
1	Документ PDF	files/Раздел_1_6.pdf	<input type="button" value="Удалить"/>	<input type="button" value="Показать"/>

Это подвал (footer) страниц сайта

Форма form\_up\_pdf.html после загрузки файла

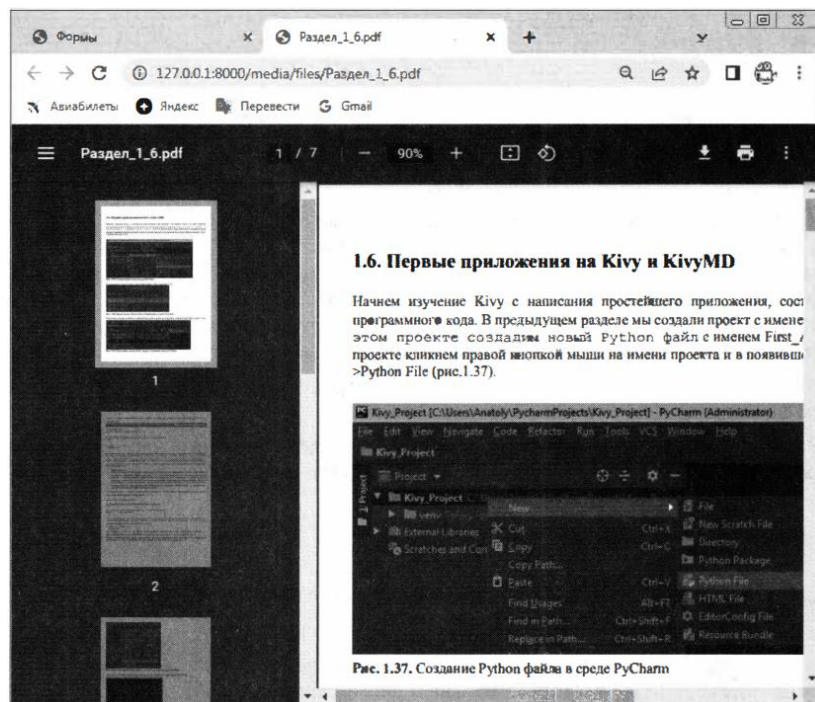
После этих действий в структуре проекта появится папка для хранения файлов media/files/, где и будут сохраняться загружаемые файлы, а имена загруженных файлов будут записаны в БД в таблицу firstapp/file.



Папки для хранения файлов PDF записи в таблице БД с именами загруженных файлов

Если нажать на кнопку Показать, то в браузере откроется новая вкладка, в которой будет показано содержание документа в формате PDF.

Как видно, на данной вкладке можно выполнять постраничное пролистывание документа, масштабирование, поворот страниц и вывод на печать. Таким образом, мы реализовали загрузку на сайт файлов формата PDF, просмотр документов и их удаление с сайта. Теперь рассмотрим выполнение аналогичных действий с файлами других типов.



Вывод содержимого PDF-файла в отдельной вкладке браузера

**Загрузка и отображение видеофайлов в формах Django.** Рассмотрим процедуру загрузки на сайт видеофайлов на примере коротких роликов в формате MP4. Для этого создадим новую модель для описания структуры БД, форму для загрузки файлов и шаблон HTML-страницы для загрузки видеофайлов. Откроем модуль `hello/firstapp/models.py` и создадим новую модель с описанием структуры таблицы БД для хранения видеофайлов.

# Загрузка видео файлов

```
class VideoFile(models.Model):
    title = models.CharField(max_length=100,
                             verbose_name="Описание файла",)
    file = models.FileField(upload_to='videos',
                             verbose_name="Видео файл",
                             null=True, blank=True)
    obj_video = models.Manager()

    def __str__(self):
        return self.title
```

Здесь была создана модель `VideoFile`, в которой описаны два поля:

- ❑ **title** — заголовок, поясняющий содержание файла;
- ❑ **file** — поле для хранения имени файла.

Для обоих полей заданы метки (`verbose_name`), а также определено, что поле `title` обязательно для заполнения (`null=False`), а поле `file` может оставаться пустым (`null=True, blank=True`). Параметр `upload_to` указывает расположение файлов, т. е. каталог, в котором будут храниться файлы. Нам не нужно создавать каталог мультимедиа для хранения файлов вручную, он появится автоматически, когда будет выполнена загрузка файлов. В данном случае файлы будут храниться в папках `media/videos/`.

К этому классу добавлен метод `ctef_str__`, который позволит отобразить поле с описанием файлов в административной панели Django.

Все необходимые настройки были выполнены в разделе, описывающем загрузку изображений, и на данном этапе нужно просто запустить следующие команды для миграции созданной модели в БД:

```
python manage.py makemigrations
python manage.py migrate
```

После этого в БД будет создана таблица `firstapp_videofile` с тремя полями:

- ☐ `id` — идентификатор записи (тип поля `integer` — целое число);
- ☐ `title` — заголовок, поясняющий содержание файла (тип поля `varchar` — текстовое поле переменной длины с максимальным числом символов 100);
- ☐ `file` — поле для хранения имени файла (тип поля `varchar` — текстовое поле переменной длины с максимальным числом символов 100).

