Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ *Робототехники и комплексной автоматизации*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Студент	Петричук Анастасия Олеговна
Группа	РК6-61б
Тип практики	Производственная
Название предприятия	ПАО «Туполев»

Студент _____ **Петричук А.О.**
подпись, дата *фамилия, и.о.*

Руководитель практики _____ _____
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2021 г.

Индивидуальное задание

Спроектировать и реализовать веб-приложение для существующей базы данных. Должен присутствовать следующий функционал:

1) просмотр лицензий на всех сайтах с возможностью фильтрации данных по каждому отдельному сайту.
2) просмотр свободных лицензий с возможностью фильтрации по сайтам и по типу лицензий
3) просмотр сводной таблицы пользователей
4) обработка заявок на выдачу лицензий

Детали реализации и информация баз данных приведена не будет, так как данная информация принадлежит ПАО «Туполев» и является засекреченной.

Оглавление

Введение

Производственная практика проходила в ПАО «Туполев». Предприятие занимается разработкой, производством, испытаниями, ремонтом и поддержанием лётной годности авиационной техники. В настоящее время ПАО «Туполев» является предприятием, способным обеспечивать все стадии жизненного цикла авиационной техники от разработки до серийного производства, модернизации, ремонта, послепродажного обслуживания и поддержки эксплуатации. Значительная часть инженеров компании работает на ПО компании Siemens NX Teamcenter. Для распределения лицензий на ПО между сотрудниками используются базы данных, импортированные из Teamcenter с помощью скриптов. Было важно обеспечить удобную и быструю работу с полученными данными. Для этого решено было разработать веб-приложение с бэкендом на Python, и фронтовой частью с использованием Bootstrap и JavaScript.

В результате прохождения практики были получены новые и закреплены уже имеющиеся знания в сфере веб-разработки. А также был получен опыт работы с языком JavaScript.

Основная часть

Обзор используемых в разработке средств

Дадим некоторые определения:

1) **Python** – высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Язык является полностью объектно-ориентированным — всё является объектами.
2) **Django** - свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC[8]. Проект поддерживается организацией Django Software Foundation.
3) **MVC** - Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения, и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.
4) **Модель** (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.
5) **Представление** (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели.
6) **Контроллер** (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.
7) **JavaScript**(также js) - мультипарадигменный язык программирования.

Поддерживает объекториентированный, императивный и функци ональный стили. Обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

8) **SQL** - (англ. structured query language — «язык структурированных запросов») — декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.
9) **MySQL** - свободная реляционная система управления базами данных.
10) **БД** – база данных.
11) **Модуль** — это один файл Python.
12) **Пакет** — это каталог таких модулей. В отличие от простых директорий, содержащих скрипты Python, пакет содержит еще и дополнительный файл — _init_.py.

Описание архитектуры проекта

Разработка приложения велась на языке Python. Скрипты в клиентской части приложения были написаны с использованием JavaScript. Верста осуществлялась с помощью CSS, HTML и Bootstrap. Запросы к базе данных были написаны на диалекте MySQL.

При создании нового проекта с использованием Django, автоматически создается следующая структура проекта:

```
TUPOLEV_APP
|
|   manage.py
|
+---tupolev_app
|   |   asgi.py
|   |   settings.py
|   |   urls.py
|   |   views.py
|   |   wsgi.py
|   |   __init__.py
|   |
|   \---static
|
\---templates
```

Структура проекта при создании приложения.

Корневая папка (в данном случае TUPOLEV_APP), в которой находится главный пакет (tupolev_app), пустая папка templates, в которой в дальнейшем будут создаваться шаблоны страниц, и модуль manage.py, который является управляющим скриптом для всего проекта. Этот скрипт на языке Python сгенерирован фреймворком Django автоматически. В пакете tupolev_app находятся файлы, управляющие проектом в целом: settings.py, в котором объявлены основные переменные для настройки приложения, urls.py, который является главным контроллером и передает управление между контроллерами других приложений, а также папка static, в которую будут помещены статические файлы (например, css и js), общие для всего проекта. В данном случае база данных находилась на другой машине и подключение к ней осуществлялось по сети.

После настройки главного пакета в корне приложения был создан пакет Licenseservers, имеющий следующую структуру:

```
Licenseservers
|   admin.py
|   apps.py
|   models.py
|   tests.py
|   urls.py
|   views.py
|   __init__.py
|
+---migrations
|
+---static
|
\---templates
```

Структура приложения Licenseservers.

