

Geekbrains

**Разработка web-приложения
«Виртуальный офис: мои документы»
на языке Python с использованием фреймворка Django**

Программирование:
Разработчик Python
Шевцов А.П.

Москва
2022

Оглавление

Введение.....	4
Глава 1. Основы разработки web-приложений на языке Python	6
1.1 Что такое web-приложение, его особенности.....	6
1.2 Особенности фреймворка Django, его преимущества и краткая история ...	7
1.3 Концепция MVT	9
Глава 2. Подготовка среды и создание проекта	11
2.1 Виртуальное окружение	11
2.2 Установка и настройка Django	12
2.3 Архитектура проекта	12
2.4 Создание структуры проекта	14
2.4 Запуск сервера и проверка работоспособности	16
2.5 Создание приложений, обзор их структуры и добавление в проект	18
2.6 Выбор метода представлений в проекте.....	20
2.7 Создание представлений	21
2.8 Определение маршрутов	22
2.9 Логирование в проекте	24
Глава 3. Организация работы с моделями проекта.....	29
3.1 Общие сведения о моделях в Django	29
3.2 Определение моделей	29
3.3 Миграции	32
3.4 Создание собственных команд manage.py	34
3.5 Работа с данными в моделях.....	35
Глава 4. Работа с шаблонами на основе представлений	40
4.1 Сопоставление представления с маршрутом	40
4.2 Шаблоны	41
4.3 Наследование шаблонов.....	43
Глава 5. Работа с формами	47
5.1 Создание форм	47
5.2 Представление и маршруты для форм.....	52

5.3 Валидация данных форм	54
5.4 Сохранение данных форм в базу данных	55
Глава 6. Административная панель и Django Debug Toolbar	57
6.1 Работа с административной панелью Django.....	57
6.2 Панель Django Debug Toolbar	59
Глава 7. Развёртывание проекта	62
7.1 Настройки безопасности	62
7.2 Настройки подключения к базе данных	63
7.3 Формирование списка пакетов	65
7.4 Создание репозитория GitHub	65
7.5 Настройка проекта на сервере	66
7.6 Создание web-приложения.....	67
7.7 Настройка web-приложения.....	68
7.8 Создание баз данных	70
Глава 8. Тестирование.....	73
Заключение	75
Список использованной литературы и ресурсов	76
Приложения	77
Приложение 1. UserCase-диаграмма	77
Приложение 2. User Interface	78
Приложение 3. ERD-диаграмма	82
Приложение 4. Код приложения и описание кода	87
Приложение 5. Краткая информация по работе с Git.	89
Приложение 6. Ответ по использованию API Госуслуги	90

Введение

В настоящее время все больше сфер жизни человека переходит в цифровое пространство. При нынешних темпах развития информационных технологий и скорости современной жизни становится очевидной необходимость перехода документооборота на цифровые носители. Не являются исключением и личные документы физических лиц. И хотя основным инструментом для работы с документами для физических лиц на сегодняшний день остаются Госуслуги, существует потребность в дополнительном развитии данного направления, в том числе, со специализацией на более узконаправленный функционал с подключением дополнительных возможностей.

Данный проект предусматривает углубление знаний о фреймворке Django посредством создания web-приложения «Виртуальный офис: мои документы». Приложение направлено на цифровизацию личных документов пользователя с интеграцией функцией просмотра и добавление контактов пользователя, планировщика и др. В работе описывается создание личного кабинета пользователя с применением фреймворка Django.

Тема проекта: Разработка web-приложения «Виртуальный офис: мои документы» на языке Python с использованием фреймворка Django

Цель проекта: Изучить особенности программирования на языке Python, получить практические навыки использования фреймворка Django, разработать web-приложение узкой направленности с возможностью расширения функционала в соответствии с потребностями пользователей

Какую проблему решает проект: возможность цифровизации личных документов физических лиц без прохождения дополнительной авторизации с возможностью загрузки данных с сервера Госуслуги, реализация постоянного доступа к данным при отсутствии интернет-подключения.

Задачи проекта:

1. Изучить литературу, касающуюся темы исследования.

2. Рассмотреть основные принципы работы языка Python во взаимодействии с фреймворком Django.

3. Ознакомиться с основными составляющими web-приложения в соответствии с паттерном «Model-View-Template» (MVT): модели, представления, шаблоны.