Поскольку `file` - это текстовое поле, то в нем будет храниться только имя файла, но не сам файл. Теперь создадим форму, которая будет принимать файл в качестве входных данных от пользователя. Откроем файл `hello/firstapp/forms.py` и напишем в нем код.

```
from .models import VideoFile # для работы с файлами видео
```

```
class VideoForm(forms.ModelForm):
    class Meta:
        model = VideoFile
        fields = '__all__'
```

Здесь из моделей был импортирован класс `VideoFile`, затем в классе `meta` на его основе созданы объекты `model` и `fields` (поля). В объект `fields` будут загружены все поля из модели `VideoFile`.

Теперь создадим представление (`view`) для обработки запросов, связанных с загрузкой файлов. Откроем модуль `hello/firstapp/views.py` и напишем в нем код.

```
# импорт модели и формы для работы с видео файлами
from .models import VideoFile
from .forms import VideoForm
```

```
# загрузка видео файлов
def form_up_video(request):
    if request.method == 'POST':
        form = VideoForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()

    my_text = 'Загруженные видео файлы'
    form = VideoForm()
    file_obj = VideoFile.obj_video.all()
    context = {'my_text': my_text, "file_obj": file_obj, "form": form}
    return render(request, 'firstapp/form_up_video.html', context)
```

```
# удаление видео файлов из БД
def delete_video(request, id):
    try:
        video = VideoFile.obj_video.get(id=id)
        video.delete()
        return redirect('form_up_video')
    except Person.DoesNotExist:
        return HttpResponseNotFound("<h2>Объект не найден</h2>")
```

В этом модуле на первом шаге выполнен импорт модели и формы для работы с видеофайлами:



```
from .models import VideoFile
from .forms import VideoForm
```

Затем создана функция для загрузки файлов - `def form_up_video(request)`. В этой функции проверяется условие, поступил ли запрос от пользователя на загрузку видеофайла (`if request.method == 'POST'`). Если такой запрос поступил, то на основе класса `VideoForm` создаётся объект `form`, который получает запрос от пользователя на сохранение данных о видеофайле (`request.POST`), и сам загружаемый файл (`request.FILES`). Если форма не содержит ошибок, то выполняется сохранение введённых пользователем данных (`form.save`). После этого происходит обновление формы для загрузки видеофайлов.

Если форма вызывается первый раз, т. е. поступил запрос `GET`, то создаются:

- ❑ текстовая переменная `my_text`;
- ❑ объект `form`, который создается на основе класса `VideoForm` (т. е. сама форма);
- ❑ объект `file_obj`, который принимает из БД все сведения о загруженных файлах.

Затем создаётся объект `context`, в который в виде словаря передаются объекты: `my_text`, `file_obj`, `form`. После этого вызывается шаблон `form_up_video.html`, в который передаются все данные через объект `context`.

Чтобы отобразить форму для загрузки видеофайлов, создадим в папке с шаблонами Django HTML-страницу с именем `hello/templates/firstapp/form_up_video.html`.

```
{% extends "firstapp/base.html" %}
{% block title %}Формы{% endblock title %}
{% block header %}Изучаем формы Django{% endblock header %}
{% block content %}
<div class="container-fluid text-start my-2
    border border-5 border-warning">
    <h5>Формы - загрузка видео файлов</h5>
    <form method="POST" enctype="multipart/form-data">
        {% csrf_token %}
        <div class="form-group my-2">
            {{ form.as_p }}
            <button type="submit">Загрузить</button>
        </div>
    </form>
    {% if file_obj.count > 0 %}
    <h3> {{ my_text }}</h3>
    <table class="table table-striped table-bordered
        text-start">
        <thead>
            <tr>
                <th>№</th>
                <th>Описание файла</th>
                <th>Имя файла</th>
                <th>Удаление</th>
                <th>Показать</th>
            </tr>
        </thead>
        <tbody>
            {% for obj in file_obj %}
            <tr>
```

```

<td>{{ obj.id }}</td>
<td>{{ obj.title }}</td>
<td>{{ obj.file }}</td>
<td><a href="delete_video/{{ obj.id }}">Удалить</a></td>
<td>
  <video width="320" height="240" controls>
    <source src="{{ obj.file.url }}"
      type="video/mp4">
  </video>
</td>
</tr>
{% endfor %}
</tbody>
</table>
{% endif %}
</div>
{% endblock content %}

```

Здесь в шаблоне есть два важных тега:

- ☐ `{{ form.as_p }}` — для отображения полей, обеспечивающих загрузку файлов;
- ☐ `{% if file_obj.count > 0 %}` — для показа файлов, которые пользователь загрузил в БД.