- **models.py** – модель приложения. Здесь описываются классы, по которым после выполнения определенной команды создаются миграции.
- **migrations** – внутрь этой папки помещаются автоматически созданные миграции – скрипты, которые способны самостоятельно записать в базу данных все необходимые описанные ранее таблицы. Этот инструмент позволяет разработчику минимизировать работу непосредственно с базой данных.
- **app.py** – здесь описан класс приложения.
- **urls.py** – контроллер приложения.
- **admin.py** – классы для администрирования сайта. В Django администрирование максимально автоматизировано, его не нужно писать с нуля. Это позволяет, опять-таки, свести к минимуму работу с БД.
- папки **templates** и **static** – аналогичны тем, что хранятся в главном пакете.

- **views.py** – в этом модуле описаны отображения приложения. Каждому отображению передается управление из контроллера приложения и в нем ведется работа по обработке запроса пользователя.

Аналогично был создан пакет Users.

```
Users
|   admin.py
|   apps.py
|   models.py
|   tests.py
|   urls.py
|   views.py
|   __init__.py
|
+---migrations
|
+---static
|
\---templates
```

Структура приложения Users.

Работа над приложением Licenseservers

Сначала будет описана работа с приложением Licenseservers. Приложение должно обрабатывать все запросы, связанные непосредственно с таблицами серверов лицензий и таблиц лицензий.

В модуле views.py был описан класс ViewLicenses, который обрабатывает запрос, связанный с просмотром лицензий на всех сайтах. Плюс, необходимо было реализовать фильтрацию данных по сайту.

При передаче управления классу, соответствующий метод проверяет поступивший get-запрос. Если в нем нет параметров или есть параметр all, то метод возвращает все имеющиеся в таблице данные о серверах лицензий, если же передан какой-то параметр (название сайта), то метод возвращает только соответствующие строки из таблицы. Передача параметров в get-запрос осуществляется с помощью выбора одного из пунктов списка на странице.

```
class ViewLicenses(ListView):
    model = Licenseservers

    def get_context_data(self, *, object_list=None, **kwargs):
        context = super().get_context_data(**kwargs)
        context['sites'] = Licenseservers.objects.values_list('site',
flat=True).distinct()
        context['select_site'] = self.request.GET['select_menu'] if
len(self.request.GET) else 'all'
        return context

    def get_queryset(self):
        req = self.request.GET
        if len(req) == 0 or req['select_menu'] == 'all':
            return Licenseservers.objects.all()
        else:
            return Licenseservers.objects.filter(site=req['select_menu'])
```

Класс ViewLicenses.

Там же был описан класс ViewFreeLicenses, связанный с запросом предоставления свободных лицензий с возможностью фильтрации по сайтам и по типу лицензий, с соответствующим приоритетом фильтров.

