

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»  
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных  
систем

## **Курсовая работа**

по дисциплине: Базы данных

тема: «Web-приложение и База Данных Туристического Агентства»

Выполнил: ст. группы ПВ-202  
Андреев Илья Романович

Проверил:  
ст. преп. Панченко М.В.

Белгород 2022 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»  
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных  
систем

## **Курсовая работа**

по дисциплине: Базы данных

тема: «Web-приложение и База Данных Туристического Агентства»

Выполнил: ст. группы ПВ-202  
Андреев Илья Романович

Проверил:  
ст. преп. Панченко М.В.

Белгород 2023 г.

## **Курсовая работа** **«Web-приложение и База Данных Туристического Агентства»**

**Цель работы:** получение навыков разработки приложений для взаимодействия с базой данных, содержащих графический интерфейс пользователя, навыков разделения ролей пользователей, навыков разработки механизмов аутентификации, навыков выгрузки данных из БД в различных форматах, навыков автоматизации создания резервных копий БД, и работы с ними.

### **Задание:**

Разработать Web-приложение и Базу Данных для Туристического Агентства.

## Оглавление

Введение .....	5
Теоретические сведения.....	6
Проектирование (Диаграмма сущность-связь).....	11
Анализ предметной области.....	11
Заполнение базы данных .....	12
Работа с файлами каталога. Подключение к БД .....	14
Описание приложения .....	16
Models .....	16
View .....	17
Paginator .....	17
URLs.....	20
Templates.....	21
Разделение полей пользователей (механизмы аутентификации):.....	25
Выгрузка данных из БД.....	30
Создание резервной копии БД.....	34
Результаты работы приложения.....	37
Заключение.....	39
Список литературы .....	39

## **Введение**

Целью данной курсовой является изучить основы языка программирования Django и его применение в области веб-разработки и работы с БД с применением Django ORM, а также разработать веб-сайт “Школы программирования” для работы с базой данных. Предусмотреть следующий функционал:

- Разделение ролей пользователей, разработка механизмов аутентификации;
- Выгрузка данных из БД по сформированным запросам на выборку в два из предложенных форматов: json, csv;
- Автоматизированное создание резервной копии базы данных и размещение ее на удаленном компьютере (хранение в облаке).

## **Цель и назначение разработки:**

Целью разработки является создание информационной системы школы программирования, автоматизирующей процессы предоставления актуального расписания занятий, списка преподавателей и модулей, записи на занятие и покупки модуля.

## **Используемые технологии:**

- Django
- PostgreSQL
- Git

## **Возможности web-приложения:**

- Добавление новых клиентов
- Просмотр всех клиентов
- Просмотр всех туров
- Добавление новых туров
- Добавление нового персонала
- Получение отчётов по контрактам/клиентам

## Теоретические сведения

### Краткие сведения о Django:

*Django* — свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC (Model View Controller).

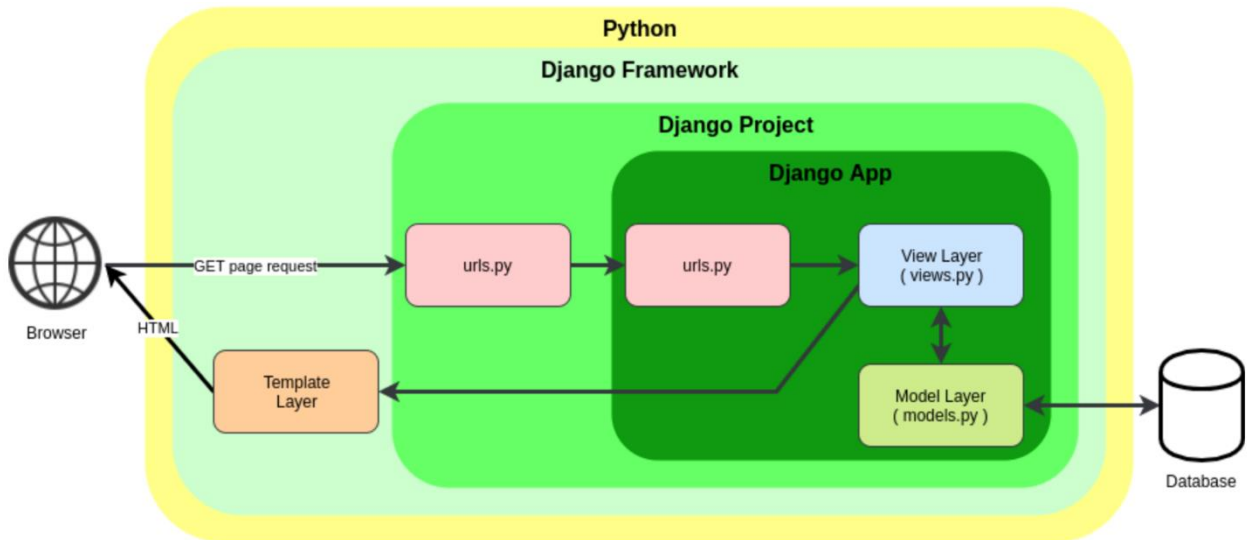
- Модель — бизнес-логика, то есть совокупности методов, правил и ограничений работы с данными.
- Представление — компонент, отображающий пользователю данные в зависимости от изменения модели.
- Контроллер — программный посредник, обрабатывающий действия пользователя и сообщающий модели, как она должна измениться.

Фреймворк Django реализует архитектурный паттерн Model-View-Template или сокращенно MVT, который по факту является модификацией паттерна MVC (Model-View-Controller).

### Основные элементы паттерна MVT:

- URLs: Гораздо удобнее писать отдельную функцию для обработки каждого ресурса, чем обрабатывать запросы с каждого URL-адреса с помощью одной функции. URL-маршрутизатор используется для перенаправления HTTP-запросов в соответствующее представление на основе URL-адреса запроса. Кроме того, URL-маршрутизатор может извлекать данные из URL-адреса в соответствии с заданным шаблоном и передавать их в соответствующую функцию отображения (view) в виде аргументов.
- View: View (англ. «отображение») — это функция обработчика запросов, которая получает HTTP-запросы и возвращает ответы. Функция view имеет доступ к данным, необходимым для удовлетворения запросов, и делегирует ответы в шаблоны через модели.
- Models: Модели представляют собой объекты Python, которые определяют структуру данных приложения и предоставляют механизмы для управления (добавления, изменения, удаления) и выполнения запросов в базу данных.
- Templates: Template (англ. «шаблон») — это текстовый файл, определяющий структуру или разметку страницы (например HTML-страницы), с полями для подстановки, которые используются для вывода актуального содержимого. View может динамически создавать HTML-страницы, используя HTML-шаблоны и заполняя их данными из модели (model). Шаблон может быть использован для определения структуры файлов любых типов, не обязательно HTML.

## Схема работы приложения Django:



## Models в ORM Django:

Django ORM (Object Relational Mapping) является одной из самых мощных особенностей Django. Это позволяет взаимодействовать с базой данных, используя код Python, а не SQL.

Модели отображают информацию о данных, с которыми можно работать. Они содержат поля и поведение ваших данных. Обычно одна модель представляет одну таблицу в базе данных.

- Каждая модель это класс унаследованный от `django.db.models.Model`.
- Атрибут модели представляет поле в базе данных.
- Django предоставляет автоматически созданное API для доступа к данным.

## Миграции:

Django использует миграции для переноса изменений в моделях на структуру базы данных. Для этого предоставляются две основные команды для работы с миграциями и структурой базы данных:

- `migrate`, которая отвечает за применение миграций, за откат миграций и за вывод статуса миграций.
- `makemigrations`, которая отвечает за создание новых миграций на основе изменений в моделях.

Следует рассматривать миграции, как систему контроля версий для базы данных. `makemigrations` отвечает за сохранение состояния моделей в файле миграции - аналог коммита - а `migrate` отвечает за их применение к базе данных.

Файлы с миграциями находятся в каталоге “migrations” приложения. Они являются частью приложения и должны распространяться вместе с остальным кодом приложения

### **QuerySet:**

Для получения объектов из базы данных, создается QuerySet через Manager модели. QuerySet представляет выборку объектов из базы данных. Он может не содержать, или содержать один или несколько фильтров – критерии для ограничения выборки по определенным параметрам. В терминах SQL, QuerySet - это оператор SELECT, а фильтры - условия такие, как WHERE или LIMIT.

Каждая модель содержит как минимум один Manager, и он называется objects по умолчанию. Обратиться к нему можно непосредственно через класс модели:

Manager - главный источник QuerySet для модели. Например, \*.objects.all() вернет QuerySet, который содержит все объекты \* из базы данных.

### **Jinja и DTL:**

Будучи веб фреймверком, Django позволяет динамически генерировать HTML. Самый распространенный подход - использование шаблонов. Шаблоны содержат статический HTML и динамические данные, рендеринг которых описан специальным синтаксисом.

Проект Django может использовать один или несколько механизмов создания шаблонов. Django предоставляет бэкенд для собственной системы шаблонов, которая называется - язык шаблонов Django (Django template language, DTL), и популярного альтернативного шаблонизатора Jinja2.

**Переменные** выводят значения из контекста, который является словарем. Переменные выделяются {{ }}. Обращение к ключам словаря, атрибутам объектов и элементам списка выполняется через точку.

Если значением переменной является вызываемый объект, шаблонизатор вызовет его без аргументов и подставит результат.

**Теги** позволяют добавлять произвольную логику в шаблон. Например, теги могут выводить текст, добавлять логические операторы, такие как “if” или “for”, получать содержимое из базы данных, или предоставлять доступ к другим тегам. Теги выделяются {% %}. Большинство тегов принимают аргументы, некоторые теги требуют закрывающий тег.