Здесь тег `csrf_token` защитит форму от вредоносных данных, а `form.as_p` отобразит поля, предназначенные для ввода данных, в виде параграфов.

#### **ПРИМЕЧАНИЕ**

Здесь используется опция `enctype = "multipart/form-data"`, которая позволяет отправлять файлы через POST-запросы. Без активации этой опции пользователь не может отправить файл через запрос POST. Эту опцию необходимо обязательно включать в формы работы с файлами и изображениями.

В теге `{ % if file_obj.count > 0 % }` делается проверка, есть ли загруженные файлы. Если это условие выполняется, то в теге `<table>` будет создана таблица, в которой в цикле (тег `{% obj in file_obj % }`) будут показаны все файлы, загруженные на сайт и находящиеся по соответствующим ссылкам. В колонке таблицы Удаление сделана ссылка на функцию удаления файла из БД. В колонке таблицы Показать находится кнопка со ссылкой на URL-адрес видеофайла, по которому он будет показан в данной колонке. В результате в данной форме можно загружать видеофайлы, просматривать их и удалять с сайта.

Здесь следует обратить особое внимание на тег `<video>`. До появления HTML 5 веб-разработчикам приходилось вставлять видео на веб-страницу с помощью подключаемого модуля, такого как Adobe Flash Player. С помощью Bootstrap 5 можно легко вставлять видео в HTML-документ с помощью тега `<video>`.

Тег `<video>` поддерживает глобальные атрибуты, которые используются всеми тегами HTML, такие как `id`, `class`, `style` и т. д. Кроме того, он имеет набор собственных атрибутов: `autoplay`, `controls`, `loop`, `muted`, `src`, `poster`, `preload` и др. Рассмотрим наиболее важные атрибуты.

Атрибут `src` используется для указания источника видео. Это может быть относительный путь к видео на локальном компьютере или ссылка на видео из Интернета:

```
<video src="weekend.mp4"></video>
```

Вместо него можно использовать тег .

Атрибут poster позволяет добавить изображение, которое будет показано до начала воспроизведения видео или во время его загрузки.

```
<video src="weekend.mp4" poster="benefits-of-coding.jpg"></video>
```

Вместо изображения первой сцены из видео браузер покажет это изображение. Атрибут controls. Когда используется элемент control, то в браузере будут отображаться контроллеры управления воспроизведением, такие как "пуск", "пауза", "громкость", "поиск" и т. д. Например:

```
<video
  controls
  src="weekend.mp4"
  poster="benefits-of-coding.jpg"
></video>
```

Атрибут loop даёт возможность автоматически повторять видео - заставляя его воспроизводиться снова каждый раз, когда показ завершится. Например:

```
<video
  controls
  loop
  src="weekend.mp4"
  poster="benefits-of-coding.jpg"
></video>
```

Атрибут autoplay позволяет настроить автоматическое воспроизведение видео сразу после загрузки страницы. Например:

```
<video
  controls
  loop
  autoplay
  src="weekend.mp4"
  poster="benefits-of-coding.jpg"
></video>
```

Атрибуты width и height дают возможность указать ширину и высоту видео в пикселах. Он принимает только абсолютные значения, например:

```
<video
  controls
  loop
  autoplay
  src="weekend.mp4"
  width="350px"
  height="250px"
  poster="benefits-of-coding.jpg"
></video>
```

Атрибут muted можно использовать для того, чтобы браузер не воспроизводил звуковую дорожку, связанную с видео. Например:

```

<video
    controls
    loop
    autoplay
    muted
    src="weekend.mp4"
    width="350px"
    height="250px"
    poster="benefits-of-coding.jpg"
></video>

```

Атрибут `preload` "подскажет" браузеру, следует ли загружать видео при загрузке страницы. Этот атрибут имеет решающее значение для взаимодействия с пользователем. Доступны следующие значения этого атрибута:

- ☐ `none` — указывает, что видео не будет загружаться, пока пользователь не нажмет кнопку воспроизведения;
- ☐ `auto` — указывает, что видео должно загружаться, даже если пользователь не нажимает кнопку воспроизведения;
- ☐ `metadata` — указывает, что браузер должен собирать метаданные, такие как длина, размер, продолжительность и т. д.

