Автор Чернов Илья

Статья для выпускного проекта курса по Python от Otus

**Оглавление**

* Описание проекта и постановка целей
* Описание используемых технологий
* Описание функций для работа с изображениями
* Создание web приложения
* Заключение

**Описание проекта и постановка целей**

Данная статья посвящена созданию сервиса конвертора изображений при помощи языка Python. Под конвертацией изображения подразумевается изменение формата изображения или изменение затрагивающее содержимое файла без смены формата. Перед началом разработки необходимо определиться какие задачи будет выполнять данный сервис:

1. Предоставление пользователям интерфейса в виде web страницы для загрузки изображения, выбора необходимого способа для конвертации изображения и получения ссылки на файл конвертированного изображения.
2. Наличие альтернативного способа конвертации при помощи api.

Для создания сервиса необходимо выполнить следующие цели:

1. Поиск средства для конвертации изображений на языке Python.
2. Реализация web интерфейса.
3. Реализация api.

**Описание используемых технологий**

Для выполнения поставленных целей использовались следующие технологии:

* Для создания web приложения был выбран Django – это py фреймворк. Относится к высокоуровневому. Позволяет осуществлять создание и поддержку безопасных веб сайтов. Характеризуется как свободный. Использует при работе шаблон проектирования MVC. Поддержка проекта осуществляется компанией Django Software Foundation.
* Для обработки изображений была выбрана библиотека Pillow, основанная на библиотеке PIL (Python Image Library).

**Описание функций для работы с изображениями**

В данном сервисе будут реализованы следующе функции:

1. Изменение формата (поддержка форматов png, jpeg, bmp, gif, ico).
2. Изменение размеров изображения.
3. Поворот изображения.
4. Вырезка фрагмента изображения.
5. Отзеркаливание изображения.

**Создание web приложения**

Для создания web приложения используется фреймворк Django поэтому начало разработки начинается с выполнения стандартных команд:

* django-admin startproject convertor для создания проекта;
* python manage.py startapp main\_app для создания приложения.

Также необходимо добавить созданное приложение в файл настроек convertor\settings.py в INSTALLED\_APPS.

После этого в созданном приложении main\_app необходимо создать модель для работы с изображениями. Для создания модели необходимо добавить в файл main\_app\models.py следующий код:

from django.db import models  
  
FORMAT\_CHOICES = (("png", "png"),  
 ("jpeg", "jpeg"),  
 ("bmp", "bmp"),  
 ("gif", "gif"),  
 ("ico", "ico"),  
 )  
  
  
class ImageModel(models.Model):  
 format = models.CharField(max\_length=5, choices=FORMAT\_CHOICES, default="jpeg", blank=True)  
 img = models.ImageField(upload\_to="images/before\_convert/")  
 new\_size = models.CharField(max\_length=30, blank=True)  
 degree = models.IntegerField(default=0, blank=True)  
 crop\_coordinates = models.CharField(max\_length=50, blank=True)

Созданная модель ImageModel хранит изображение и параметры для его конвертации. Модель состоит из следующих полей:

1. format – новый формат изображения.
2. img – изображение.
3. new\_size – новый размер изображения.
4. degree – угол поворота.
5. crop\_coordinates – координаты фрагмента изображения.

Так как при разных способах конвертации требуются разные параметры, то поля отвечающие за параметры конвертации созданы необязательными (blank = True).

Для удобства пользователя выбор формата для конвертации можно сделать в виде выпадающего списка, для этого при создании поля format укажем параметр choices.

В поле img важным является параметр upload\_to, с помощью этого параметра указывается папка для загрузки изображения.

Чтобы работала загрузка изображений необходимо внести правки в файл convertor\settings.py.

STATICFILES\_DIRS = (os.path.join(BASE\_DIR, "static/"), os.path.join(BASE\_DIR, "static/images"))  
STATIC\_URL = '/static/'  
MEDIA\_ROOT = os.path.join(BASE\_DIR, "static/")  
MEDIA\_URL = ''

Функции для конвертации изображения для удобства реализованы методами модели для хранения изображения. Все они схожи и поэтому для краткости рассмотрим одну из них, а именно функцию смены формата (с остальными можно ознакомиться в репозитории на github по ссылке в конце статьи).

