Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

ФАКУЛЬТЕТ Информатика и системы управления
КАФЕДРА Компьютерные системы и сети

Отчет по преддипломной практике

на предприятии ООО «НПЦ ИСУ»

Студент _____ **Д.Н. Акимов**
(Подпись, дата) (И.О.Фамилия)

Руководитель _____ **Е.В. Смирнова**
(Подпись, дата) (И.О.Фамилия)

Москва, 2016

Оглавление

Введение

Цель поставленная при прохождении преддипломной практики – создание приложения, автоматизирующего печать документов по шаблону. Проблема состояла в том что было необходимо составить большое количество одинаковых документов и их заполнение в ручную отнимало много времени. Для упрощения процесса было разработано веб-приложение “Автоматизированная печать документов”. Оно позволило значительно облегчить задачу заполнения документов. В отчете представлены этапы создания приложения, описаны методы его реализации.

1 Создание проекта

Для начала работы был создан проект, для того что бы его создать нужно было ввести в консоли команду:

$ django-admin startproject mysite, где mysite – название проекта.

При помощи команды mysite были созданы некоторые стандартные файлы, рассмотрим их ниже:

```
mysite/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

Внешний каталог mysite/ – это просто контейнер для проекта. Его название никак не используется Django, его можно переименовывать как угодно.

manage.py: Скрипт, который позволяет взаимодействовать с проектом Django.

Внутренний каталог mysite/ - это пакет Python моего проекта. Его название – это название пакета Python, которое я использовал для импорта чего-либо из проекта (например, mysite.urls).

mysite/__init__.py: Пустой файл, который указывает Python, что текущий каталог является пакетом Python.

mysite/settings.py: Настройки/конфигурация проекта.

mysite/urls.py: Конфигурация URL-ов для моего проекта Django. Это “содержание” всех Django-сайтов.

Сервер для разработки приложения (локальный сервер):

Для того что бы запустить локальный сервер разработки нужно было ввести в консоли команду:

$ python manage.py runserver

После чего в консоли был получен следующий ответ:

```
Performing system checks...

System check identified no issues (0 silenced).

You have unapplied migrations; your app may not work properly until they are applied.
Run 'python manage.py migrate' to apply them.

March 31, 2016 - 15:50:53
Django version 1.9, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Рисунок 1 – Ответ от сервера после его первого запуска.

Как видно из Рисунка 1 – сервер запустился на адресе 127.0.0.1:8000. Это специальный сервер для разработки, где далее мною так же велась отладка проекта.

2 Создание Супер-пользователя (Администратора проекта)

В Django имеется возможнось создать стандартного супер-пользователя, это я сделал с помощью команды:

$ python manage.py createsuperuser

Далее в консоли мне было предложено выбрать логин и пароль для администратора, я выбрал root root – логин и пароль соотверственно.

После чего переходим по ссылке 127.0.0.1:8000/admin и заходим на страницу администратора. Отсюда я могу управлять проектом, добавлять или удалять записи из базы данных, регистрировать пользователей, давать или убирать у них права.