Вот пример использования этого атрибута:

```

<video
    controls
    loop
    autoplay
    muted="true"
    preload="metadata"
    src="weekend.mp4"
    width="350px"
    height="250px"
    poster="benefits-of-coding.jpg"
></video>

```

Теперь, когда и форма, и обработчик формы готовы, сопоставим форму с её URL-адресом в `urls.py`. Открываем модуль `hello/firstapp/urls.py` и добавляем в него две строки.

```

from django.urls import path
from . import views

```

```

urlpatterns = [
    path("", views.index, name='index'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
    path('my_form/', views.my_form, name='my_form'),
    path('my_form/edit_form/<int:id>', views.edit_form, name='edit_form'),
    path('my_form/delete/<int:id>', views.delete),
    path('form_up_img/', views.form_up_img, name='form_up_img'),
    path('form_up_img/delete_img/<int:id>', views.delete_img),
    path('form_up_pdf/', views.form_up_pdf, name='form_up_pdf'),
    path('form_up_pdf/delete_pdf/<int:id>', views.delete_pdf),

```

```

path('form_up_video/', views.form_up_video, name='form_up_video'),
path('form_up_video/delete_video/<int:id>/', views.delete_video),
]

```

В этом программном коде была сопоставлена функция загрузки видеофайлов (`views.form_up_video`) с URL-адресом шаблона HTML-страницы, которая отображает саму форму (`'form_up_video/'`), и функция удаления файла (`views.delete_video`) с её URL-адресом (`form_up_pdf/delete_video/`). В этой инструкции в функцию `delete_video` передаётся идентификатор той записи, которую нужно удалить из БД.

Остался последний шаг - необходимо включить вызов формы `form_up_video.html` из меню главной страницы сайта. Для этого в базовый шаблон сайта `hello/templates/firstapp/base.html` добавим одну строку - ссылку на страницу `form_up_video.html`.

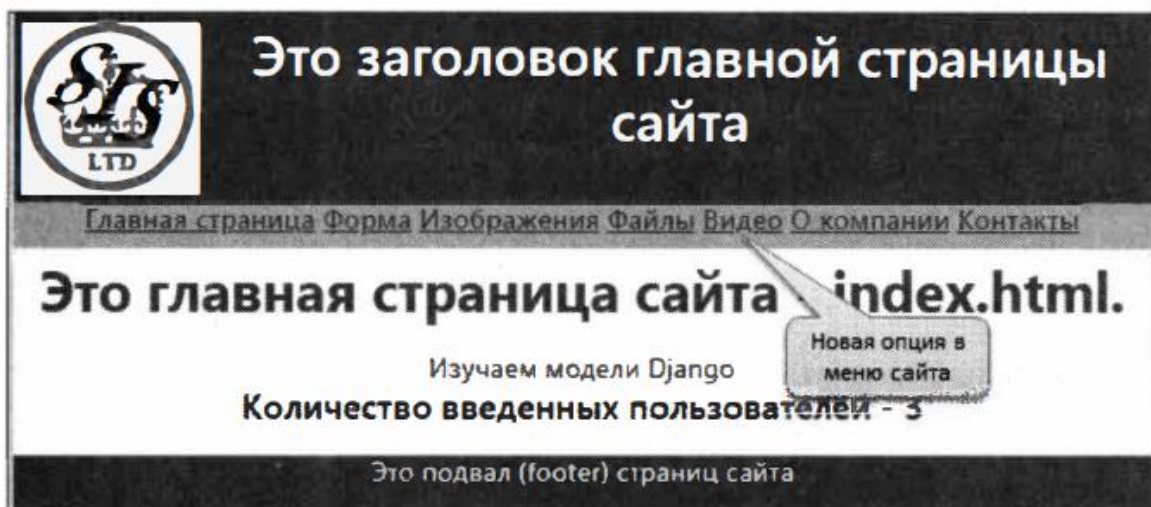
```

<a href="{% url 'index' %}">Главная страница</a>
<a href="{% url 'my_form' %}">Форма</a>
<a href="{% url 'form_up_img' %}">Изображения</a>
<a href="{% url 'form_up_pdf' %}">Файлы</a>
<a href="{% url 'form_up_video' %}">Видео</a>
<a href="{% url 'about' %}">О компании</a>
<a href="{% url 'contact' %}">Контакты</a>

```

Итак, у нас все готово для того, чтобы продемонстрировать загрузку видеофайлов в БД нашего учебного сайта. Запускаем наш веб-сервер, выполнив команду `python manage.py runserver`

На главной странице сайта в главном меню появилась новая опция Видео.



**Опция в меню сайта вызова формы для загрузки видеофайлов**

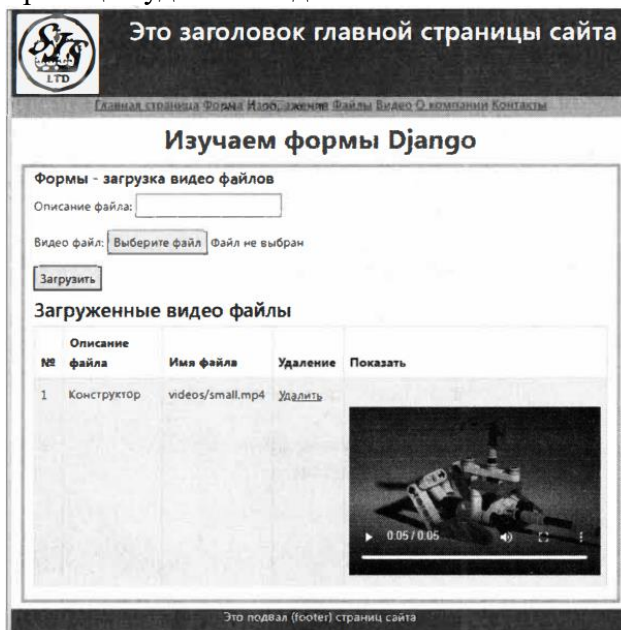
Если теперь щёлкнуть мышью на ссылке Видео, то откроется форма для загрузки видеофайлов.