Метод возвращает данные для заполнения полей фильтров, и запрошенные данные: все, либо фильтрованные по указанным параметрам. Получение данных осуществляется с помощью сложного SQL-запроса, описанного в методе get_request.

```
class ViewFreeLicenses(ListView):
    model = Licenseservers
    template_name = 'freelicenseservers_list.html'

    def get_context_data(self, *, object_list=None, **kwargs):
        context = super().get_context_data(**kwargs)
        context['sites'] = Licenseservers.objects.values_list('site',
flat=True).distinct()
        context['lic_names'] = dict()
        context['lic_names']['all'] = Licenseservers.objects.values_list('name',
flat=True).distinct()
        for site in context['sites']:
            context['lic_names'][f'{site}'] =
Licenseservers.objects.filter(site=site).values_list('name',
flat=True).distinct()

        context['select_site'] = self.request.GET['select_menu1'] \
            if 'select_menu1' in self.request.GET else 'all'
        context['select_lic_name'] = self.request.GET['select_menu2'] \
            if 'select_menu2' in self.request.GET else 'all'
        return context

    def get_queryset(self):
        req = self.request.GET
        sql_request = '''
            SELECT
                lic.id,
                lic.site,
```

```
                lic.name,
                lic.host,
                lic.total_auth,
                us1.count_auth,
                lic.total_auth - us1.count_auth as difference_author,
                lic.total_cons,
                us2.count_cons,
                lic.total_cons - us2.count_cons as difference_consumer

                FROM tcusers.licenseservers lic
                left join (select
                        site,
                        licenseServer,
                        licenseLevel,
                        count(*) as count_auth
                            from tcusers.users as us where us.status = 0 and
us.licenseLevel = 0
                            group by us.licenseLevel, us.site, us.licenseServer)
as us1
                on lic.name=us1.licenseServer and lic.site = us1.site
                left join (select
                        site,
                        licenseServer,
                        licenseLevel,
                        count(*) as count_cons
                            from tcusers.users as us where us.status = 0 and
us.licenseLevel = 1
                            group by us.licenseLevel, us.site, us.licenseServer)
as us2
                on lic.name=us2.licenseServer and lic.site = us2.site
            '''
        # если это не первый заход на страницу, то в запросе будут хоть какие то
аргументы
        if len(req) != 0:
            # проверка выбраны ли фильтры по сайту и не all ли это
            site_filter = 'select_menu1' in self.request.GET \
                          and self.request.GET['select_menu1'] != 'all'
            # проверка выбраны ли фильтры по лицензии и не all ли это
            name_lic_filter = 'select_menu2' in self.request.GET \
                              and self.request.GET['select_menu2'] != 'all'

            # если оба фильтра выбраны all то нет смысла фильтровать данные,
достаточно выполнить исходный запрос
            if site_filter or name_lic_filter:
                sql_request += '\nWHERE '
                params = []

                if site_filter:
                    sql_request += 'lic.site=%s'
                    params.append(self.request.GET['select_menu1'])
                    if name_lic_filter:
                        sql_request += ' and '
                if name_lic_filter:
                    sql_request += 'lic.name=%s'
                    params.append(self.request.GET['select_menu2'])

                data = Licenseservers.objects.raw(sql_request, params=params)
                return data

        data = Licenseservers.objects.raw(sql_request)
        return data
```

Класс ViewFreeLicenses.

Фильтрация реализована на js. Функция set_lic_list срабатывает по клику на одно из значений выпадающего списка. Работа функции описана в комментариях в приведенном коде. Данный файл находится в TUPOLEV_APP/Licenseservers/static/js/.

```
function set_lic_list() {
    // получаем выбранное значение из списка сайтов
    let elem1 = document.getElementById('select_menu1')
    let select_site_value = elem1.options[elem1.selectedIndex].value

    /*
    заменяем список, создающийся сервером, на список, обрабатываемый js

    сервер создал выпадающие списки для каждого из выбранных сайтов(см html-
шаблон), но все они кроме all изначально скрыты
    при выборе значения сайта мы прячем все списки, а потом показываем только
нужный
    атрибут display отвечает за отображение на странице
    атрибут disabled блокирует элемент, чтобы данные из него не обрабатывались
    */
    let elems = document.getElementsByName('select_menu2')
    for (let i = 0; i < elems.length; i++) {
        elems[i].style.display = 'none'
        elems[i].setAttribute("disabled", "disabled");
    }
    let elem2 = document.getElementById(`select_menu2_for_${select_site_value}`)
    elem2.style.display = 'block'
    elem2.removeAttribute('disabled')
}
```

Функция set_lic_list, осуществляющая фильтрацию данных.

Работа над приложением Users

Необходимо реализовать показ данных, в которых можно будет видеть соответствие пользователей и групп, к которым они принадлежат. Должна быть предусмотрена фильтрация либо по ФИО пользователя, либо по его логину.

Для этого был написан класс ViewGroupMembers. Его работа в целом аналогична работе предыдущих описанных классов, поэтому будет приведен только код с комментариями.

```
class ViewGroupmembers(ListView):
    model = Users

    def get_context_data(self, *args, object_list=None, **kwargs):
        # кладем все нужное в контекстную переменную
        context = super().get_context_data(**kwargs)

        context['names'] = Users.objects.values_list('fullname',
```

```
flat=True).order_by('fullname').distinct()
        context['tcnames'] = Users.objects.values_list('tcname',
flat=True).order_by('tcname').distinct()
        context['table_print'] = len(self.request.GET) != 0

        return context

    def get_queryset(self):
        req = self.request.GET
        sql_request = '''
            SELECT
            us.id,
            us.fullName as fullname,
            us.tcName as tcname,
            us.osName as osname,
            gm.group,
            gm.role,
            us.lastLogin as lastlogin,
            us.licenseServer as licserver,
            us.site as site,
            us.status

            FROM tcusers.users us right join tcusers.groupmembers gm on us.tcName
= gm.tcName and us.site = gm.site
        '''

        if req:
            sql_request += '\nWHERE '
            params = []
            if req['fullname_choice'] != '':
                sql_request += 'fullname=%s'
                params.append(self.request.GET['fullname_choice'])
                if req['tcname_choice'] != '':
                    sql_request += ' and '

            if req['tcname_choice'] != '':
                sql_request += 'us.tcName=%s'
                params.append(self.request.GET['tcname_choice'])

            data = Users.objects.raw(sql_request, params=params)
            return data

        data = Users.objects.raw(sql_request)
        return data
```

Функция set_lic_list, осуществляющая фильтрацию данных.