## Представления Views и класс Class Based Views:

Представление, в самом общем виде, - это исполняемый (callable) объект, который принимает “на вход” запрос (request), и возвращает ответ (response). В этой роли может выступать не только функция, но и классы. Такие классы позволяют создавать структурированные, повторно используемые представления, базируясь на возможностях наследования и, в том числе, множественного наследования, - с использованием примесей(mixins). Django включает в себя набор общих(generic) представлений, которые идеально подходят для решения ряда рутинных задач.

Django предлагает нам базовый набор классов представлений, который может использоваться в широком спектре приложений. Все классы-представления наследуют класс View, который обрабатывает “привязку” представления с соответствующими URL, диспетчеризацию HTTP запросов (анализ методов GET, POST и последующий вызов одноименного метода для обработки запроса) и ряд других простых действий. Класс RedirectView служит для простого HTTP перенаправления (redirect), а класс TemplateView расширяет базовый класс, предоставляя возможность обработки шаблонов и т.д.

## Forms:

Форма в HTML – это набор элементов в `<form>...</form>`, которые позволяют пользователю вводить текст, выбирать опции, изменять объекты и контролы страницы, и так далее, а потом отправлять эту информацию на сервер.

Кроме `<input>` элементов форма должна содержать еще две вещи:

- куда: URL, на который будут отправлены данные
- как: HTTP метод, который должна использовать форма для отправки данных

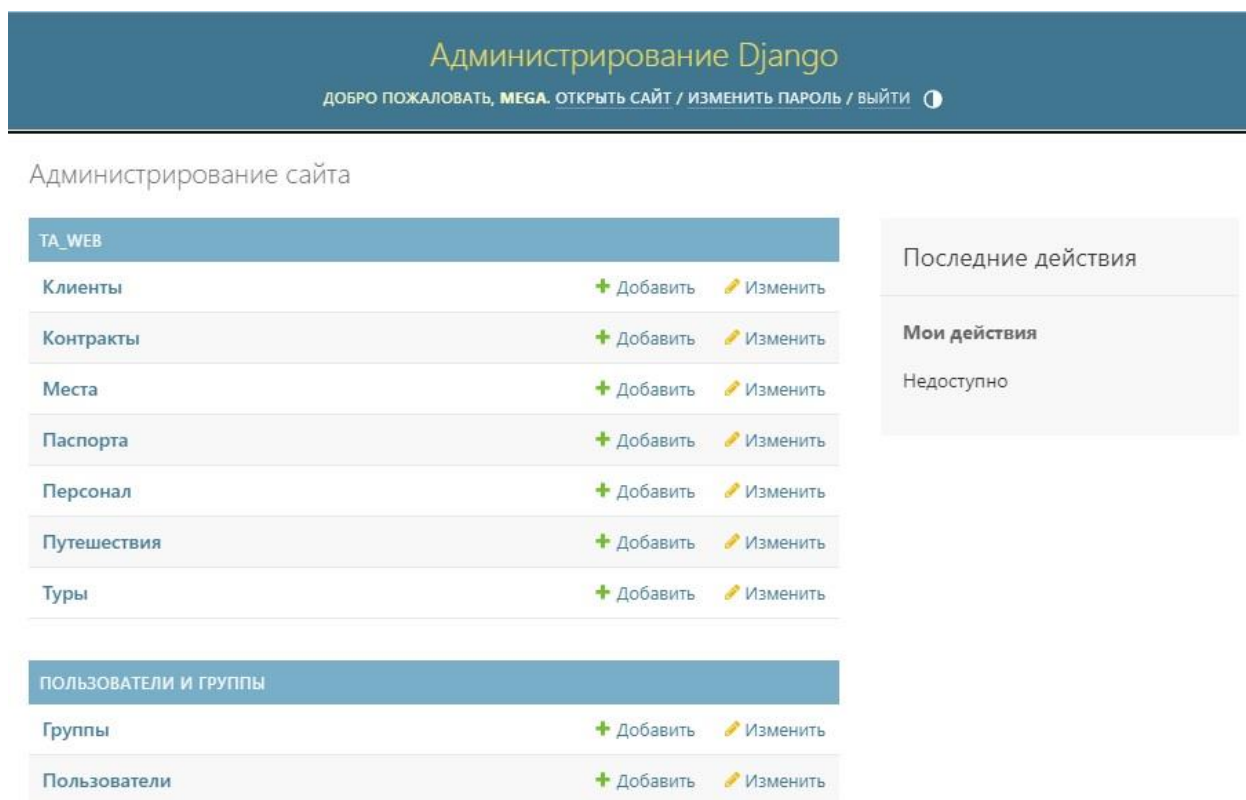
Сердце всего механизма – класс Form. Как и модель в Django, которая описывает структуру объекта, его поведение и представление, Form описывает форму, как она работает и показывается пользователю. Как поля модели представляют поля в базе данных, поля формы представляют HTML `<input>` элементы. (ModelForm отображает поля модели в виде HTML `<input>` элементов, используя Form. Используется в админке Django.)

Поля формы сами являются классами. Они управляют данными формы и выполняют их проверку при отправке формы. Например, DateField и FileField работают с разными данными и выполняют разные действия с ними.

Поле формы представлено в браузере HTML “виджетом” - компонент интерфейса. Каждый тип поля представлен по умолчанию определенным классом Widget, который можно переопределить при необходимости.

### Admin панель:

Одна из сильных сторон Django – это автоматический интерфейс администратора. Он использует мета-данные модели чтобы предоставить многофункциональный, готовый к использованию интерфейс для работы с содержимым сайта.



### Django-filters:

Django-filter предоставляет простой способ отфильтровать набор запросов на основе параметров, предоставляемых пользователем. Допустим, у нас есть модель Client, и мы хотим позволить пользователю фильтровать, каких клиентов они видят на странице списка. Здесь используется очень похожий API на ModelForm от Django. Как и в случае с ModelForm, мы также можем переопределять фильтры или добавлять новые, используя декларативный синтаксис.

## Проектирование (Диаграмма сущность-связь)

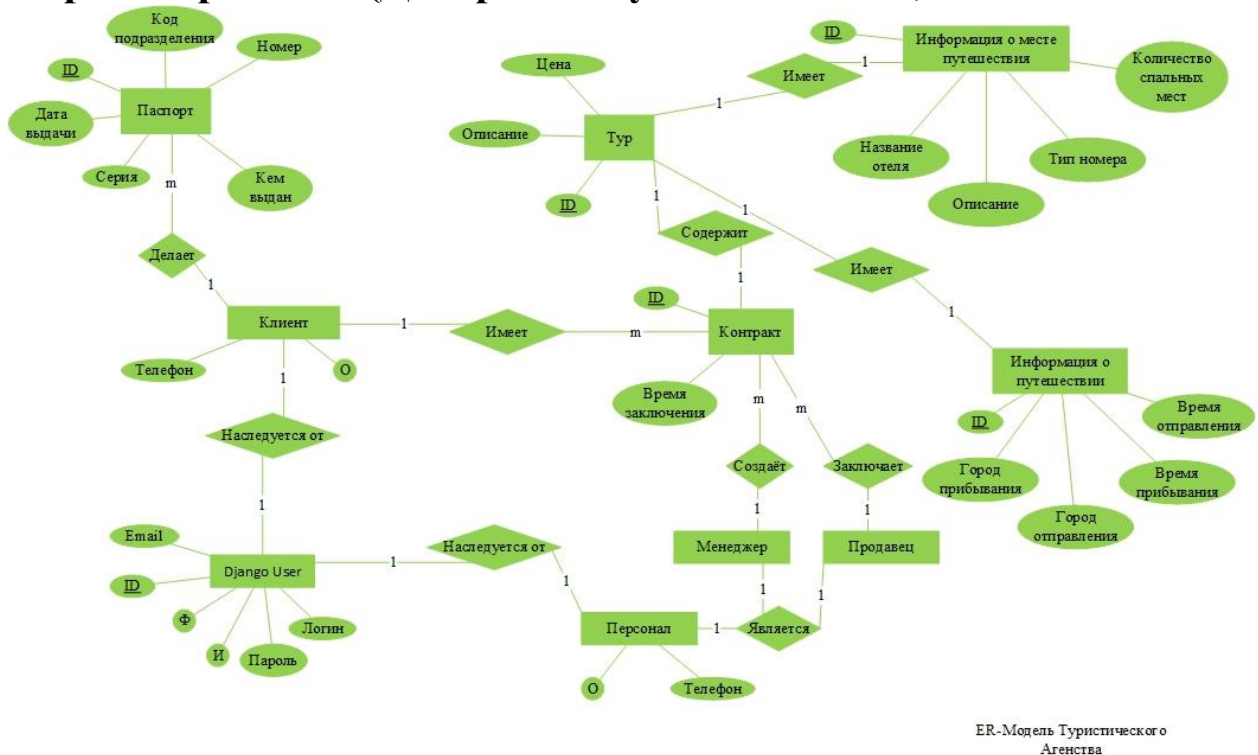


Рис. 1. Диаграмма «сущность-связь» предметной области «Туристическое агентство»

## Анализ предметной области

Выделим следующие связи между сущностями, которые будут храниться в базе данных:

**Клиент** заключает договор с продавцом. В договоре содержится информация о **туре**, путешествии и месте отдыха. **Менеджер** может добавлять, как новые **туры**, в которые входит информация о **путешествии** и **месте отдыха**, так и **продавцов**. Клиент может просмотреть информацию о себе в своём личном кабинете.