**Форма для загрузки видеофайлов**



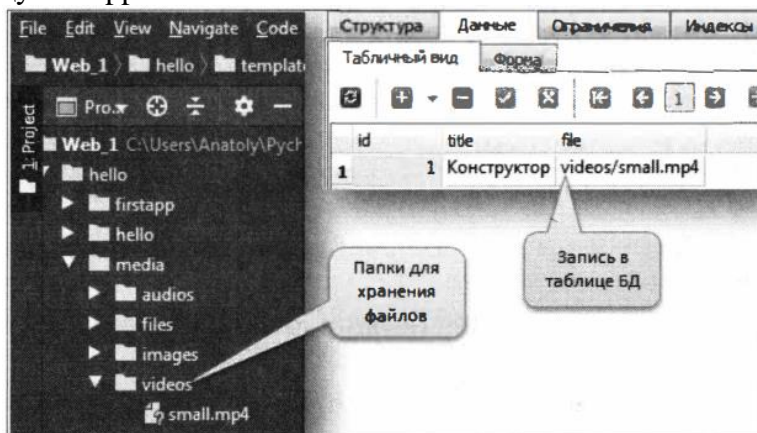
Как видно, на форме имеется поле для ввода текста с описанием файла и две кнопки: Выберите файл - для выбора файла, Загрузить - для загрузки файла в БД сайта.

Если теперь выбрать какой-нибудь видеофайл (в формате MP4), ввести его описание и нажать на кнопку Загрузить, то данный файл будет загружен на сайт, а имя загруженного файла будет сохранено в БД. После этих действий страница будет выглядеть так.



Форма form\_up\_video.html после загрузки файла

После этих действий в структуре проекта будет создана папка для хранения видеофайлов media/videos/, где и будут сохраняться загружаемые файлы, а имена загруженных файлов будут записаны в БД в таблицу firstapp/videofile.



Папки для хранения видеофайлов записи в таблице БД с именами загруженных файлов

Таким образом, мы реализовали загрузку на сайт видеофайлов, просмотр видео и их удаление с сайта. Теперь рассмотрим выполнение аналогичных действий с аудиофайлами.

**Загрузка и озвучивание аудиофайлов в формах Django.** Рассмотрим процедуру загрузки на сайт аудиофайлов на примере коротких роликов в формате MP3. Для этого создадим новую модель для описания структуры БД, форму для загрузки файлов и шаблон HTML-страницы для загрузки аудиофайлов. Откроем модуль hello/firstapp/models.py и создадим новую модель с описанием структуры таблицы БД для хранения аудиофайлов.

```
# Загрузка аудио файлов
class AudioFile(models.Model):
    title = models.CharField(max_length=100,
```

```

        verbose_name="Описание файла",)
file = models.FileField(upload_to='audios',
        verbose_name="Аудио файл",
        null=True, blank=True)
obj_audio = models.Manager()

def __str__(self):
    return self.title

```

Здесь была создана модель AudioFile, в которой описаны два поля:

- ❑ **title** — заголовок, поясняющий содержание файла;
- ❑ **file** — поле для хранения имени файла;

Для обоих полей заданы метки (verbose\_name), а также определено, что поле title обязательно для заполнения (null=False), а поле file может оставаться пустым (null=True, blank=True). Параметр upload\_to указывает расположение файлов, т. е. каталог, в котором будут храниться файлы. Нам не нужно создавать каталог мультимедиа для хранения файлов вручную, он появится автоматически, когда будут загружены файлы. В данном случае файлы будут храниться в папках media/audios/.

К этому классу добавлен метод stef\_str\_, который отобразит поле с описанием файлов в административной панели Django.

Все необходимые настройки были выполнены в разделе, описывающем загрузку изображений, и на данном этапе нужно просто запустить следующие команды для миграции созданной модели в БД:

```

python manage.py makemigrations
python manage.py migrate

```

После этого в БД будет создана таблица firstapp\_audiofile с тремя полями:

- ❑ **id** — идентификатор записи (тип поля integer — целое число);
- ❑ **title** — заголовок, поясняющий содержание файла (тип поля varchar — текстовое поле переменной длины с максимальным числом символов 100);
- ❑ **file** — поле для хранения имени файла (тип поля varchar — текстовое поле переменной длины с максимальным числом символов 100).