Функция смены формата:

from PIL import Image

# Пропущен код со

class ImageModel(models.Model):

# Пропущен код полей

def convert(self, for\_api=False):  
 img = Image.open(self.img)  
 rgb\_img = img.convert('RGB')  
 img\_name = f"static/{str(self.img).split('.')[0]}.{self.format}"  
 if self.format not in [f[0] for f in FORMAT\_CHOICES]:  
 raise ValueError("bad format")  
 rgb\_img.save(img\_name, self.format)  
 if for\_api:  
 return img\_name  
 else:  
 return img\_name, rgb\_img.size

Данный метод при помощи библиотеки pillow создает RGB представление изображения, на основе которого создает изображение нового формата.



Рисунок 1 – Алгоритм конвертации изображения

Предполагается что данный метод будет использоваться как при работе с web страницей, так и при работе с api. Поэтому у метода есть флаг for\_api. Разница лишь в том, что для web страницы помимо имени изображения (на его основе будет формироваться ссылка на скачивание) необходим размер картинки для корректного отображения.

Итак, мы имеем модель для хранения изображения с методом для смены формата. Теперь для этого метода необходимо создать форму, представление и поддержку api.

Для создания формы необходимо добавить в файл main\_app\forms.py следующий код:

from django import forms  
from .models import ImageModel  
  
  
class ChangeFormatForm(forms.ModelForm):  
 class Meta:  
 model = ImageModel  
 fields = ("img", "format")  
  
 def \_\_init\_\_(self, \*args, \*\*kwargs):  
 super(ChangeFormatForm, self).\_\_init\_\_(\*args, \*\*kwargs)  
 for key in self.fields:  
 self.fields[key].required = True  
  
 def clean(self):  
 cleaned\_data = super().clean()  
 format = cleaned\_data.get('format')  
 img = cleaned\_data.get('img')  
 if format == str(img).split(".")[-1]:  
 raise forms.ValidationError(f'Your choose same format {str(img).split(".")[-1]} ')  
 else:  
 return cleaned\_data

Данная форма будет использована для получения данных при смене формата, поэтому в ней не нужны все поля модели. Выбор необходимых полей осуществляется при помощи параметра fields.

Валидация данных в форме осуществляется при помощи метода clean. В данном методе выполняется проверка на то, что формат исходного изображения не совпадает с форматом после конвертации.

Проверка того что новый формат является поддерживаемым не нужна, так как в форме это поле реализовано через выпадающий список из-за наличия параметра choices.

Данную форму необходимо использовать в шаблоне web страницы. Использование формы в шаблоне выглядит следующим образом (фрагмент из main\_app\templates\load\_img.html):

<form method="post" enctype='multipart/form-data'>  
 {% csrf\_token %}  
 {{ form.as\_p }}  
 <button type="submit">Load</button>  
 </form>

После создания формы необходимо создать представление для этого необходимо добавить в файл main\_app\views.py следующий код:

from django.views.generic.edit import FormView

from .forms import ChangeFormatForm

from .models import ImageModel

class ChangeFormatImg(FormView):  
 template\_name = 'load\_img.html'  
 form\_class = ChangeFormatForm  
 success\_url = '/'  
  
 def form\_valid(self, form):  
 context = {}  
 if form.data['format'] and "img" in self.request.FILES.keys():  
 img\_obj = ImageModel.objects.create(format=form.data['format'], img=self.request.FILES['img'])  
 img\_obj.save()  
 img\_name, size = img\_obj.convert()  
 context['img'] = img\_name  
 context['size'] = size  
 return render(self.request, 'get\_result.html', context)

Данное представление является class based view и наследуется от FormView, что позволяет удобно работать с ранее созданной формой. Действия которые происходят после успешной проверки формы описаны в методе form\_valid. Из формы извлекаются данные и на их основе создается объект модели ImageModel, при помощи этого объекта выполняется изменение формата изображения. Метод convert() возвращает путь к изображению после конвертации и ра­змер для его отображения на странице с результатом. Эти данные передаются в контекст для шаблона web страницы.

Страница с результатом исполь­­зует шаблон main\_app\ templates\ get\_result.html.

<img src="\{{ img }}" height={{ size.1 }} width={{ size.0 }}>  
<br>  
<a href="\{{ img }}" download="\{{ img }}"> Результат конвертации вашего файла </a>  
<a href="\{{ img }}"><img src="/static/images/download.png" alt="Whitesquare logo" height=40 width=40></a>

Данный шаблон отображает изображение после конвертации, а также показывает ссылку на скачивание результата.