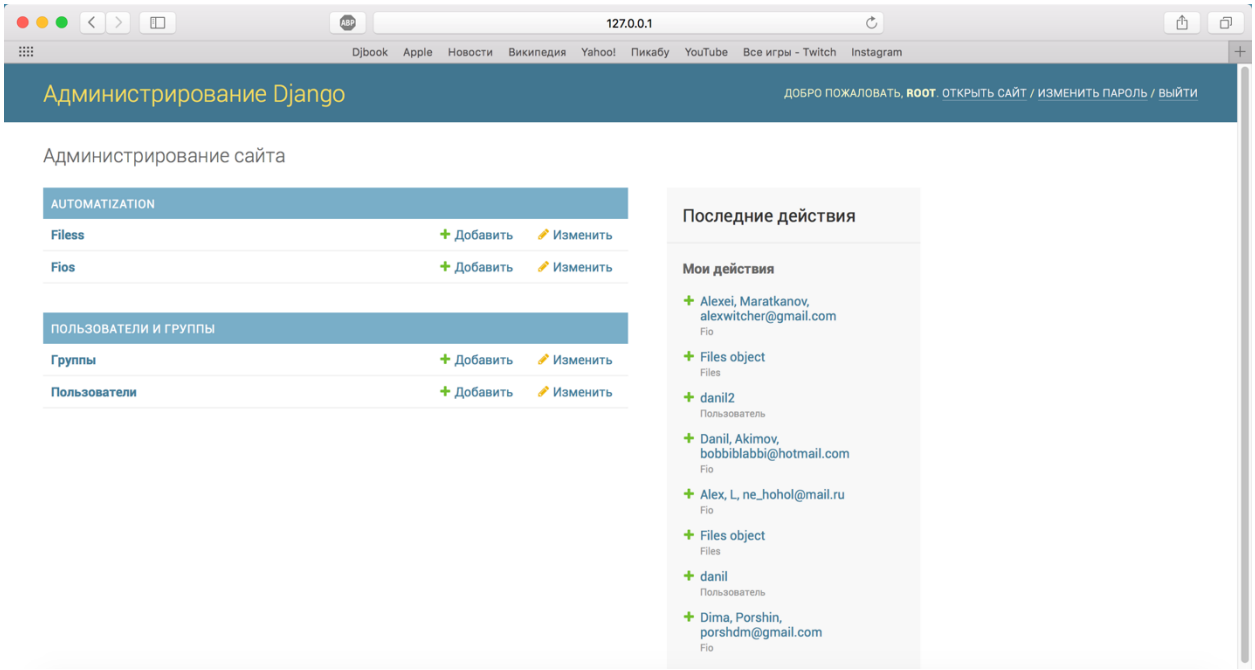

Рисунок 2 – Страница Супер пользователя.

3 Настройка базы данных и создание моделей

Для создания базы данных я использовал команду:

$ python manage.py syncdb

Данная команда создает базу данных и далее предлагает нам выбрать логин и пароль для администратора нашей БД.

Так же в проекте создался файл models.py в котором я создал подели базы данных. Для того что бы применить созданные модели я воспользовался командой:

$ python manage.py migrate

Данная команда мигрировала созданные мной модели.

```
class FIO(models.Model):
    name = models.CharField(max_length=50)
    surname = models.CharField(max_length=50)
    email = models.CharField(max_length=50)

    def __unicode__(self):
        return '{}, {}, {}'.format(self.name, self.surname, self.email)


class Files(models.Model):
    which_user = models.ForeignKey(User)
    upload = models.FileField(upload_to='uploads/')
```

Рисунок 3 – Модели базы данных.