Для заполнения базы данных и генерации групп были написаны скрипты на языке Python:

```
from django.core.management import BaseCommand
from ...models import *
```

```
from django.core.management import BaseCommand
from django.contrib.auth.models import Group
from ...models import *
```

```
class Command(BaseCommand):
    def handle(self, *args, **options):
        group = Group.objects.get(name='Client')
        for i in range(15, 30):
            new_passport = Passport.objects.create(
                series='7777',
                number=f'00000{i - 13}',
                code='123456',
                issue_date=f'2011-10-{i + 1}',
                giver='Каким-то отделением какого-то органа по какой-то
области где-то'
            )
            new_user = Client.objects.create(username=f'Test{i - 13}',
                password='test_test',
                email=f'test{i - 13}@mail.ru',
                first_name=f'Тест{i - 13}',
                last_name=f'Тестовый{i - 13}',
                patronymic=f'Тестович{i - 13}',
                phone=f'+79000000000{i - 13}',
```

```

        birthday=f'1991-10-{i + 1}',
        passport=new_passport,
        is_superuser=False,
        is_staff=False,
    )

    group.user_set.add(new_user)

```

Создание Типов:

```

from django.core.management import BaseCommand
from ...models import *

```

```

class Command(BaseCommand):
    def handle(self, *args, **options):
        for i in range(2, 8):
            new_place = Place.objects.create(
                count=i,
                name=TYPES[i],
                hotel=f'Какой-то классный отель №{i}',
                description=f'Описание какого-то классного тура №{i}',
            )

            new_movement = Movement.objects.create(
                d_city=f'Какой-то город №{i}',
                d_time=f'2023-07-0{i} 0{i}:0{i}:00+03',
                a_city=f'Другой город №{i}',
                a_time=f'2023-07-0{i + 1} 0{i + 1}:0{i + 1}:00+03',
            )

            new_tour = Tour.objects.create(
                place=new_place,
                nights=f'{i}',
                cost=f'{i}0000.0{i}',
                movement=new_movement,
                description=f'Описание какого-то классного тура №{i}',
            )

```

## Работа с файлами каталога. Подключение к БД

Каталог файлов Django генерирует самостоятельно при создании проекта. Что удобно. К стандартным файлам настройки можно будет в дальнейшем добавить нужные для разработки файлы, которые будут отделять разные части приложения друг от друга.

По умолчанию Django создает в основной папке приложения файлы настройки, среди которых есть:

- settings.py для настройки сайта (в папке сайта)
- urls.py содержит список шаблонов, по которым ориентируется urlresolver (в папке сайта)
- manage.py, который находится в основной папке и служит для запуска приложения и управлением сайта.

Основное, что надо сделать, так это в файле settings.py подключиться к БД. По умолчанию Django использует СУБД SQLite, но ее функционал ограничен. Так как при разработке приложения первое, что было выполнено – создана база данных и описаны все таблицы в СУБД PostgreSQL, то было решено работать с уже подготовленной БД.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'TourAgency',
        'USER': 'postgres',
        'PASSWORD': '',
        'HOST': 'localhost',
    },
}
```

Кроме этого, в настройках указываем информацию о статических файлах:

```
MEDIA_ROOT = BASE_DIR / 'static/ta_web/media'

STATICFILES_DIRS = [
    BASE_DIR / 'static/'
]
```

Непосредственно работа по запуску сайта, по созданию миграций и т.п. происходит с помощью команд, в которых присутствует название файла manage.py. Чтобы запустить веб-сервер необходима команда:

```
python manage.py runserver
```

Либо можно настроить точку входа при работе с PyCharm.

Создание приложения Django происходит с помощью команды:

```
python manage.py startapp <название>
```

После этого в файл `settings.py` необходимо подключить наше приложение в список установленных приложений:

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'ta_web.apps.TaWebConfig'  
]
```

В новой папке появятся и файлы для управления приложением, среди которых будут:

- `admin.py`, с помощью которого осуществляется администрирование сайта (управление панелью управления администратора Django)
- `migrations.py`, в который автоматически сохраняются все миграции приложения
- `models.py`, в котором будем создавать все модели для работы с базой данных
- `views.py`, в котором создаем все представления для отображения информации на сайте

## Описание приложения

Пользователем данного приложения являются работники Туристического агентства и его возможные клиенты. В связи с этим, был разработан следующий функционал:

- Добавление новых клиентов
- Просмотр всех клиентов
- Просмотр всех туров
- Добавление новых туров
- Добавление нового персонала
- Получение отчётов по контрактам/клиентам

## Models

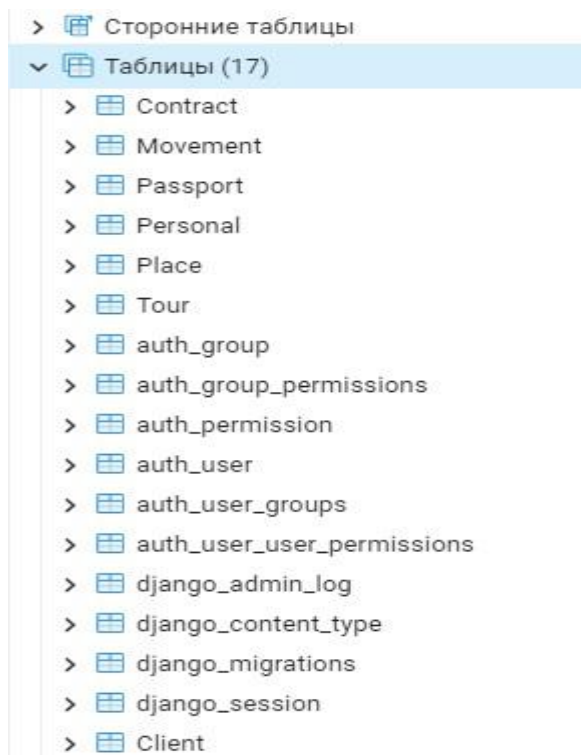
Основным механизмом работы с БД является ORM Django, поэтому сначала были описаны модели, а после чего были сделаны начальные миграции с помощью команды:

```
python manage.py makemigrations len
```

И миграция была применена с помощью команды:

```
python manage.py migrate
```

В результате в базы данных были созданы соответствующие модели, а также добавились новые таблицы, которые являются дефолтными для приложения Django (связанные с аутентификацией, миграциями, разделением ролей, администрированием и др.).





## View

Все представления для работы с таблицами БД для удобства были написаны в виде классов, наследованных от базового класса View. Для части манипуляций данными было описано одно представление:

```
class Delete(View):
    @staticmethod
    @login_required
    def get(request, etype, pk):
        role = get_user_role(request)

        if role == 'Manager':
            if etype == 'sellers':
                Personal.objects.get(id=pk).delete()
            elif etype == 'tours':
                Tour.objects.get(id=pk).delete()
            else:
                raise Http404
        elif role == 'Seller':
            if etype == 'clients':
                Client.objects.get(id=pk).delete()
            elif etype == 'contracts':
                Contract.objects.get(id=pk).delete()
            else:
                raise Http404
        else:
            raise Http404

        return redirect('entities', etype)
```

## Paginator

Для представлений, осуществляющих вывод списка объектов из QuerySet была добавлена пагинация, реализованная с помощью класса Paginator. На каждой странице выводятся 3-5 объектов.

Притом если после фильтрации на странице не удастся разместить столько элементов (и, соответственно, разделить на страницы), будет выведено столько элементов, сколько получилось после фильтрации.

```
class List(View):
    @staticmethod
    @login_required
    def get(request, etype):
        role = get_user_role(request)
        entities = None
        filter_ = None
        pgs = 5

        if role == 'Manager':
            if etype == 'sellers':
                entities =
Personal.objects.filter(groups__name='Seller').order_by('id')
                # filter_ = LFPFilter(request.GET, queryset=entities)
            elif etype == 'tours':
                entities = Tour.objects.all().order_by('id')
                # filter_ = TourFilter(request.GET, queryset=entities)
                pgs = 3
        else:
```

```

        raise Http404
    elif role == 'Seller':
        if etype == 'clients':
            entities = Client.objects.order_by('id')
            # filter = ClientFilter(request.GET, queryset=entities)
        elif etype == 'tours':
            entities = Tour.objects.all().order_by('cost')
            # filter_ = TourFilter(request.GET, queryset=entities)
            pgs = 3
        elif etype == 'contracts':
            entities =
Contract.objects.filter(seller=request.user).order_by('time')
            # filter_ = ContractFilter(request.GET, queryset=entities)
            pgs = 3
        else:
            raise Http404
    elif role == 'Client':
        if etype == 'tours':
            entities = Tour.objects.all().order_by('-id')
            # filter_ = ContractFilter(request.GET, queryset=entities)
            pgs = 3
        elif etype == 'my_movements':
            contracts =
Contract.objects.filter(client=Client.objects.get(id=request.user.pk)).order_
by('time')
            entities = []
            for tmp in contracts:
                entities.append(tmp.tour)
            # filter_ = MovementFilter(request.GET, queryset=entities)
            etype = 'movements'
        elif etype == 'my_tours':
            contracts =
Contract.objects.filter(client=Client.objects.get(id=request.user.pk)).order_
by('time')
            entities = []
            for tmp in contracts:
                entities.append(tmp.tour)
            # filter_ = TourFilter(request.GET, queryset=entities)
            pgs = 3
            etype = 'tours'
        else:
            raise Http404

    paginator = Paginator(entities, per_page=pgs)
    page_number = request.GET.get('page', None)
    if not page_number:
        page_number = 1

    page_obj = paginator.get_page(page_number)
    page_obj.adjusted_elided_pages =
paginator.get_elided_page_range(page_number)

    context = {
        'page_obj': page_obj,
        'filter_': filter_,
        'etype': etype,
    }

    return render(request, f'ta_web/entities.html', context=context)