Поскольку file - это текстовое поле, то в нем будет храниться только имя файла, но не сам файл.

Теперь создадим форму, которая будет принимать файл в качестве входных данных от пользователя. Откроем файл hello/firstapp/forms.py и напишем в нем код.

```

from .models import AudioFile # для работы с файлами аудио

```

```

class AudioForm(forms.ModelForm):
    class Meta:
        model = AudioFile
        fields = '__all__'

```

Здесь из моделей был импортирован класс AudioFile, затем в классе meta на его основе созданы объекты model и fields (поля). В объект fields будут загружены все поля из модели AudioFile.

Теперь создадим представление (view) для обработки запросов, связанных с загрузкой файлов. Откроем модуль hello/firstapp/views.py и напишем в нем код.

```

# импорт модели и формы для работы с аудио файлами
from .models import AudioFile
from .forms import AudioForm

```

```
# загрузка аудио файлов
def form_up_audio(request):
    if request.method == 'POST':
        form = AudioForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()

    my_text = 'Загруженные аудио файлы'
    form = AudioForm()
    file_obj = AudioFile.obj_audio.all()
    context = {'my_text': my_text, "file_obj": file_obj, "form": form}
    return render(request, 'firstapp/form_up_audio.html', context)

# удаление аудио файлов из БД
def delete_audio(request, id):
    try:
        audio = AudioFile.obj_audio.get(id=id)
        audio.delete()
        return redirect('form_up_audio')
    except Person.DoesNotExist:
        return HttpResponseNotFound("<h2>Объект не найден</h2>")
```

В этом модуле на первом шаге выполнен импорт модели и формы для работы с аудиофайлами:

```
from .models import AudioFile
from .forms import AudioForm
```

Затем создана функция для загрузки файлов - `def form_up_audio(request)`. В этой функции проверяется условие, поступил ли запрос от пользователя на загрузку аудиофайла (`if request.method == 'POST'`). Если такой запрос поступил, то на основе класса `AudioForm` создается объект `form`, который получает запрос от пользователя на сохранение данных об аудиофайле (`request.POST`), и сам загружаемый файл (`request.FILES`). Если форма не содержит ошибок, то выполняется сохранение введенных пользователем данных (`form.save`). После этого происходит обновление формы для загрузки аудиофайлов.

Если форма вызывается первый раз, т. е. поступил запрос `GET`, то создаются:

- ☐ текстовая переменная `my_text`;
- ☐ объект `form`, который создается на основе класса `AudioForm` (т. е. сама форма);
- ☐ объект `file_obj`, который принимает из БД все сведения о загруженных файлах.

Затем создаётся объект `context`, в который в виде словаря передаются объекты `my_text`, `file_obj`, `form`. После этого вызывается шаблон `form_up_audio.html`, в который передаются все данные через объект `context`.

Чтобы отобразить форму для загрузки аудиофайлов, создадим в папке с шаблонами Django HTML-страницу с именем `hello/templates/firstapp/form_up_audio.html`.

```
{% extends "firstapp/base.html" %}
{% block title %}Формы{% endblock title %}
{% block header %}Изучаем формы Django{% endblock header %}
{% block content %}
<div class="container-fluid text-start my-2
    border border-5 border-warning">
    <h5>Формы - загрузка аудио файлов</h5>
    <form method="POST" enctype="multipart/form-data">
```

```

{% csrf_token %}
<div class="form-group my-2">
  {{ form.as_p }}
  <button type="submit">Загрузить</button>
</div>
</form>
{% if file_obj.count > 0 %}
  <h3> {{ my_text }} </h3>
  <table class="table table-striped table-bordered
    text-start">
    <thead>
      <tr>
        <th>№</th>
        <th>Описание файла</th>
        <th>Имя файла</th>
        <th>Удаление</th>
        <th>Прослушать</th>
      </tr>
    </thead>
    <tbody>
      {% for obj in file_obj %}
        <tr>
          <td>{{ obj.id }}</td>
          <td>{{ obj.title }}</td>
          <td>{{ obj.file }}</td>
          <td><a href="delete_audio/{{ obj.id }}">Удалить</a></td>
          <td>
            <audio controls>
              <source src="{{ obj.file.url }}"
                type="audio/mp3">
            </audio>
          </td>
        </tr>
      {% endfor %}
    </tbody>
  </table>
{% endif %}
</div>
{% endblock content %}

```

Здесь в шаблоне есть два важных тега:

- ❑ `{{ form.as_p }}` — для отображения полей, обеспечивающих загрузку файлов;
- ❑ `{% if file_obj.count > 0 %}` — для показа файлов, которые пользователь загрузил в БД.

Здесь тег `csrf_token` защитит форму от вредоносных данных, а `form.as_p` отобразит поля, предназначенные для ввода данных, в виде параграфов.