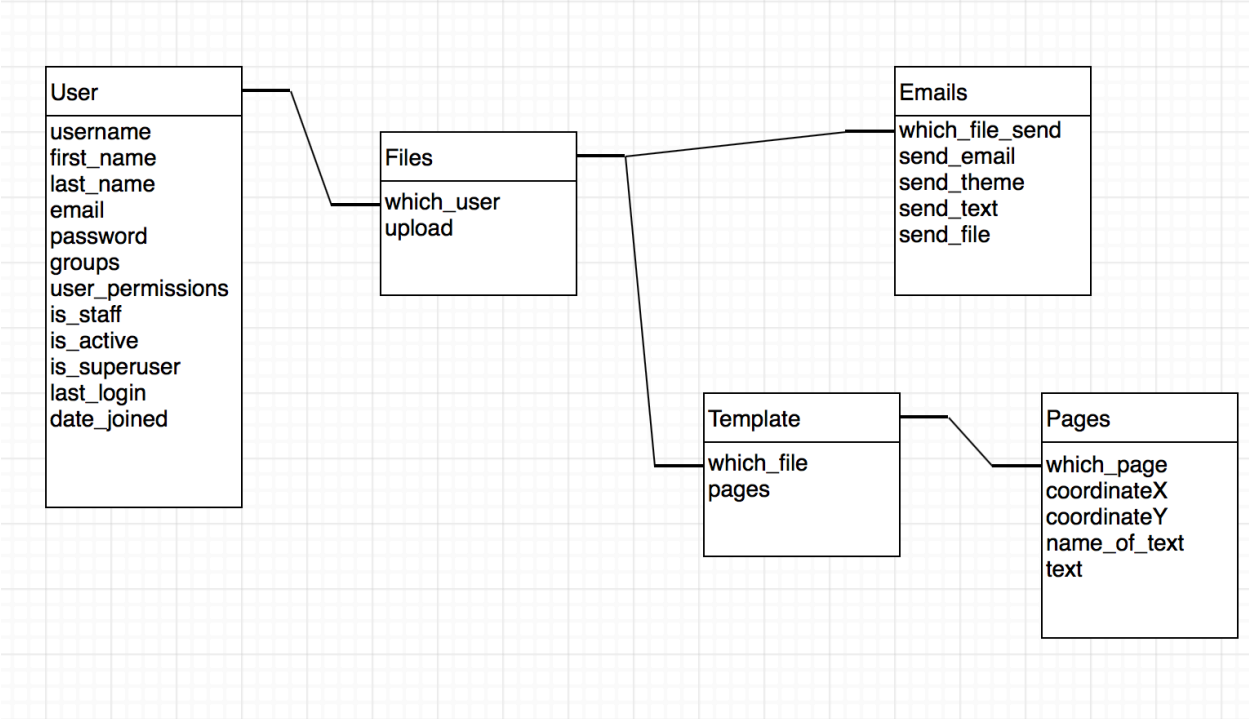

Рисунок 4 – Даталогическая модель БД.

В результате структура базы данных получилась следующая (Рисунок 4):

Модель User – стандартная модель пользователя в Django, в ней достаточно много полей, однако основными являются поля username (логин), first_name (имя), last_name (фамилия) и password (пароль). Модели User связана с моделью Files, связь реализована с помощью ключа, в данном случае ключом является which_user, в данной модели находится файловое поле upload, именно в нем хранятся загруженные файлы. Для каждого файла создается шаблон, поэтому необходимо связать модель Files с данными шаблона – моделью Templates. Для этого так же используется ключ, в модели Template хранятся данные о количестве страниц – pages. У каждой страницы имеются свои координаты, по которым будут заполнятся данные. Для этих данных используется модель Page – она хранит в себе данные координат X,Y описание поля в которое будет вводиться информация и данные самого поля, связана модель Page с моделью Template с помощью ключа which_page. Также модель Files связана с моделью Emails при помощи ключа which_file_send. Emails хранит в себе данные исходящего письма – адрес отправки, текст письма, его тему, а так же прикрепленный файл.

4 Разработка пользовательского интерфейса и использование фреймворка Bootstrap

Для создания удобного интерфейса мною был использован фреймворк Bootstrap.

Bootstrap (также известен как Twitter Bootstrap) — свободный набор инструментов для создания сайтов и веб-приложений. Включает в себя HTML и CSS шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая JavaScript-расширения.

Для подключения фрейиворка необходимо было скачать css файлы, которые я далее подключил к html страницам приложения, для этого и использовал тег <script>:

```
<script src={% static "js/jquery.min.js" %}></script>
<script
src="//netdna.bootstrapcdn.com/bootstrap/3.3.2/js/bootstrap.min.js"></script>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js"></scr
ipt>
<script src="../../dist/js/bootstrap.min.js"></script>
<script src="offcanvas.js"></script>
```

Рисунок 5 – Подключение Bootstrap.