```

**В шаблоне пагинация выглядит так:**

```

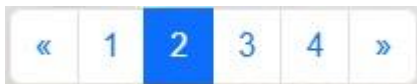
<nav>
    <ul class="pagination mt-2 mb-0">
        {% if page_obj.has_previous %}
            <li class="page-item">
                <a class="page-link" href="?page={{
page_obj.previous_page_number }}" aria-label="Previous">
                    <span aria-
hidden="true">&laquo;</span></a>
            </li>
        {% endif %}

        {% for page_number in
page_obj.adjusted_elided_pages %}
            {% if page_number ==
page_obj.paginator.ELLIPSIS %}
                <li class="page-item">
                    <a class="page-link disabled"
href="?page={{ page_number }}">{{ page_number }}</a>
                </li>
            {% else %}
                {% if page_obj.number == page_number %}
                    <li class="page-item">
                        <a class="page-link active"
href="?page={{ page_number }}">{{ page_number }}</a>
                    </li>
                {% else %}
                    <li class="page-item">
                        <a class="page-link"
href="?page={{ page_number }}">{{ page_number }}</a>
                    </li>
                {% endif %}
            {% endif %}
        {% endfor %}

        {% if page_obj.has_next %}
            <li class="page-item">
                <a class="page-link" href="?page={{
page_obj.next_page_number }}" aria-label="Next">
                    <span aria-
hidden="true">&raquo;</span></a>
            </li>
        {% endif %}
    </ul>
</nav>

```

На сайте же она отображается так:



## URLs

В папке приложения был отдельно создан файл `urls.py`, чтобы удобнее управлять URL адресами приложения.

В файле `urls.py` сайта записываем следующие адреса для перехода к админ панели и для перенаправления запросов к `ta_web.urls`:

```
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('ta_web.urls')),
] + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

В файле `ta_web/urls.py` прописываем все пути, которые связываем с `view`, либо функциями, реализующими нужный функционал. Кроме этого также выдаём каждому URL своё уникальное имя, чтобы удобнее создавать переходы в шаблонах `html` с помощью `Jinja2`:

```
from django.urls import path
from .views import *

urlpatterns = [
    path('', index, name='home'),
    path('about/', about, name='about'),
    path('feedback/', feedback, name='feedback'),
    path('register/', RegisterUser.as_view(), name='register'),
    path('login/', LoginUser.as_view(), name='login'),
    path('logout/', logout_user, name='logout'),
    path('profile/', Profile.as_view(), name='profile'),
    path('profile/update/', ProfileUpdate.as_view(), name='profile_update'),
    path('profile/delete/', profile_delete, name='profile_delete'),
    path('<str:etype>/list', List.as_view(), name='entities'),
    path('<str:etype>/<int:pk>', Card.as_view(), name='entity'),
    path('<str:etype>/add/', Add.as_view(), name='entity_add'),
    path('<str:etype>/<int:pk>/update/', Update.as_view(),
name='entity_update'),
    path('<str:etype>/<int:pk>/delete/', Delete.as_view(),
name='entity_delete'),

    path('export-<str:etype>-xlsx', get_entities_xlsx, name='entities_xlsx'),
    path('export-<str:etype>-json', get_entities_json, name='entities_json'),
]

handler404 = error_404_view
handler500 = error_500_view
```

## Templates

Отдельно стоит написать про шаблоны, которые использовались. Шаблоны в Django это, по сути, встроенный инструмент фронтенда, который представляет собой обычный язык HTML, усиленный возможностями Django.

Django templates Jinja позволяют создать основу сайта, чтобы затем не копировать большое количество кода из файла в файл. Базовый файл у меня называется base.html, в нем определена шапка сайта, подключены стили и библиотеки. Для некоторых функций и красивого оформления были подключены Bootstrap и jQuery.

### base.html:

```
{% load static %}
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, user-scalable=no,
initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>{% block title %}{% endblock %}</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/css/bootstrap.min.css">
    <script type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.7.0/jquery.min.js"></scr
ipt>
    <link rel="shortcut icon" href="{% static 'ta_web/images/favicon.ico'
%}" />
    {% block style %}
    {% endblock %}
</head>
<body>
    <nav class="navbar navbar-expand-md navbar-dark bg-dark fixed-top">
        <div class="container-fluid">
            <a href="{% url 'home' %}" class="navbar-brand w-50 me-auto" >
                
                Туристическое агентство
            </a>
            <button class="navbar-toggler" type="button" data-bs-
toggle="collapse" data-bs-target="#navbarCollapse" aria-
controls="navbarCollapse" aria-expanded="false" aria-label="Toggle
navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarCollapse">
                <ul class="navbar-nav me-auto mb-2 mb-md-0">
                    <li class="nav-item">
                        <a href="{% url 'home' %}" class="nav-
link">Главная</a>
                    </li>
                    <li class="nav-item">
                        <a href="{% url 'about' %}" class="nav-link">О
нас</a>
                    </li>
```

```

        <li class="nav-item">
            <a href="{% url 'feedback' %}" class="nav-
link">Обратная связь</a>
        </li>
    </ul>
    <form class="d-flex">
        {% if user.is_authenticated %}
            <a class="text-white text-decoration-none me-2 mt-
2">Приветствуем,</a>
            <a href="{% url 'profile' %}" class="text-white text-
decoration-underline me-3 mt-2">{{ user.username }}!</a>
            <a href="{% url 'logout' %}" class="btn btn-outline-
light align-content-end">Выйти</a>
        {% else %}
            <a href="{% url 'login' %}" class="btn btn-outline-
light me-2">Войти</a>
            <a href="{% url 'register' %}" class="btn btn-
outline-light">Регистрация</a>
        {% endif %}
    </form>
</div>
</div>
</nav>
<div id="bg">
    {% block background %}
    {% endblock %}
</div>
{% block content %}
{% endblock %}
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/js/bootstrap.bundle.min.js" integrity="sha384-
w76AqPfDkMBDXo30jS1Sgez6pr3x5MlQ1ZAGC+nuZB+EYdgRZgiwxhTBTkF7CXvN"
crossorigin="anonymous"></script>
</body>
<script>
    $(document).ready(
        function() {
            const sel = $('select[id="phone_0"]');
            const $aux =
$('<select/>').append($('<option/>').text(sel.find('option:selected').text())
);
            sel.after($aux)
            if($aux.width() > 228) {
                $('div[class="form-floating"]').width($aux.width() + 10);
            } else {
                $('div[class="form-floating"]').width("228");
            }
            $aux.remove()
        }
    );
</script>
<script>
    function maxLengthCheck(object) {
        if (object.value.length > object.maxLength)
            object.value = object.value.slice(0, object.maxLength)
    }
</script>
</html>

```

Можно заметить, что в коде определено несколько основных блоков которые потом расширялись в остальных файлах. В частности были созданы файлы для заголовка, стилей оформления, скриптов, а также основного «тела» сайта.

Вывод профиля, например, выглядит при этом так:

```
{% extends 'ta_web/base.html' %}
{% load static %}

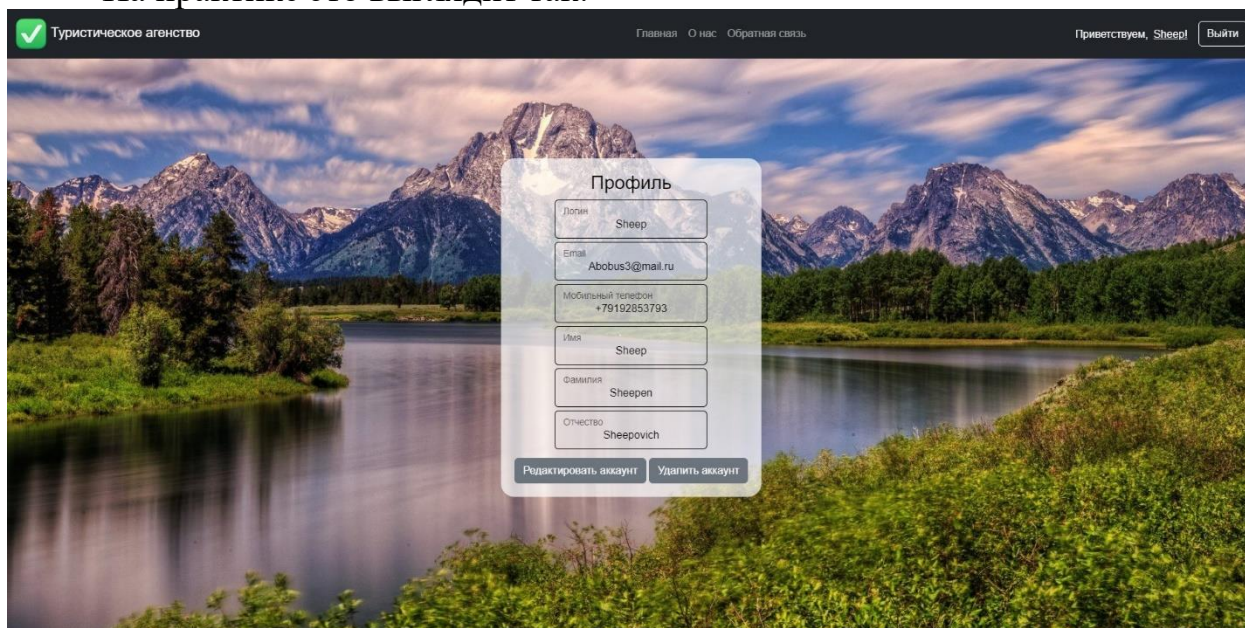
{% block title %}Профиль{% endblock %}

{% block style %}
    <link href="{% static 'ta_web/css/profile.css' %}" rel="stylesheet">
{% endblock %}