**ПРИМЕЧАНИЕ.**

Здесь используется опция `enctype = "multipart/form-data"`, которая позволяет отправлять файлы через POST-запросы. Без активации этой опции пользователь не может отправить файл через запрос POST. Эту опцию необходимо обязательно включать в формы работы с файлами и изображениями.

В теге `{% if file_obj .count > 0 %}` делается проверка, есть ли загруженные файлы. Если это условие выполняется, то в теге `<tab1>` будет создана таблица, в которой в цикле (тег `{% obj in file_obj %!}`) будут показаны все файлы, которые были загружены на сайт и находятся по соответствующим ссылкам. В столбце таблицы Удаление сделана ссылка на функцию удаления файла из БД. В столбце таблицы Прослушать находится кнопка со ссылкой на URL-адрес аудиофайла, по которому он будет запущен на прослушивание. Таким образом, в данной форме можно загружать аудиофайлы, прослушивать их и удалять с сайта.

Здесь следует обратить особое внимание на тег `<audio>`, который используется для встраивания звукового контента на HTML-страницы. Он может содержать один или более источников аудио, представленных с помощью атрибута `src` или элемента - браузер выберет один наиболее подходящих. Атрибут `src` указывает источник аудио. Это может быть относительный путь к аудио на локальном компьютере или ссылка на аудио из Интернета:

```
<audio src="weekend.mp3"></ audio >
```

Его можно заменить тегом `<source>`.

Атрибут `controls`. Когда используется атрибут `control`, то в браузере будут отображаться контроллеры управления воспроизведением, такие как "пуск", "пауза", "громкость". Например:

```
<audio
    controls
    src="weekend.mp3"
    poster="benefits-of-coding.jpg"
></audio >
```

Атрибут `loop` даёт возможность автоматически повторять аудио - заставляя его воспроизводиться снова каждый раз, когда оно завершится. Например:

```
<audio
    controls
    loop
    src="weekend.mp3"
></audio >
```

Атрибут `autoplay` позволяет настроить автоматическое воспроизведение аудио сразу после загрузки страницы. Например:

```
<audio
    controls
    loop
    autoplay
    src="weekend.mp3"
></audio >
```

Атрибут `muted` можно использовать для того, чтобы браузер не воспроизводил звук. Например:



```

<audio
    controls
    loop
    autoplay
    muted
    src="weekend.mp3"
></audio >

```

Атрибут `preload` "подскажет" браузеру, следует ли загружать аудио при загрузке страницы. Этот атрибут имеет решающее значение для взаимодействия с пользователем.

Возможны следующие значения атрибута предварительной загрузки:

- ☐ `none` — указывает, что аудио не будет загружаться, пока пользователь не нажмет кнопку воспроизведения;
- ☐ `auto` — указывает, что аудио должно загружаться, даже если пользователь не нажимает кнопку воспроизведения;
- ☐ `metadata` — указывает, что браузер должен собирать метаданные, такие как размер, продолжительность и т. д.

Вот пример использования этого атрибута:

```

<audio
    controls
    loop
    autoplay
    muted="true"
    preload="metadata"
    src="weekend.mp3"
></audio >

```

Теперь, когда и форма, и обработчик формы готовы, сопоставим форму с её URL-адресом в `urls.py`. Открываем модуль `hello/firstapp/urls.py` и добавляем в него две строки.

```

from django.urls import path
from . import views

```

```

urlpatterns = [
    path("", views.index, name='index'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
    path('my_form/', views.my_form, name='my_form'),
    path('my_form/edit_form/<int:id>', views.edit_form, name='edit_form'),
    path('my_form/delete/<int:id>', views.delete),
    path('form_up_img/', views.form_up_img, name='form_up_img'),
    path('form_up_img/delete_img/<int:id>', views.delete_img),
    path('form_up_pdf/', views.form_up_pdf, name='form_up_pdf'),
    path('form_up_pdf/delete_pdf/<int:id>', views.delete_pdf),
    path('form_up_video/', views.form_up_video, name='form_up_video'),
    path('form_up_video/delete_video/<int:id>', views.delete_video),
    path('form_up_audio/', views.form_up_audio, name='form_up_audio'),
    path('form_up_audio/delete_audio/<int:id>', views.delete_audio),
]

```

В этом программном коде была сопоставлена функция загрузки аудиофайлов (views.form\_up\_audio) с URL-адресом шаблона HTML-страницы, которая отображает саму форму ('form\_up\_audio/'), и функция удаления файла (views.delete\_audio) с её URL-адресом (form\_up\_ydf/delete\_audio/). В этой инструкции в функцию delete\_audio передаётся идентификатор той записи, которую нужно удалить из БД.