Так же были использованы стандартный Header и стили кнопок, представленные на рисунках 5 и 6:

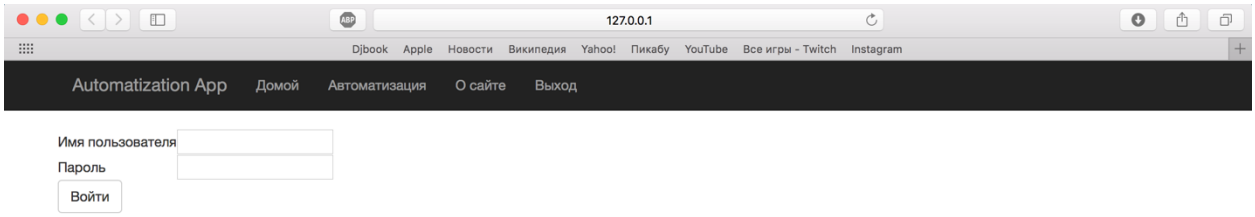

Рисунок 6 – Страница авторизации

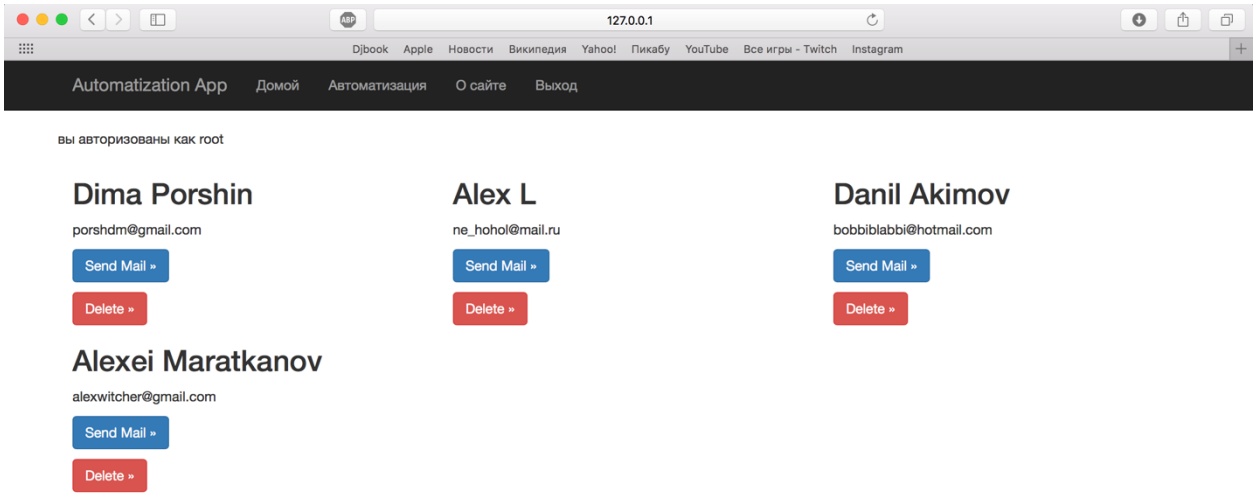

Рисунок 7 – страница пользователей, которым можно отправлять документы.

Фреймворк значительно облегчил задачу по визуализации веб-приложения.

5 Библиотека ReportLab для работы с документами

Для того что бы работать с документами я выбрал библиотеку ReportLab.

ReportLab – это open source (BSD license) библиотека, позволяющая создавать PDF-документы любой сложности напрямую из Python’а. Кроме того, она умеет отрисовывать всевозможные графики и диаграммы как в различные растровые и векторные форматы файлов так и в PDF. Документ создается набором высокоуровневых команд, что делает ReportLab чрезвычайно привлекательным инструментом для динамической генерации отчетов баз данных. Показательно, что NASA интенсивно использует ReportLab в своей повседневной работе.