{% block content %}
    <main>
        {% block background %}
            
        {% endblock %}
        <div class="container">
            <form method="GET" class="form">
                <h1 class="h3 mb-2 fw-normal text-black">Профиль</h1>
                {% for field in form %}
                    <div class="form-floating">
                        <input id="{% if field.widget_type ==
'phonenumbertype' %}phone{% else %}{{ field.id_for_label }}{% endif %}"
class="form-control" readonly value="{{ field.value }}">
                        <label for="{% if field.widget_type ==
'phonenumbertype' %}phone{% else %}{{ field.id_for_label }}{% endif %}"
class="floating-input">{{ field.label }}</label>
                    </div>
                {% endfor %}
                {% if role == 'Manager' or role == 'Seller' %}
                    <a href="{% url 'profile_update' %}" type="button"
class="btn btn-secondary mt-2">Редактировать аккаунт</a>
                {% endif %}
                <a href="{% url 'profile_delete' %}" type="button" class="btn
btn-secondary mt-2">Удалить аккаунт</a>
            </form>
        </div>
        <script>

    </script>
    </main>
{% endblock %}
```

На практике это выглядит так:



Помимо этого стоит упомянуть, что в базовом шаблоне было сделано всё верхнее меню, которое может сворачиваться при уменьшении размеров экрана.



## Разделение полей пользователей (механизмы аутентификации):

У Django есть свои встроенные формы для регистрации с большим функционалом (форматы для ввода, хэширование паролей и т.д.). Именно они и были использованы.

Для того, чтобы настроить отображение на странице регистрации поля формы были написаны классы:

```
class RegisterUserForm(UserCreationForm):
    username = forms.CharField(
        label='Логин',
        widget=forms.TextInput(attrs={'class': 'form-control', 'id':
'login'})
    )

    email = forms.EmailField(
        label='Email',
        max_length=256,
        widget=forms.EmailInput(attrs={'class': 'form-control', 'id':
'email'})
    )

    password1 = forms.CharField(
        label='Пароль',
        widget=forms.PasswordInput(attrs={'class': 'form-control', 'id':
'password1'})
    )

    password2 = forms.CharField(
        label='Повтор пароля',
        widget=forms.PasswordInput(attrs={'class': 'form-control', 'id':
'password2'})
    )

    phone = forms.CharField(
        label='Мобильный телефон',
        widget=PhoneNumberPrefixWidget(attrs={
            'class': 'form-control',
            'id': 'phone',
        },
            initial='RU',
            country_attrs={'id': 'phone_0'},
            number_attrs={'id': 'phone1'}
        )
    )

    first_name = forms.CharField(
        label='Имя',
        max_length=256,
        widget=forms.TextInput(attrs={'class': 'form-control', 'id': 'name'})
    )

    last_name = forms.CharField(
        label='Фамилия',
        max_length=256,
        widget=forms.TextInput(attrs={'class': 'form-control', 'id':
'lName'})
    )
```

```

    patronymic = forms.CharField(
        label='Отчество',
        max_length=256,
        widget=forms.TextInput(attrs={'class': 'form-control', 'id':
'patronymic'})
    )

    birthday = forms.DateField(
        label='Дата рождения',
        widget=forms.DateInput(attrs={'class': 'form-control', 'id':
'birthday'})
    )

    series = forms.CharField(
        label='Серия паспорта',
        widget=forms.TextInput(attrs={
            'class': 'form-control',
            'id': 'series',
            'type': 'number',
            'maxlength': 4,
            'oninput': 'maxLengthCheck(this)'
        })
    )

    number = forms.CharField(
        label='Номер паспорта',
        widget=forms.TextInput(attrs={
            'class': 'form-control',
            'id': 'number',
            'type': 'number',
            'maxlength': 6,
            'oninput': 'maxLengthCheck(this)'
        })
    )

    code = forms.CharField(
        label='Код подразделения',
        widget=forms.TextInput(attrs={
            'class': 'form-control',
            'id': 'code',
            'type': 'number',
            'maxlength': 6,
            'oninput': 'maxLengthCheck(this)'
        })
    )

    issue_date = forms.DateField(
        label='Дата выдачи',
        widget=forms.DateInput(attrs={'class': 'form-control', 'id':
'issue_date'})
    )

    giver = forms.CharField(
        label='Кем выдан',
        max_length=256,
        widget=forms.TextInput(attrs={'class': 'form-control', 'id':
'giver'})
    )

    passport = forms.CharField(
        label='',

```

```

        max_length=256,
        required=False,
        widget=forms.HiddenInput(attrs={'class': 'form-control', 'id':
'passport'}))
    )

    class Meta:
        model = Client
        fields = ('username', 'email', 'password1',
                  'password2', 'phone', 'first_name',
                  'last_name', 'patronymic', 'birthday', 'passport')

    def clean(self):
        if not self._errors:
            cleaned_data = super(RegisterUserForm, self).clean()
            email = cleaned_data.get('email')
            phone = cleaned_data.get('phone')
            series = cleaned_data.get('series')
            number = cleaned_data.get('number')
            code = cleaned_data.get('code')
            try:
                try:
                    int(series)
                    int(number)
                    int(code)
                except ValueError:
                    raise forms.ValidationError(u'ВОТ СКАЖИ, ЗАЧЕМ ТЫ \'-\'
ВВОДИШЬ, А? ПУСЬКА ТЫ!')
                if User.objects.filter(email=email).exists():
                    raise forms.ValidationError(u'Пользователь с таким e-mail
уже существует.')
                elif Client.objects.filter(phone=phone).exists():
                    raise forms.ValidationError(u'Пользователь с таким
телефоном уже существует.')
                elif Passport.objects.filter(series=series, number=number,
code=code).exists():
                    raise forms.ValidationError(u'Пользователь с такими
паспортными данными уже существует.')
                finally:
                    try:
                        cleaned_data['passport'] =
Passport.objects.create(series=series,
number=number,
code=code,
issue_date=cleaned_data.get('issue_date'),
giver=cleaned_data.get('giver'))
                    finally:
                        return cleaned_data

class RegisterPersonalForm(UserCreationForm):
    username = forms.CharField(
        label='Логин',
        widget=forms.TextInput(attrs={'class': 'form-control', 'id':
'login'}))
    )

```

```

        email = forms.EmailField(
            label='Email',
            max_length=256,
            widget=forms.EmailInput(attrs={'class': 'form-control', 'id':
'email'}})
        )

        password1 = forms.CharField(
            label='Пароль',
            widget=forms.PasswordInput(attrs={'class': 'form-control', 'id':
'password1'}})
        )

        password2 = forms.CharField(
            label='Повтор пароля',
            widget=forms.PasswordInput(attrs={'class': 'form-control', 'id':
'password2'}})
        )

        phone = forms.CharField(
            label='Мобильный телефон',
            widget=PhoneNumberPrefixWidget(attrs={
                'class': 'form-control',
                'id': 'phone'
            },
            initial='RU',
            country_attrs={'id': 'phone_0'},
            number_attrs={'id': 'phone1'}
        )
    )

    first_name = forms.CharField(
        label='Имя',
        max_length=256,
        widget=forms.TextInput(attrs={'class': 'form-control', 'id': 'name'})
    )

    last_name = forms.CharField(
        label='Фамилия',
        max_length=256,
        widget=forms.TextInput(attrs={'class': 'form-control', 'id':
'lName'})
    )

    patronymic = forms.CharField(
        label='Отчество',
        max_length=256,
        widget=forms.TextInput(attrs={'class': 'form-control', 'id':
'patronymic'})
    )

    class Meta:
        model = Personal
        fields = ('username', 'email', 'password1',
            'password2', 'phone', 'first_name',
            'last_name', 'patronymic')

class LoginUserForm(AuthenticationForm):
    username = forms.CharField(
        label='Логин',

```

```

        widget=forms.TextInput(attrs={'class': 'form-control', 'id':
'login'})
    )
    password = forms.CharField(
        label='Пароль',
        widget=forms.PasswordInput(attrs={'class': 'form-control', 'id':
'password'})
    )

```

Переопределение формы полей даёт подстроить возможность их вывода на странице в нужном стиле, используя CSS-стили.

Представления для отображения страниц регистрации и авторизации:

```

class RegisterUser(CreateView):
    form_class = RegisterUserForm
    template_name = 'ta_web/registration.html'
    success_url = reverse_lazy('login')

    def form_valid(self, form):
        user = form.save()
        user.groups.add(Group.objects.get(name='Client'))
        login(self.request, user)
        return redirect('home')

class LoginUser(LoginView):
    form_class = LoginUserForm
    template_name = 'ta_web/login.html'
    redirect_authenticated_user = reverse_lazy('home')

    def get_success_url(self):
        return reverse_lazy('home')

```

Для выхода была написана следующая функция:

```

@login_required
def logout_user(request):
    logout(request)
    return redirect('home')

```

Для разграничения доступа к страницам, был написан декоратор в файле **decorators.py**:

```

from django.http import Http404
from functools import wraps

def allowed_roles(allowed=None):
    if allowed is None:
        allowed = []

    def decorator(view_func):
        @wraps(view_func)
        def _wrapped_view(request, *args, **kwargs):
            user = request.user
            if user and user.groups.exists() and user.groups.all()[0].name in
allowed:
                return view_func(request, *args, **kwargs)
            else:
                raise Http404

```

```
return _wrapped_view
```

```
return decorator
```

Декоратор `allowed_users` разрешает просмотр страницы только тем зарегистрированным пользователям, которым выдана роль, передающаяся в аргументах при его использовании.

Для проверки авторизованности пользователя был использован стандартный Django-декоратор `@loginrequired`.

Были определены следующие группы ролей:

<input type="checkbox"/>	ГРУППА
<input type="checkbox"/>	Admin
<input type="checkbox"/>	Client
<input type="checkbox"/>	Django specific
<input type="checkbox"/>	Manager
<input type="checkbox"/>	Seller

## Выгрузка данных из БД

Необходимо было по сформированным запросам осуществить выборку в формате: Json, Xlsx.