Остался последний шаг - необходимо включить вызов формы form\_up\_audio.html из меню главной страницы сайта. Для этого в базовый шаблон сайта hello/templates/firstapp/base.html добавим одну строку - ссылку на страницу form\_up\_audio.html.

```
<a href="{% url 'index' %}">Главная страница</a>
<a href="{% url 'my_form' %}">Форма</a>
<a href="{% url 'form_up_img' %}">Изображения</a>
<a href="{% url 'form_up_pdf' %}">Файлы</a>
<a href="{% url 'form_up_video' %}">Видео</a>
<a href="{% url 'form_up_audio' %}">Аудио</a>
<a href="{% url 'about' %}">О компании</a>
<a href="{% url 'contact' %}">Контакты</a>
```

Итак, у нас все готово для того, чтобы продемонстрировать загрузку аудиофайлов в БД нашего учебного сайта. Запускаем наш веб-сервер, выполнив команду `python manage.py runserver`. На главной странице сайта в главном меню появилась новая опция Аудио.



Опция в меню сайта вызова формы для загрузки аудиофайлов

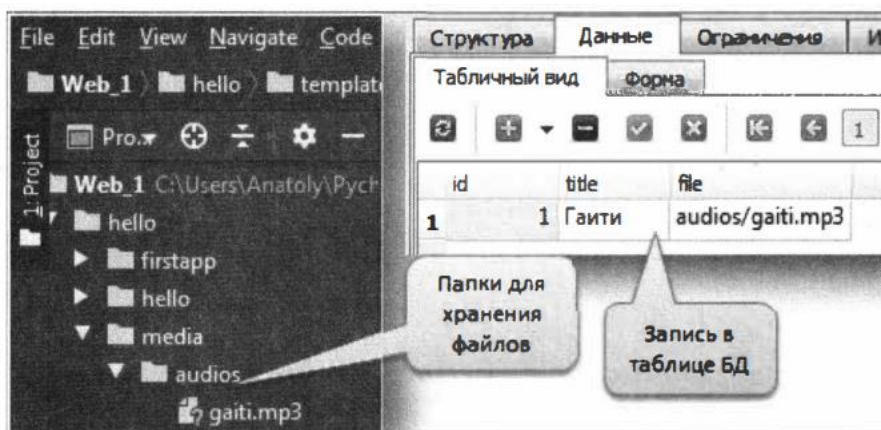
Если теперь щёлкнуть мышью на ссылке Аудио, то откроется форма для загрузки аудиофайлов.

Форма для загрузки аудиофайлов

Как видно из данного рисунка, на форме имеется поле для ввода текста с описанием файла и две кнопки: Выберите файл - для выбора файла, Загрузить - для загрузки файла в БД сайта. Если теперь выбрать какой-нибудь аудиофайл (в формате МР3), ввести его описание и нажать на кнопку Загрузить, то данный файл будет загружен на сайт, а имя загруженного сайта будет сохранено в БД. После этих действий страница будет выглядеть так.

Форма form\_up\_audio.html после загрузки файла

После этих действий в структуре проекта появится папка для хранения аудиофайлов media/audios/, где и будут сохраняться загружаемые файлы, а имена загруженных файлов будут записаны в БД в таблицу firstapp/audiofile.



Папки для хранения аудиофайлов записи в таблице БД с именами загруженных файлов

Таким образом, мы реализовали загрузку на сайт аудиофайлов, прослушивание и их удаление с сайта.

## Отчет к практическому занятию

**Требования к отчету:** скомпилированный проект со скриншотами хода работы, ответы на контрольные вопросы – файл в формате .docx, при необходимости блок-схема к созданной программе в MS Visio.

**Литература:** методические рекомендации к проведению лекционных занятий, Литература: Дронов В. А. Django. Практика создания веб-сайтов на Python - Санкт-Петербург : БХВ-Петербург, 2023. - 800 с., Постолиит А. Python, Django и PyCharm для начинающих – - Санкт-Петербург : БХВ-Петербург, 2021.

– 464 с., METANIT.COM: сайт о программировании. - URL: <https://metanit.com/sharp/> – режим доступа: свободный, инструкционная карта для проведения лекционного занятия.

**Критерии оценки:**

**Оценка «отлично»** - работа выполнена в полном объеме с соблюдением необходимой последовательности ее проведения; самостоятельно и рационально загрузил необходимое программное обеспечение, все задания выполнил в условиях и режимах, обеспечивающих получение результатов и выводов с наибольшей точностью.

**Оценка «хорошо»** - работа выполнена в полном объеме с соблюдением необходимой последовательности ее проведения; самостоятельно и рационально загрузил необходимое программное обеспечение, но задания выполнил в условиях, не обеспечивающих достаточной точности результатов, или допущено 2-3 недочета.

**Оценка «удовлетворительно»** - работа выполнена не полностью, но объем выполненной части позволяет получить правильные результаты и выводы по основным, принципиально важным задачам работы.

**Оценка «неудовлетворительно»** - работа выполнена не полностью, объем выполненной части не позволяет сделать правильных выводов.