ReportLab кроссплатформенный пакет, написан большей частью на Python’е и немного на C. Работает довольно быстро – должен вполне адекватно функционировать на слабых компьютерах. Утверждается, что работает и на win9x платформах – сам еще не успел проверить, но знаю, что для многих разработчиков отчетов к БД это важный момент.

ReportLab позволяет оперировать как простейшими командами типа “разместить на листе рисунок/линию/текст с координатами x,y” так обладает и более продвинутыми инструментами для разметки параграфов, колонтитулов, конструирования и автозаполнения таблиц, расстановки номеров страниц.

Основная идея заключается в том что я использую два документа, первый это шаблон, а второй документ на котором рисуются строки с помощью функции drawstring(), после чего два документа склеиваются и сохраняются.

Сначала функция canvas.Canvas() создает новый документ, в котором далее будут рисоваться. Далее функция drawstring() рисует строки по координатам X и Y. Функция showPage() закрывает страницу (действия на ней заканчиваются), после этого save() сохраняет документ. Итак, первый документ готов.

Теперь для слияния двух документов используется функции mergePage() – она постранично склеивает два документа. В конце функция close() закрывает и сохраняет конечный документ.

Пример кода функции для формирования PDF документа:

```
def save_ready_template(request, id):
    person_print = FIO.objects.get(id=id)
    packet = StringIO.StringIO()
    # create a new PDF with Reportlab
    can = canvas.Canvas(packet, pagesize=letter)
    can.drawString(284, 579, "{} {}".format(person_print.name, person_print.surname))
    can.showPage()
    can.drawString(260, 494, "{} {}".format(person_print.name, person_print.surname))
    can.showPage()
    can.save()
    # move to the beginning of the StringIO buffer
    packet.seek(0)
    new_pdf = PdfFileReader(packet)
    # read your existing PDF
    existing_pdf = PdfFileReader(file("/Users/danilakimov/Desktop/template1.pdf", "rb"))
    output = PdfFileWriter()
    # add the "watermark" (which is the new pdf) on the existing page
    page = existing_pdf.getPage(0)
    page.mergePage(new_pdf.getPage(0))
    output.addPage(page)
    page = existing_pdf.getPage(1)
    page.mergePage(new_pdf.getPage(1))
    output.addPage(page)
    # finally, write "output" to a real file
    outputStream = file("/Users/danilakimov/Desktop/readytemplate.pdf", "wb")
    output.write(outputStream)
    outputStream.close()
    return render(request, 'template_page.html', {'person_template': person_print})
```

Рисунок 9 – функция формирования готового документа по шаблону.

6 Использование SMTP сервера для отправки документов

Для отправки сформированного документа я решил использовать SMTP сервер.

SMTP (англ. Simple Mail Transfer Protocol — простой протокол передачи почты) — это широко используемый сетевой протокол, предназначенный для передачи электронной почты в сетях TCP/IP.