```
def get_xlsx(request, qset, columns, ws_name):
    output = io.BytesIO()

    filename = f'current-{ws_name}.xlsx'
    workbook = xlsxwriter.Workbook(output, {'in_memory': True, 'default_date_format': 'yyyy/mm/dd',
                                             'time_format': 'hh:mm', 'remove_timezone': True})
    worksheet = workbook.add_worksheet(ws_name)

    row_num = 0
    for col_num in range(len(columns)):
        worksheet.write(row_num, col_num, columns[col_num])

    print(field for field in columns)
    rows = qset.order_by(columns[0]).values_list(*columns)

    for row in rows:
        row_num += 1
        for col_num in range(len(row)):
            worksheet.write(row_num, col_num, row[col_num])

    workbook.close()
    output.seek(0)

    response = HttpResponse(output.read(),
                             content_type='application/vnd.openxmlformats-officedocument.spreadsheetml.sheet')
    response['Content-Disposition'] = f'attachment; filename={filename}'

    output.close()
```

```
return response
```

```
def get_json(request, items_list, filename):
    serialized_items = serialize('json', items_list)
    print(serialized_items)

    with open(filename, 'w') as f:
        jsoned = json.loads(serialized_items)
        json.dump(jsoned, f, indent=2, ensure_ascii=False)

    response = HttpResponse(serialized_items, content_type='application/json')
    response['Content-Disposition'] = f'attachment; filename={filename}'

    return response
```

Выгрузка осуществляется следующим образом:

```
@login_required(login_url='login')
def get_entities_xlsx(request, etype):
    role = get_user_role(request)
    columns = None
    entities = None
    placeholder = None

    if role == 'Manager':
        if etype == 'sellers':
            entities = Personal.objects.filter(groups__name='Seller').order_by('id')
            columns = [f.name for f in Personal._meta.get_fields()]
            placeholder = 'Продавцы'
        elif etype == 'tours':
            entities = Tour.objects.all().order_by('id')
            columns = [f.name for f in Tour._meta.get_fields()]
            placeholder = 'Туры'
        else:
            raise Http404
    elif role == 'Seller':
        if etype == 'clients':
            entities = Client.objects.order_by('id')
            columns = [f.name for f in Client._meta.get_fields()]
            placeholder = 'Клиенты'
        elif etype == 'tours':
            entities = Tour.objects.all().order_by('cost')
            columns = [f.name for f in Tour._meta.get_fields()]
            placeholder = 'Туры'
        elif etype == 'contracts':
            entities = Contract.objects.filter(seller=request.user).order_by('time')
            columns = [f.name for f in Contract._meta.get_fields()]
            placeholder = 'Контракты'
        else:
            raise Http404

    return get_xlsx(request, entities, columns, ws_name=placeholder)
```

```
@login_required()
def get_entities_json(request, etype):
    role = get_user_role(request)
    entities_list = None
    filename = None
```

```

if role == 'Manager':
    if etype == 'sellers':
        entities_list = Personal.objects.filter(groups__name='Seller').order_by('id')
        filename = 'Продавцы'
    elif etype == 'tours':
        entities_list = Tour.objects.all().order_by('id')
        filename = 'Туры'
    else:
        raise Http404
elif role == 'Seller':
    if etype == 'clients':
        entities_list = Client.objects.order_by('id')
        filename = 'Клиенты'
    elif etype == 'tours':
        entities_list = Tour.objects.all().order_by('cost')
        filename = 'Туры'
    elif etype == 'contracts':
        entities_list = Contract.objects.filter(seller=request.user).order_by('time')
        filename = 'Контракты'
    else:
        raise Http404

```

```

return get_json(request, entities_list, filename)

```

id	username	first_name	last_name	email	patronymic	phone	birthday	passport
4	Test1	Тест1	Тестовый1	test1@mail.r	Тестович1	+790000000001	1991.10.15	1
11	Test8	Тест8	Тестовый8	test8@mail.r	Тестович8	+790000000008	1991.10.22	8
17	Test14	Тест14	Тестовый14	test14@mail.	Тестович14	+790000000001	1991.10.28	14
12	Test9	Тест9	Тестовый9	test9@mail.r	Тестович9	+790000000009	1991.10.23	9
10	Test7	Тест7	Тестовый7	test7@mail.r	Тестович7	+790000000007	1991.10.21	7
18	Test15	Тест15	Тестовый15	test15@mail.	Тестович15	+790000000001	1991.10.29	15
15	Test12	Тест12	Тестовый12	test12@mail.	Тестович12	+790000000001	1991.10.26	12
13	Test10	Тест10	Тестовый10	test10@mail.	Тестович10	+790000000001	1991.10.24	10
5	Test2	Тест2	Тестовый2	test2@mail.r	Тестович2	+790000000002	1991.10.16	2
19	Test16	Тест16	Тестовый16	test16@mail.	Тестович16	+790000000001	1991.10.30	16
8	Test5	Тест5	Тестовый5	test5@mail.r	Тестович5	+790000000005	1991.10.19	5
6	Test3	Тест3	Тестовый3	test3@mail.r	Тестович3	+790000000003	1991.10.17	3
16	Test13	Тест13	Тестовый13	test13@mail.	Тестович13	+790000000001	1991.10.27	13
14	Test11	Тест11	Тестовый11	test11@mail.	Тестович11	+790000000001	1991.10.25	11
9	Test6	Тест6	Тестовый6	test6@mail.r	Тестович6	+790000000006	1991.10.20	6
7	Test4	Тест4	Тестовый4	test4@mail.r	Тестович4	+790000000004	1991.10.18	4



```
[
{
  "model": "ta_web.client",
  "pk": 4,
  "fields": {
    "patronymic": "Тестович1",
    "phone": "+790000000001",
    "birthday": "1991-10-15",
    "passport": 1
  }
},
{
  "model": "ta_web.client",
  "pk": 5,
  "fields": {
    "patronymic": "Тестович2",
    "phone": "+790000000002",
    "birthday": "1991-10-16",
    "passport": 2
  }
},
{
  "model": "ta_web.client",
  "pk": 6,
  "fields": {
    "patronymic": "Тестович3",
    "phone": "+790000000003",
    "birthday": "1991-10-17",
    "passport": 3
  }
},
{
  "model": "ta_web.client",
  "pk": 7,
  "fields": {
    "patronymic": "Тестович4",
    "phone": "+790000000004",
    "birthday": "1991-10-18",
    "passport": 4
  }
},
{
  "model": "ta_web.client",
  "pk": 8,
  "fields": {
    "patronymic": "Тестович5",
    "phone": "+790000000005",
    "birthday": "1991-10-19",
    "passport": 5
  }
},
{

```

## Создание резервной копии БД

Было разработано автоматизированное создание резервной копии базы данных и хранение ее в облаке.

Непосредственное создание копии осуществляется с помощью утилиты резервного копирования `pg_dump.exe` через bat-файл:

```
REM СОЗДАНИЕ РЕЗЕРВНОЙ КОПИИ БАЗЫ ДАННЫХ POSTGRESQL
CLS
ECHO OFF
CHCP 1251

SET PYTON=C:\Users\MegaEater\AppData\Local\Programs\Python\Python310
SET CLOUD=F:\MegaEater\PyCharmProjects\cloudSaver\main.py

REM Установка переменных окружения
SET PGBIN=F:\Program Files\PostgreSQL\14\bin
SET PGDATABASE=TourAgency
SET PGHOST=localhost
SET PGPORT=5432
SET PGUSER=postgres
SET PGPASSWORD=

REM Смена диска и переход в папку из которой запущен bat-файл
%~d0 REM возвращает диск
CD %~dp0 REM переходит к bat-файлу

REM Формирование имени файла резервной копии и файла-отчета
SET DATETIME=%DATE:~6,4%-~3,2%-~0,2% %TIME:~0,2%-~3,2%-~6,2%
SET DUMPFIL=%PGDATABASE% %DATETIME%.sql
SET LOGFIL=%PGDATABASE% %DATETIME%.log
SET DUMPPATH="backup\daily%\%DUMPFIL%"
SET LOGPATH="backup%\%LOGFIL%"

REM Непосредственное СОЗДАНИЕ РЕЗЕРВНОЙ КОПИИ
IF NOT EXIST backup\daily\Daily MD backup\daily
IF NOT EXIST Daily\%DATETIME% MD Daily\%DATETIME%
CALL "%PGBIN%\pg_dump.exe" --format=plain --verbose --file=%DUMPPATH%
2>%LOGPATH%

REM Анализ кода завершения
IF NOT %ERRORLEVEL%==0 GOTO Error
GOTO Successfull

REM В случае ошибки удаляется поврежденная резервная копия и делается
соответствующая запись в журнале
:Error
DEL %DUMPPATH%
MSG * "Ошибка при создании резервной копии базы данных. Смотрите backup.log."
ECHO %DATETIME% Ошибки при создании резервной копии базы данных %DUMPFIL%.
Смотрите отчет %LOGFIL%. >> backup.log
GOTO End

REM В случае удачного резервного копирования делается запись в журнал, и
вызывается программа загрузки в облако
:Successfull
ECHO %DATETIME% Успешное создание резервной копии %DUMPFIL% >> backup.log
cd PYTHON
```