Для полей фильтров реализована система быстрого ввода с помощью стандартных средств html, а именно тега datalist. Его работа такова: например, пользователь начинает вводить какое-то ФИО, и ему предлагаются варианты, соответствующие уже написанной части строки, полученные из БД. Алгоритм работы аналогичен: если никаких дополнительных данных при get-запросе не поступило, будут возвращены все данные, которые в последствии отобразятся пользователю. Иначе, будут отображены фильтрованные данные.

Последний из реализованных классов ViewAddingOrders описывает процесс обработки поступающих запросов о создании и выдаче лицензий. Запрос поступает пользователю либо в текстовом виде, либо в виде exel-таблицы. В первом случае пользователю удобнее было бы иметь на странице форму, в которую можно было бы быстро занести все необходимые данные.

Если класс получает на вход форму, он обрабатывает данные в ней, валидирует каждую строку, а затем записываю всю информацию в файл и в специальную таблицу логов в базе данных. Если на вход не пришло никаких параметров, отображается пустая таблица-форма с одной строкой.

```
class ViewAddingOrders(TemplateView):
    template_name = 'adding_orders.html'

    def get(self, request, *args, **kwargs):
        req = self.request.GET
        if req:  # эта часть кода сработает, когда мы отправим данные из формы
            # достаем данные
            form_set = TableFormSet0(req).data
            table = defaultdict(list)
            for inf in form_set:
                key = inf.split('-')[1]
                table[key].append(form_set[inf])

            # валидируем каждую строку
            for key in table:
                self.validate_data(table[key])

            # запись в файл
            with open(FILE_FOR_PROC_REQUESTS, 'w', encoding='ansi') as f:
                for key in table:
                    f.write('|'.join(table[key]))
                    f.write('\n')

            # запись в логи
            new_info = ''
            for key in table:
                new_info += '|'.join(table[key][1:]) + '\n'

            Logs(info=new_info,
                 sd_number=table[key][0],
                 session_id=os.getlogin(),
                 action='something'
                 ).save()

            return redirect('menu')
        # эта часть кода сработает чтобы отобразить таблицу в начальном состоянии
        return render(request, self.template_name, self.get_context_data())

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        # помещаем данные из сессии в контекстную переменную
        if self.request.session.get('table_data') != None:
            context['form_set'] =
```

```
TableFormSet0(initial=self.request.session.get('table_data'))
        else:
            context['form_set'] = TableFormSet1()

        try:  # чистим сессию
            del self.request.session['table_data']
        except KeyError:
            pass

        # вставка полей для подсказок
        context['sites'] = Licenseservers.objects.values_list('site',
flat=True).distinct()
        context['lic_servers'] = Licenseservers.objects.values_list('name',
flat=True).distinct()

        return context

    def validate_data(self, data: list):  # функция для валидации строки таблицы
        def valid_req_name(dat):  # валидация номера заявки
            return False

        def valid_full_name(dat):  # валидация фио
            return False

        if valid_req_name(data[1]):
            raise ValidationError('текст ошибки', code='код ошибки')

        if valid_full_name(data[0]):
            raise ValidationError('текст ошибки', code='код ошибки')
```

Класс ViewAddingOrders.