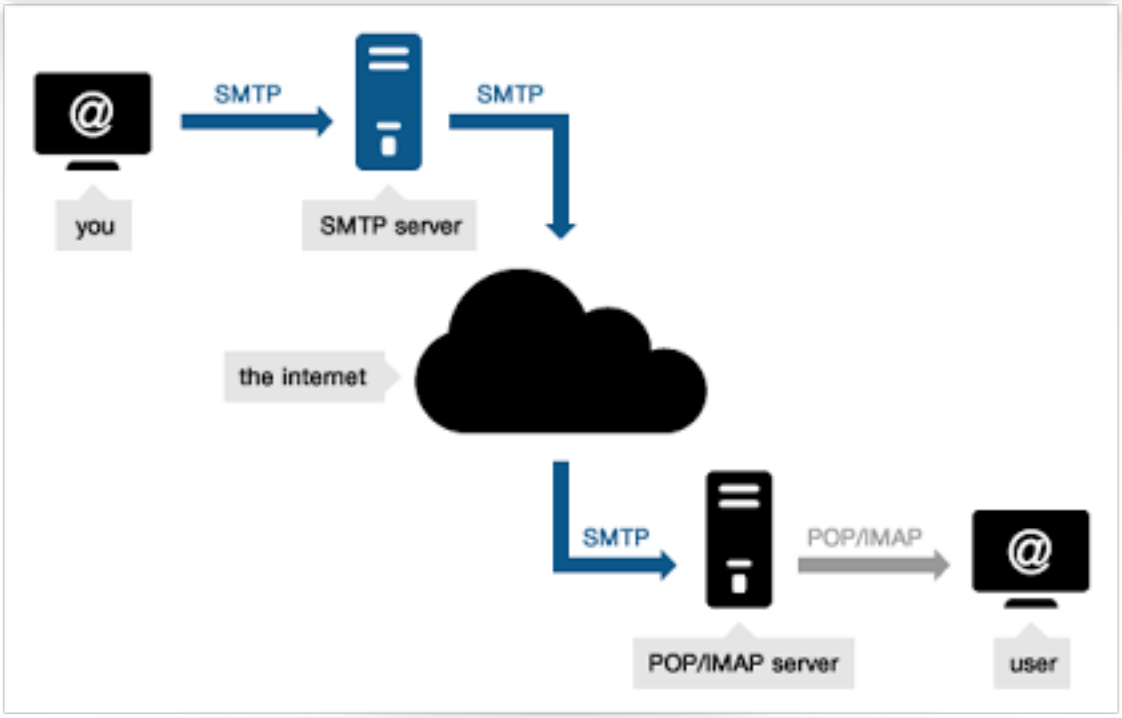

Рисунок 10 – Схема работы SMTP сервера.

Пример функции отправки исходящего письма с вложением:

```
def send_email(request, id):
    person_print = FIO.objects.get(id=id)
    # creating email
    subject, from_email, to = u'Письмо', 'bobbiblabbi@gmail.com', person_print.email
    text_content = u"Письмо".format(person_print.name, person_print.surname)
    msg = EmailMultiAlternatives(subject, text_content, from_email, [to])
    msg.attach_file('/Users/danilakimov/Desktop/readytemplate.pdf')
    msg.send()
    return render(request, 'template_page.html', {'person_template': person_print})
```

Рисунок 11 – функция отправки письма с вложением (документом).

С помощью функции EmailMultiAlternatives() я создаю письмо. В функции указывается Тема письма, Текст в письме, Почта с которой отправляется письмо и Почта на которую оно отправляется. Далее выполняется функция attach_file() – прикрепление вложения (документа) к письму. После чего функция send() отправляет письмо.

Для того что бы сервер заработал, необходимо к нему подключиться.

Подключение производится в файле settings.py. Я указал сервер, почту, а так же пароль от нее.

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'