```
python.exe CLOUD
```

```
REM Анализ кода завершения
```

```
IF NOT %ERRORLEVEL%==0 GOTO ErrorCloud
```

```
GOTO End
```

```
REM В случае ошибки делается соответствующая запись в журнале
```

```
:ErrorCloud
```

```
MSG * "Ошибка при отправке резервной копии базы данных на облако. Смотрите backup.log."
```

```
cd CD %~dp0
```

```
ECHO %DATE% Ошибки при отправке созданной выше резервной копии базы данных на облако. Смотрите отчет %LOGFILE%. >> backup.log
```

```
GOTO End
```

```
:End
```

Для того, чтобы создание резервной копии происходило регулярно, необходимо создать задание для планировщика Windows с использованием команды SHTASKS:

```
CLS
```

```
ECHO OFF
```

```
CHCP 1251
```

```
SHTASKS /Create /RU SYSTEM /SC DAILY /TN "Резервная копия" /TR  
"F:MegaEater\ta_web_backup\backup.bat" /ST 16:00:00
```

```
IF NOT %ERRORLEVEL%==0 MSG * "Ошибка при создании задачи резервного  
копирования! %ERRORLEVEL%"
```

Размещение резервной копии происходит в облаке, на Яндекс Диск. Для регулярной загрузки бэкапа в облако был написан скрипт и bat-file для его запуска:

```
import yadisk
```

```
import os
```

```
import locale
```

```
import schedule
```

```
y = yadisk.YaDisk(token="")
```

```
DAILY_DIR = "/daily-backup-db"
```

```
FILE_PATH = r' F:MegaEater\ta_web_backup\daily'
```

```
def backup_daily():
```

```
    for address, dirs, files in os.walk(FILE_PATH):
```

```
        for file in files:
```

```
            if not y.exists(f'{DAILY_DIR}/{file}')
```

```
                y.upload(f'{address}/{file}', f'{DAILY_DIR}/{file}')
```

```
                print(f'Файл {file} загружен')
```

```
def main():
```

```
    schedule.every().day.at('16:05').do(backup_daily)
```

```
    while True:
```

```
        schedule.run_pending()
```

```

if __name__ == '__main__':
    locale.setlocale(locale.LC_ALL)
    print(y.check_token())
    main()

```