При необходимости, в нее можно добавлять еще строки (процесс реализован с помощью js, функция add_line_func). К тому же, в html добавлена возможность предзаполнения для полей, которые чаще всего имеют одно и то же значение.

```
// добавление новой доп строки в таблицу
function add_line_func() {
    let table = document.getElementsByName('table_line');
    let last_line = table[table.length - 1];

    let new_table_line = `<tr name="table_line" id="${Number(last_line.id) + 1}">
                            <td>
                                <input type="text" name="form-
${Number(last_line.id) + 1}-req_num" class="form-control" id="request_number"
autocomplete="off">
                            </td>
                            <td>
                                <input type="text" name="form-
${Number(last_line.id) + 1}-full_name" class="form-control" id="full_name"
onchange="os_name_auto_fill(this)" autocomplete="off">
                            </td>
                            <td>
                                <input type="text" name="form-
${Number(last_line.id) + 1}-os_name" class="form-control" id="os_name"
autocomplete="off">
                            </td>
                            <td>
                                <input type="text" name="form-
${Number(last_line.id) + 1}-tc_name" class="form-control" id="tc_name"
```

```
placeholder="авт." autocomplete="off">
                            </td>
                            <td>
                                <input type="text" name="form-
${Number(last_line.id) + 1}-tc_pass" class="form-control" id="tc_pass"
autocomplete="off" placeholder="авт.">
                            </td>
                            <td>
                                <input type="text" name="form-
${Number(last_line.id) + 1}-group" class="form-control" id="group"
autocomplete="off">
                            </td>
                            <td>
                                <input type="text" name="form-
${Number(last_line.id) + 1}-role" class="form-control" id="role"
autocomplete="off">
                            </td>
                            <td>
                                <input type="text" name="form-
${Number(last_line.id) + 1}-lic_server" class="form-control" id="lic_server"
autocomplete="off" placeholder="начните ввод..." list="lic_list">
                            </td>
                            <td>
                                <input type="text" name="form-
${Number(last_line.id) + 1}-site" class="form-control" id="site"
autocomplete="off" placeholder="начните ввод..." list="site_list">
                            </td>
                        </tr>
                        `


    last_line.insertAdjacentHTML("afterend", new_table_line);
}
```

Функция add_line_func, осуществляющая добавление строки в форму-таблицу.

Также, некоторые поля, например, логин, можно сгенерировать из ФИО по заданным правилам. Этот функционал реализован на js в функции os_name_auto_fill, которая также вызывает функцию toTranslit, переводящую ФИО на кириллице в транслит на латинице. Функция os_name_auto_fill срабатывает при заполнении поля ФИО в строке таблицы. Далее пользователь сможет исправить автоматически сгенерированные поля, если это требуется.

```
function toTranslit(text) {
    return text.replace(/([а-яё])|([\s_-])|([^a-z\d])/gi,
        function (all, ch, space, words, i) {
            if (space || words) {
                return space ? ' ' : '';
            }
            var code = ch.charCodeAt(0),
                index = code == 1025 || code == 1105 ? 0 :
                    code > 1071 ? code - 1071 : code - 1039,
                t = ['yo', 'a', 'b', 'v', 'g', 'd', 'e', 'zh',
                    'z', 'i', 'y', 'k', 'l', 'm', 'n', 'o', 'p',
                    'r', 's', 't', 'u', 'f', 'h', 'c', 'ch', 'sh',
                    'shch', '', 'y', '', 'e', 'yu', 'ya'
                ];
```

```
            return t[index];
        });
}

function os_name_auto_fill(obj) {
    // получение ФИО из поля ввода
    let word = obj.value.split(' ')
    let id = obj.parentElement.parentElement.id

    // генерация логина для tc
    let new_word_tc = ''
    new_word_tc += toTranslit(word[1].charAt(0))
    new_word_tc += toTranslit(word[0])
    let field = document.getElementsByName(`form-${id}-tc_name`)
    document.getElementsByName(`form-${id}-tc_name`)[0].value = new_word_tc
    // генерация пароля для tc
    new_word_tc += Math.floor(Math.random() * 999)
    document.getElementsByName(`form-${id}-tc_pass`)[0].value = new_word_tc
}
```

Функция os_name_auto_fill, осуществляющая автоматическое предзаполнение некоторых полей.

Если же данные заявки поступили в виде таблицы, соответствующей строго заданному формату, то добавлена возможность загрузки xls файла. Тогда срабатывает функция upload_excel, которая обрабатывает файл и передает управление и обработанные данные из файла классу ViewAddingOrders, описанному выше.