Чтобы проверить созданное представление необходимо настроить маршрутизацию url. Для этого необходимо в файл main\_app\urls.py добавить следующий код:

from django.urls import path, include  
from .views import ChangeFormatImg

urlpatterns = [  
 path("/change\_format/", ChangeFormatImg.as\_view(), name="change\_format"),  
]

Также для настройки маршрутизации необходимо в файл covertor\urls.py добавить следующий код:

from django.contrib import admin  
from django.urls import path, include  
from django.conf import settings  
from django.conf.urls.static import static  
  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path("", include("main\_app.urls"), name='main'),  
] + static(settings.STATIC\_URL, document\_root=settings.STATIC\_ROOT) \  
 + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

Результат конвертации файлов представлен на рисунках 2-3. Расширение скаченного файла совпадает с выбранным форматом jpeg.

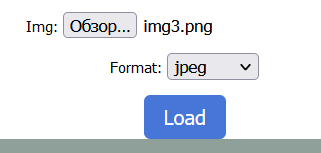


Рисунок 2 – Ввод данных в форму

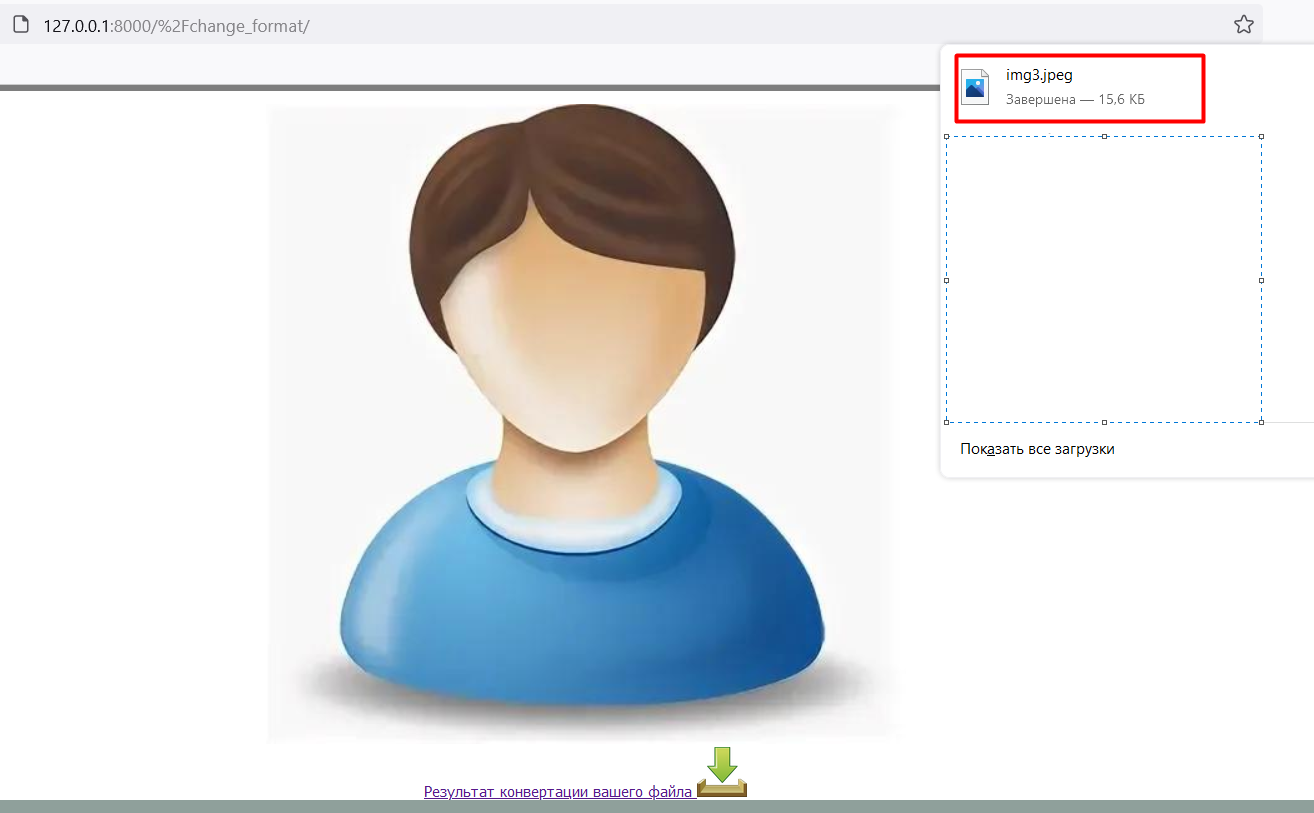


Рисунок 3 – Результат конвертации

Проверим валидацию формы.