```

@echo off
SET CLOUDSAVER=F:\MegaEater\PyCharmProjects\cloudSaver\main.py
SET PYTHON=C:\Users\MegaEater\AppData\Local\Programs\Python\Python310
cd %PYTHON%
python.exe %CLOUDSAVER%
pause

```

### Результаты работы автоматизированного создания резервной копии:

.idea	09.01.2023 16:11	Папка с файлами
backup	09.01.2023 15:56	Папка с файлами
venv	09.01.2023 15:07	Папка с файлами
backup.bat	09.01.2023 15:56	Пакетный файл Wi...
backup.log	09.01.2023 15:56	Текстовый документ
main.py	09.01.2023 16:01	JetBrains PyCharm
schedule_backup.bat	09.01.2023 15:42	Пакетный файл Wi...

backup.log – Блокнот

Файл

Правка

Формат

Вид

Справка

2023-05-26 11-56-20 Успешное создание резервной копии 2023-05-26 11-56-20.sql

### Полученный файл бэкапа:

 len_ceramic 2023-05-26 11-56-20.sql	26.05.2023 11:56	JetBrains DataGrip
---	------------------	--------------------

### Бэкап, размещенный на Яндекс Диске:


Файлы

По названию

←

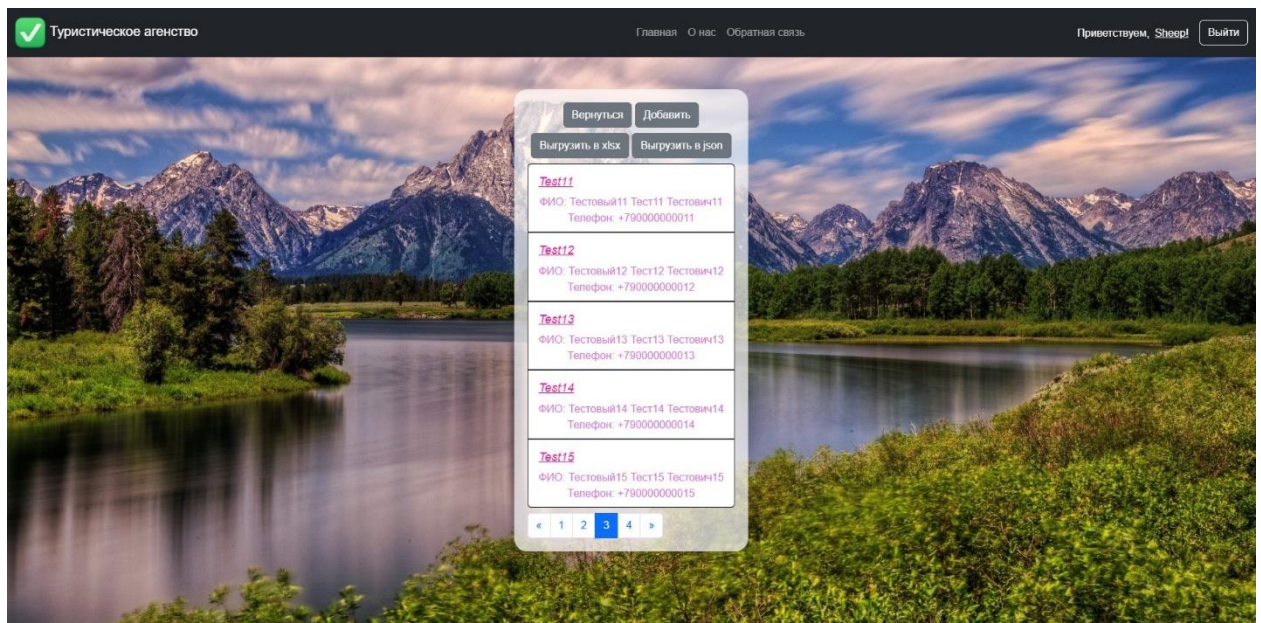
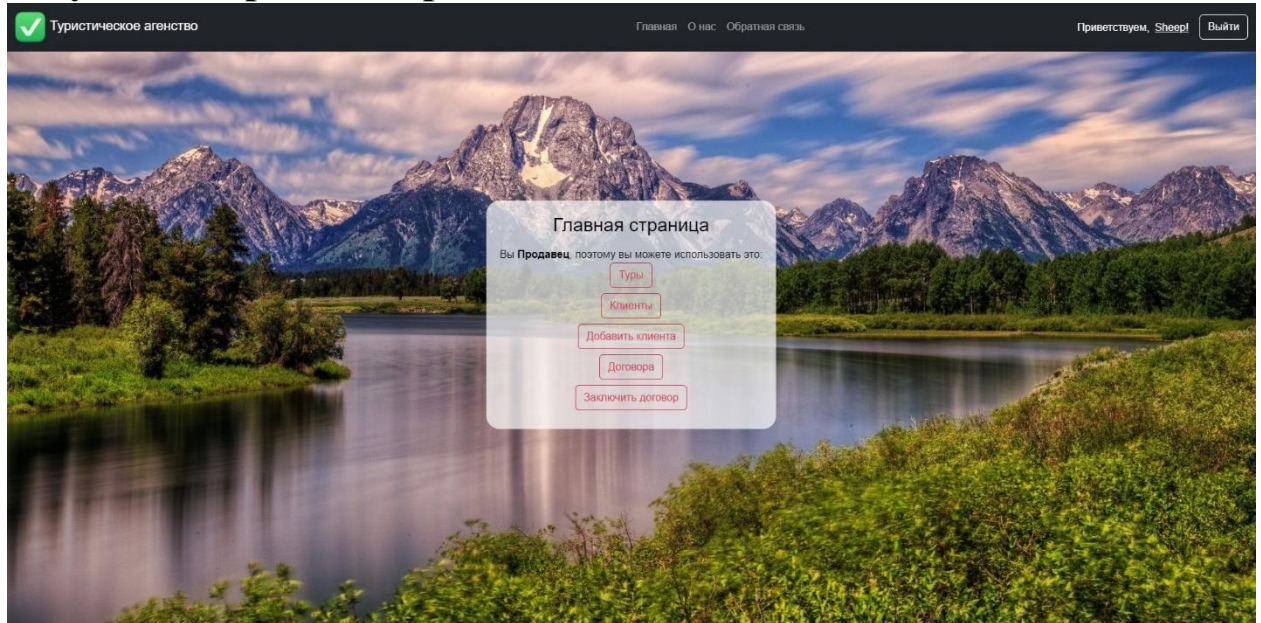
daily-backup-db

:



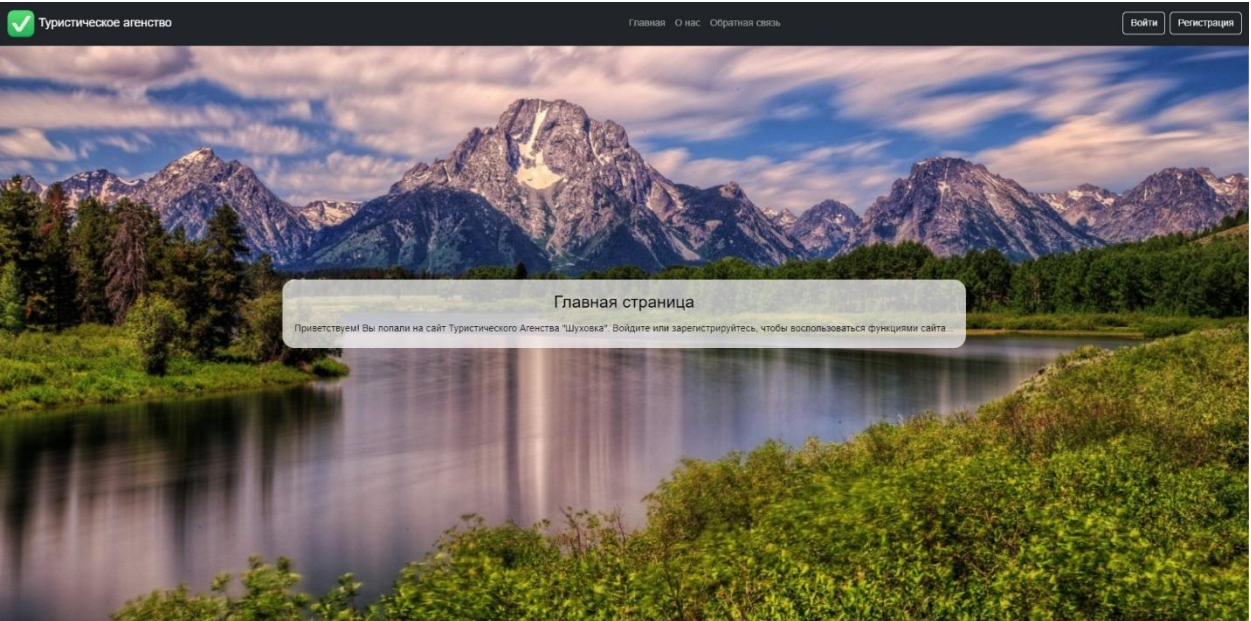
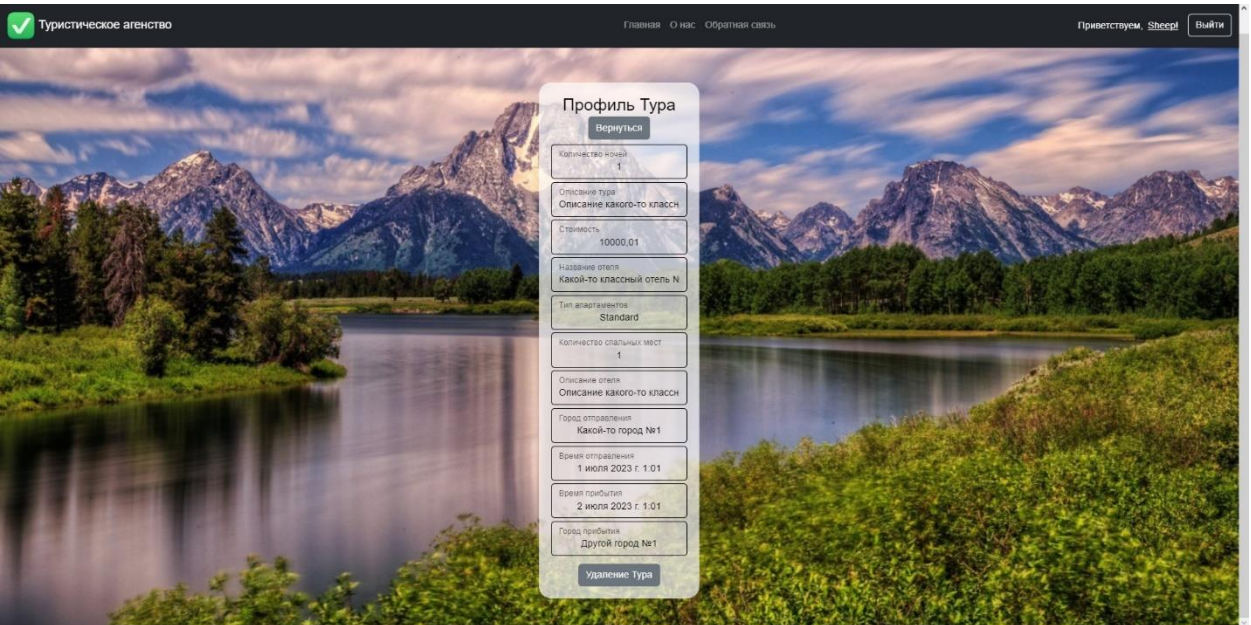
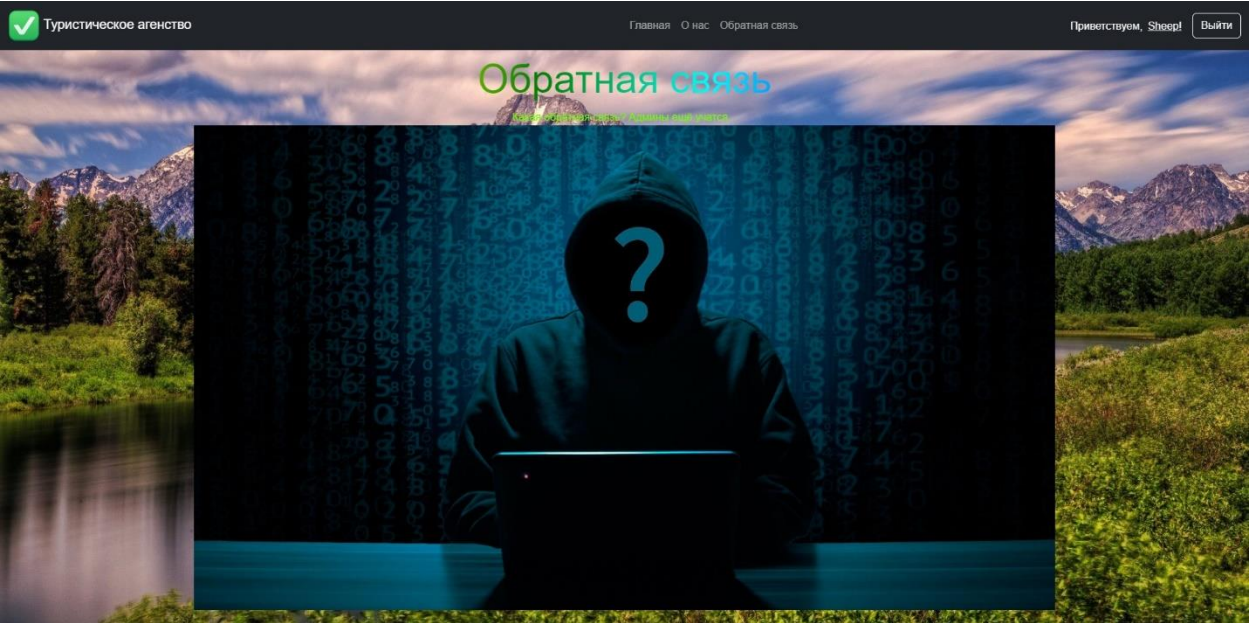
2023-0...-20.sql

# Результаты работы приложения

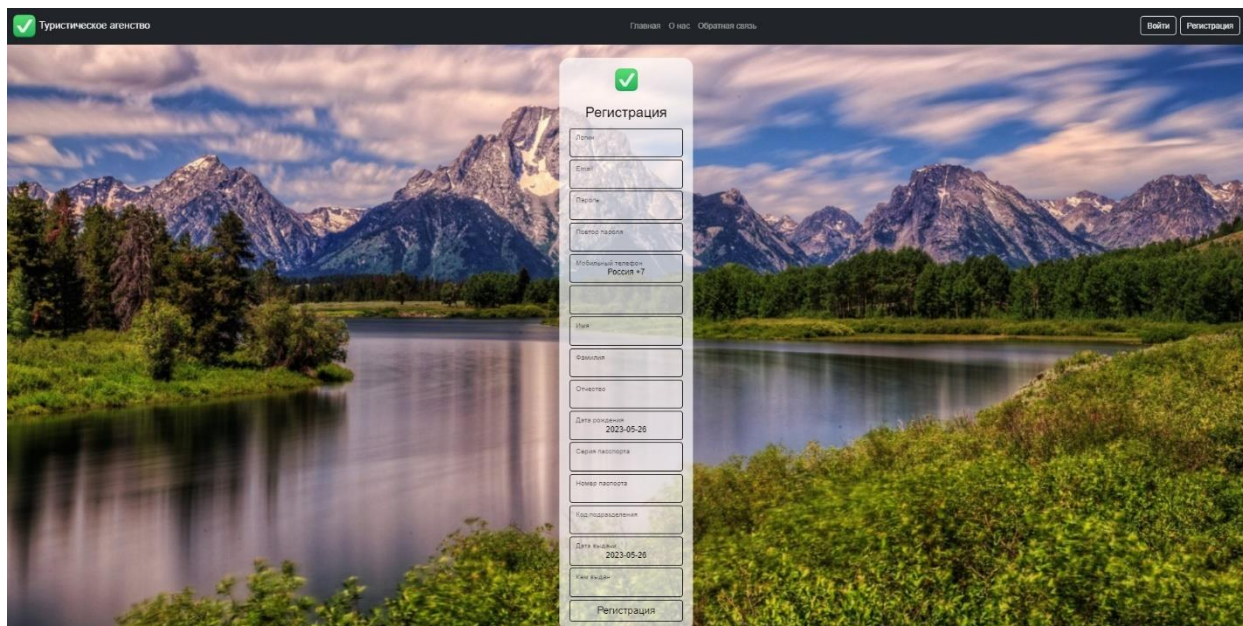


Полный исходный код можно найти по адресу:  
[https://github.com/me\\_go\\_42/ta\\_web](https://github.com/me_go_42/ta_web).









## Заключение

В ходе написания курсовой работы были закреплены навыки и знания, полученные в рамках курса «Базы данных». Также были изучены основы framework'a Django, получен опыт работы с ним, а также разработаны web-приложение и база данных Туристического агентства с учётом требований к работе.

## Список литературы

[В Интернете]. - <https://docs.python.org/3/library/tkinter.html>.

[В Интернете]. - <https://ramziv.com/article/2>.