# Host for sending e-mail.
EMAIL_HOST = 'smtp.gmail.com'

# Port for sending e-mail.
EMAIL_PORT = 587

# Optional SMTP authentication information for EMAIL_HOST.
EMAIL_HOST_USER = 'bobbiblabbi@gmail.com'
EMAIL_HOST_PASSWORD = ''
EMAIL_USE_TLS = True
```

Рисунок 13 – Настройки подключения к серверу.

7 Структура Веб-приложения

В результате проделанной работы веб приложение имеет следующую структуру:

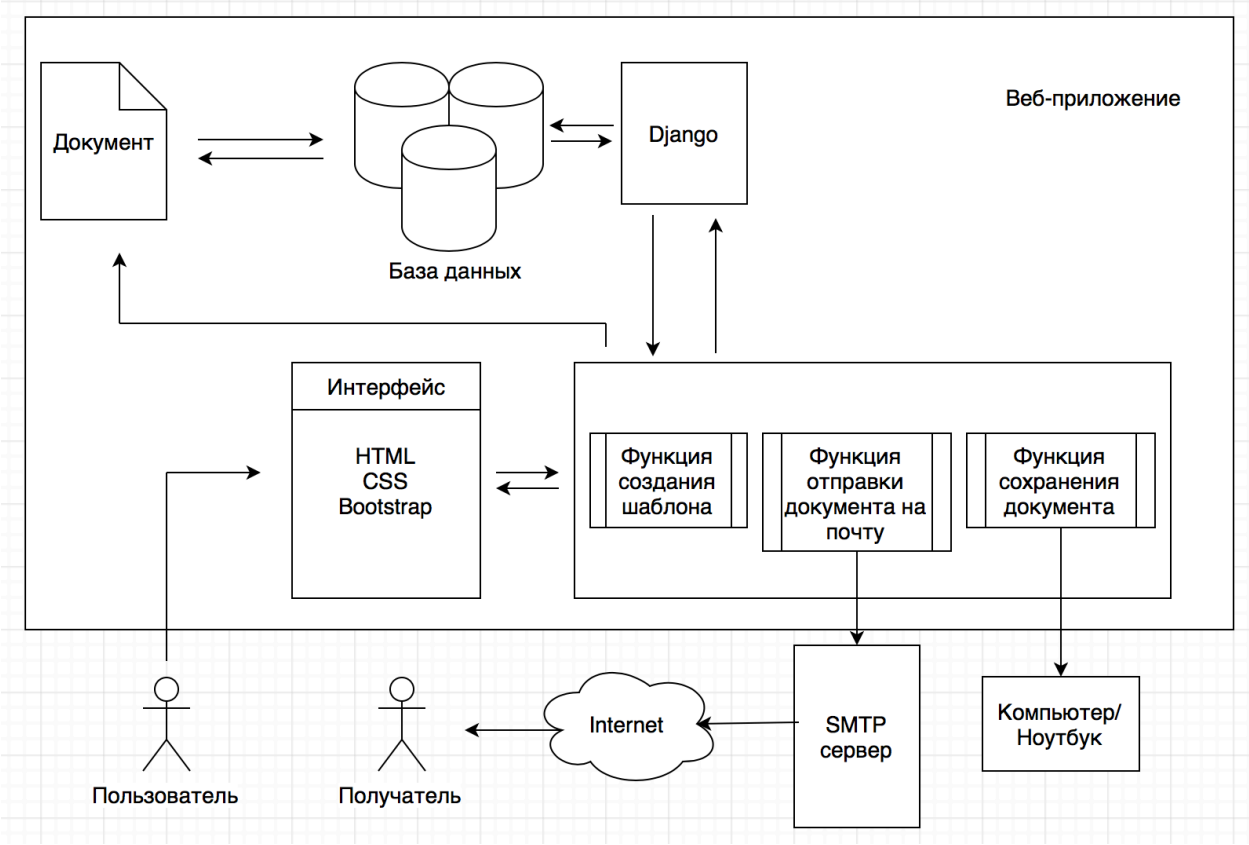

Рисунок 14 – Структурная схема работы Веб-приложения

Пользователь заходит в приложение, за интерфейс отвечает HTML, фреймворк Bootstrap и стили CSS. Пользователь может загрузить свой документ на сервер, в базу данных, БД общается с приложением при помощи Django. Приложение имеет функции создания шаблона для загруженного документа (данные документа – кол-во страниц, координаты размещения строк для заполнения, хранятся в БД), это реализовано при помощи библиотеки ReportLab, документы обрабатываются в формате PDF. Так же имеется функция сохранения и отправки документа по почте. Документ отправляется через SMTP сервер. Все функции Веб-приложения написаны на языке Python.

Вывод

В результате выполненной работы мною было создано Веб-приложение для автоматизации печати документов, а так же их отправки по почте. Приложение значительно ускорило работу над составлением документов за счет того что не приходится в ручную искать и вбивать текст в нужных местах. Шаблон для документа делается один раз, далее необходимо только вбивать нужные поля.