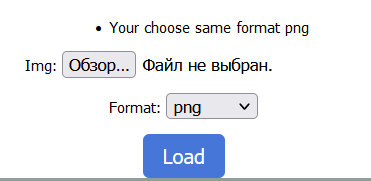


Рисунок 4 – Проверка валидации (ввод исходного формата)

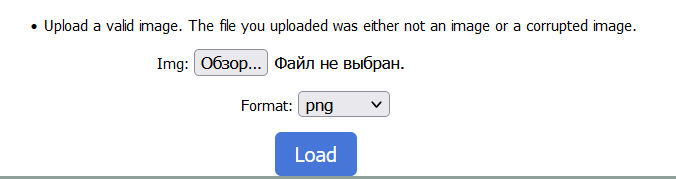


Рисунок 5 – Проверка валидации (файл не является изображением)

Теперь реализуем этот функционал для api, для этого необходимо создать сериалайзер, преставление и настроить маршрутизацию url. Для реали­зации сериалайзера необходимо в файл main\_app\serializers.py добавить следующий код:

from rest\_framework import serializers  
from .models import ImageModel  
  
  
class ImgSerializer(serializers.ModelSerializer):  
 img = serializers.ImageField(max\_length=None, use\_url=False)  
  
 class Meta:  
 model = ImageModel  
 fields = '\_\_all\_\_'

Так как данный сериалайзер будет использован для всех функций конвертации изображений, он включает все поля модели ImageModel.

Далее добавим представление, для этого добавим в файл main\_app\views.py следующий код:

from rest\_framework import mixins, viewsets  
from rest\_framework.response import Response

from .serializers import ImgSerializer

def validate\_format(format):  
 if format in [f[0] for f in FORMAT\_CHOICES]:  
 return True  
 else:  
 return False

class ChangeFormatImgAPI(mixins.ListModelMixin, viewsets.GenericViewSet):  
 serializer\_class = ImgSerializer  
 queryset = ImageModel.objects.none()  
  
 def post(self, request, \*args, \*\*kwargs):  
 serializer = ImgSerializer(data=self.request.data)  
 if serializer.is\_valid():  
 try:  
 format = serializer.validated\_data['format']  
 except KeyError as err:  
 return Response(str(err), status=HTTPStatus.BAD\_REQUEST)  
 img = serializer.validated\_data['img']  
 if format == str(img).split(".")[-1]:  
 return Response("Same format", status=HTTPStatus.BAD\_REQUEST)  
 if not validate\_format(str(img).split(".")[-1]):  
 return Response("Bad file", status=HTTPStatus.BAD\_REQUEST)  
 if not validate\_format(format):  
 return Response("Bad format", status=HTTPStatus.BAD\_REQUEST)  
 img\_obj = ImageModel.objects.create(format=format, img=img)  
 img\_obj.save()  
 try:  
 img = img\_obj.convert(for\_api=True)  
 except ValueError as err:  
 return Response(str(err), status=HTTPStatus.BAD\_REQUEST)  
 return Response({'img\_url': img}, status=HTTPStatus.OK)  
 return Response(serializer.errors, status=HTTPStatus.BAD\_REQUEST)

Так как представление для api не использует форму, валидация вынесена в представление. В случае успешного запроса, возвращается ссылка на конвертированное изображение. Далее необходимо настроить маршрутизацию, для этого необходимо внеси правки в файл main\_app\urls.py.

from rest\_framework import routers

router = routers.DefaultRouter()  
router.register('change\_format', ChangeFormatImgAPI, basename='api\_change\_format')

urlpatterns = [  
 path("/change\_format/", ChangeFormatImg.as\_view(), name="change\_format"),  
 path(r'api/', include(router.urls), name="api"),  
]