Обработка переданного производится с помощью функции handle_upload_file. В ней из файла достаются данные и помещаются в структуру данных языка Python – словарь. Также эта функция несколько раз вызывает функцию transliterate, которая работает аналогично приведенной ранее функции toTranslite. Разница в работе этих функцию только в том, что одна работает на сервере и обрабатывает данные из файла, а другая на клиенте и обрабатывает данные формы. С помощью этих функций мы получаем одинаковый функционал для пользователя в обоих сценариях: и при заполнении формы вручную, и при помощи загрузки файла, некоторые поля таблицы будут определенным образом предзаполнены.

```
def upload_excel(request):
    if request.method == 'POST' and request.FILES['file']:
        # обработка данных полученных из формы
        file = request.FILES['file']
        table_data = handle_uploaded_file(file)
```

```
        # поместим полученные данные из файла в сессию
        request.session['table_data'] = table_data

        return redirect('adding_orders')  # будет вызван метод get_context_data
класса ViewAddingOrders
    else:
        # вывод самой формы при первом заходе на страницу
        form = UploadExcelFileForm()
    return render(request, 'adding_excel_file.html', {'form': form})


def handle_uploaded_file(f):
    try:
        ex_file = openpyxl.load_workbook(f)
    except:
        raise ValidationError('Invalid format', code='invalid')

    sheet = ex_file.get_sheet_by_name(ex_file.get_sheet_names()[0])

    table_data = []

    req_num = sheet['A2'].value
    for row in range(2, sheet.max_row + 1):
        trans_name = transliterate(sheet[(f"B{row}")].value.split(' ')[1][:1]) \
                     + transliterate(sheet[(f"B{row}")].value.split(' ')[0])
        trans_name = trans_name.lower()

        table_data.append({
            'req_num': f'{req_num}',
            'full_name': f'{sheet[(f"B{row}")].value}',
            'os_name': f'{sheet[(f"F{row}")].value}',
            'tc_name': f'{trans_name}',
            'tc_pass': f'{trans_name + str(randint(111, 999))}',
            'group': f'{sheet[(f"D{row}")].value}',
            'role': f'{sheet[(f"E{row}")].value}',
            'lic_server': f'{sheet[(f"G{row}")].value}',
            'site': f'{sheet[(f"H{row}")].value}',
        })

    return table_data


def transliterate(name):
    # Слоаврь с заменами
    slovar = {
        'а': 'a', 'б': 'b', 'в': 'v', 'г': 'g', 'д': 'd', 'е': 'e', 'ё': 'yo',
        'ж': 'zh', 'з': 'z', 'и': 'i', 'й': 'y', 'к': 'k', 'л': 'l', 'м': 'm',
'н': 'n',
        'о': 'o', 'п': 'p', 'р': 'r', 'с': 's', 'т': 't', 'у': 'u', 'ф': 'f',
'х': 'h',
        'ц': 'c', 'ч': 'ch', 'ш': 'sh', 'щ': 'shch', 'ъ': '', 'ы': 'y', 'ь': '',
'э': 'e',
        'ю': 'yu', 'я': 'ya',

        'А': 'A', 'Б': 'B', 'В': 'V', 'Г': 'G', 'Д': 'D', 'Е': 'E', 'Ё': 'Yo',
        'Ж': 'Zh', 'З': 'Z', 'И': 'I', 'Й': 'Y', 'К': 'K', 'Л': 'L', 'М': 'M',
'Н': 'N',
        'О': 'O', 'П': 'P', 'Р': 'R', 'С': 'S', 'Т': 'T', 'У': 'U', 'Ф': 'F',
'Х': 'H',
        'Ц': 'C', 'Ч': 'Ch', 'Ш': 'Sh', 'Щ': 'Shch', 'Ъ': '', 'Ы': 'Y', 'Ь': '',
'Э': 'E',
        'Ю': 'Yu', 'Я': 'Ya',

        ' ': ' ',
    }

    # Циклически заменяем все буквы в строке
    for key in slovar:
```

```
        name = name.replace(key, slovar[key])
    return name
```

Функции upload_excel и handle_upload_file, осуществляющие загрузку и обработку данных из excel файла.

Класс ViewAddingOrders выводит всё ту же форму, заполненную извлеченными данными. Пользователь может проверить, что все было обработано верно, отредактировать что-то при необходимости, а затем отправить данные формы далее. При этом будет создана запись в таблице логов, которая будет содержать всю необходимую информацию о заявке.

Заключение

В процессе прохождения практики был получен опыт работы с реальными задачами под руководством опытных наставников, удалось принципы и методы работы и проектирования веб-приложений.

За время практики было разработано веб-приложение, состоящее из 1034 строк на Python и 125 строк на JavaScript, реализующее удобный интерфейс для удобной работы с существующей базой данных. В процессе удалось закрепить и углубить знания веб-фреймворка Django, познакомиться с языком JavaScript. Получить практический опыт разработки коммерческого приложения и его архитектуры по требованиям заказчика, в роли которого выступал руководитель, также контролирующий и управляющий процессом разработки.