Проверка работы api выполнена на рисунках 6-7:



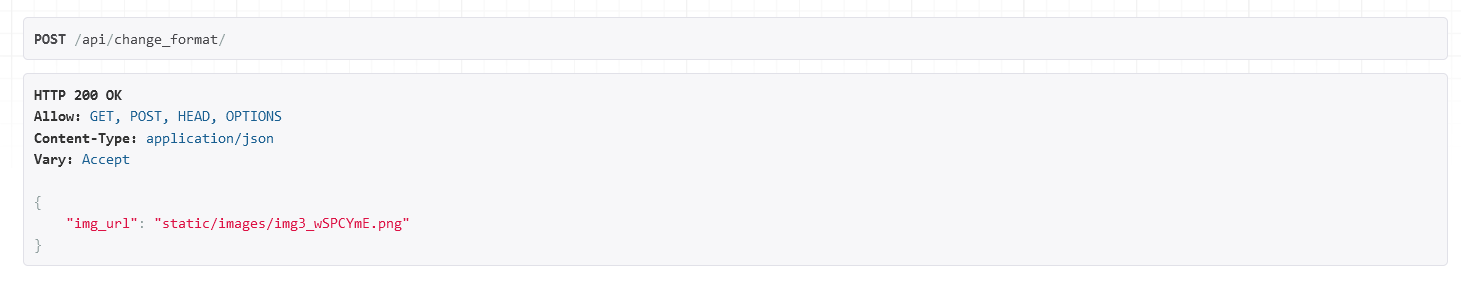


Рисунок 6 – Проверка работы api

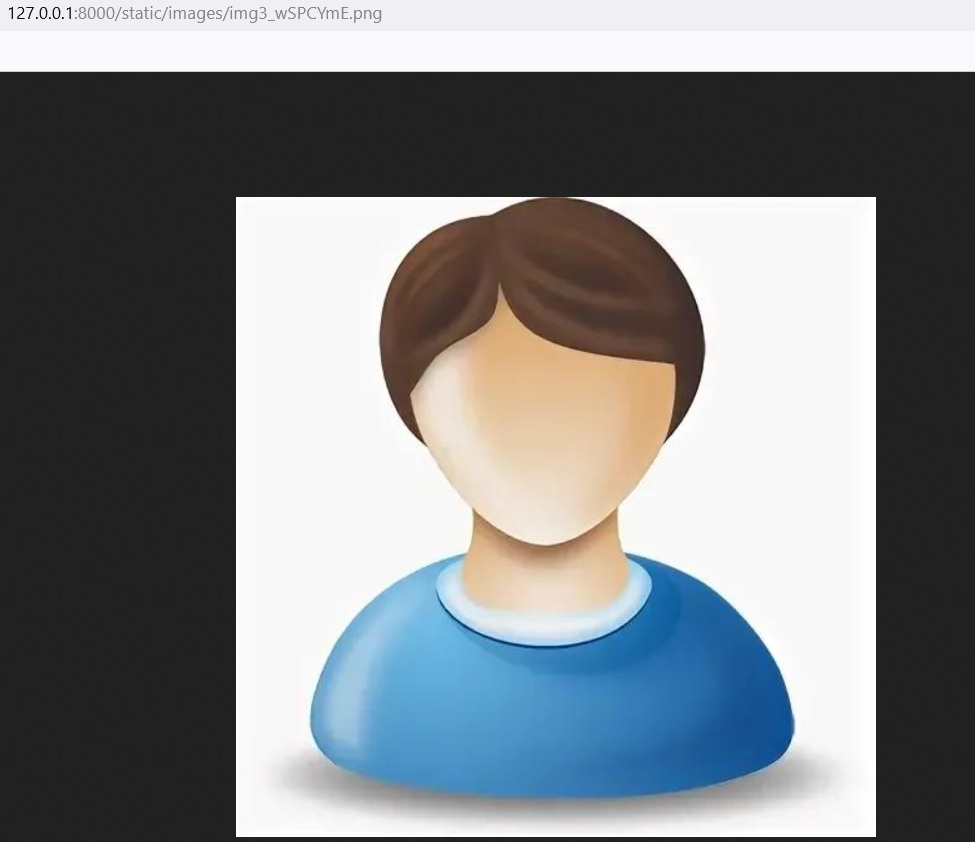


Рисунок 7 – Проверка ссылки

Далее проверим валидацию:



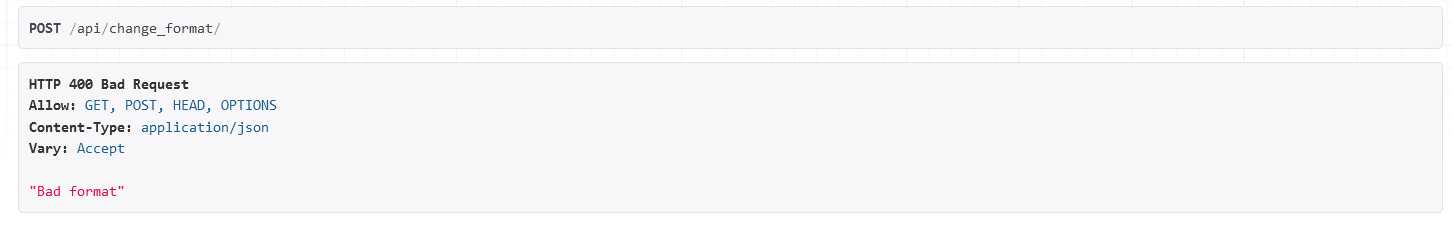
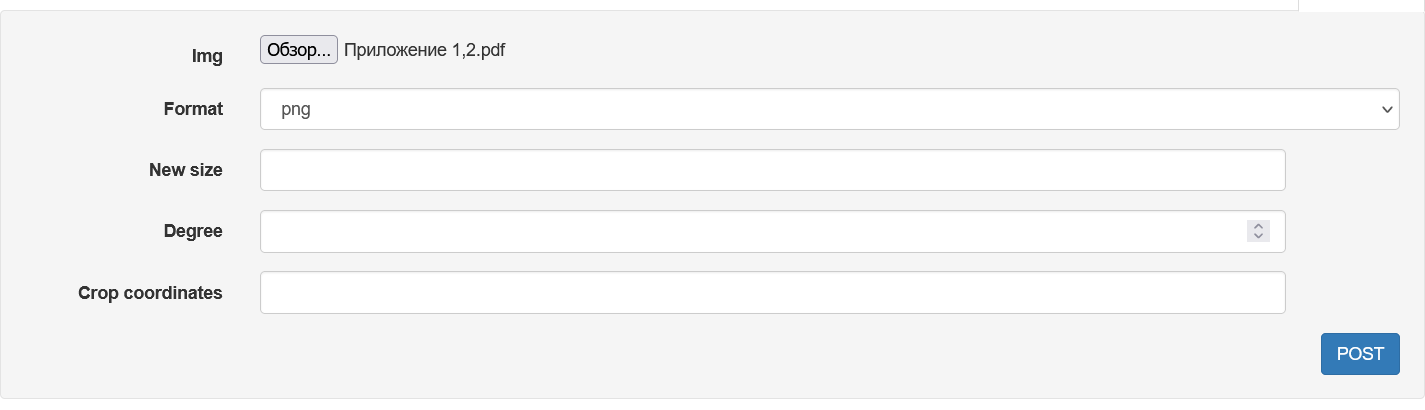


Рисунок 8 – Проверка валидации (нет формата)



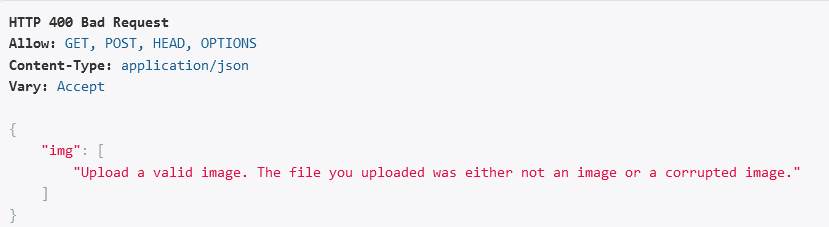


Рисунок 9 – Проверка валидации (не корректный файл)

**Заключение**

В ходе написания данной статьи, был разработан сервис для конвертации изображений. Проект расположен по ссылке <https://github.com/7361cid/image_conventor>. В ходе разработки пришлось улучшить знания по фреймворку Django, а также узнать о способах обработки изображений на языке Python.