4. Составить схему функционала приложения (UserCase)

5. Разработать приложение в соответствии со схемой UserCase.

6. Реализовать функционал работы пользователя и администрирования приложения.

7. Провести тестирование приложения на этапе разработки.

8. Развернуть приложение на web-сервере.

Инструменты: Python, Django, Git, Visual Studio Code.

Состав команды: Шевцов Алексей Павлович - разработчик

В рамках реализации проекта реализованы следующие действия:

- выполнен анализ существующих решений поставленной проблемы, осуществлен сбор информации о необходимости данного сервиса;

- выполнена разработка UserCase модели приложения;

- разработан пользовательский интерфейс в первом приближении;

- проведена разработка приложения и его тестирование на этапе разработки;

- произведена оценка возможного усовершенствования, расширения и дальнейшего развития.

Глава 1. Основы разработки web-приложений на языке Python

1.1 Что такое web-приложение, его особенности

Web-сервис или web-приложение — это программа, которая осуществляет исполнение своей логики через web-браузер пользователя. В качестве примера web-приложений можно привести поиск Яндекса или онлайн-словарь, переводчик, интернет-магазин и т.д.

Web-сервис отличается от обычного приложения тем, что вся его работа происходит на сервере, а результат операций возвращается пользователям в виде web-страницы. Когда мы задаём вопрос Яндексу, этот вопрос улетает в центр, где на основе огромного количества данных готовится ответ, и он возвращается в браузер в виде простой страницы.

Web-сервисы используют свои внутренние языки: на серверах, которые обрабатывают запросы браузера, работают программы, которые обеспечивают все необходимую логику web-приложения.

Одним из популярных языков web-приложений является язык 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 в том плане, что всё является объектами. В настоящее время для Python создано достаточно большое количество фреймворков.

Фреймворк — это надстройка над языком программирования, которая упрощает разработку. Внутри фреймворка собрано множество готовых конструкций, которые разработчику не нужно выполнять самому — вместо этого он использует команды фреймворка и сразу получает результат.

1.2 Особенности фреймворка Django, его преимущества и краткая история

Django — это высокоуровневый фреймворк для web-приложений на языке Python. Он был создан в 2005 году, и с тех пор активно развивается и обновляется сообществом разработчиков по всему миру.

Django был создан в 2005 году. Первый коммит в июле 2005, а версия 0.90 в ноябре того же года. С тех пор Django активно развивается и обновляется сообществом разработчиков по всему миру. В настоящее время он является одним из самых популярных фреймворков для web-разработки на языке Python, в связи с чем, в том числе, и был выбран для использования в данном проекте.

Стоит отметить, что Django достаточно долго был в нулевой и первой версии. С 2005 по 2014 год обновления выходили хаотично, по мере создания. С 2015 года закрепились месяцы выхода обновлений: апрель, декабрь, август.

Начиная с версии 1.4, стали выходить версии LTS.

LTS (Long Time Support) — версия, с поддержкой в течение длительного времени (3 года). LTS-версии стабильнее по сравнению с обычными, так как при их выпуске разработчики стараются не экспериментировать со всевозможными новинками.

В декабре 2017 года выходит Django 2.0. Он меняет подход к нумерации версий. С тех пор это тройка x.0, x.1 и x.2. При этом версии x.2 выходят как LTS.

Даже если сейчас последняя версия имеет номер 5, это не значит, что тот же Django 3.2 более не актуален и использовать его нецелесообразно. Большая часть материалов, используемых в рамках данного проекта, появилась в первых версиях Django и сохраняет свою актуальность до сих пор.

Django предоставляет разработчикам множество инструментов и функций для создания web-приложений, таких как:

- ORM (Object-Relational Mapping) для работы с базами данных
- URL-маршрутизация
- Аутентификация и авторизация пользователей
- Шаблонизация

- Кеширование
- Интернационализация
- Административная панель

Использование Django имеет множество преимуществ, таких как:

- Быстрая разработка web-приложений
- Простота и удобство использования
- Высокая производительность
- Безопасность
- Масштабируемость

Большим преимуществом Django является встроенная панель управления сайтом. Разработчику не нужно писать свою административную панель — она уже доступна сразу после запуска сайта.

В панели управления сайтом можно:

- настраивать структуру сайта;
- управлять пользователями;
- работать с фильтрами и выгружать все необходимые данные;
- поправить код и сразу увидеть изменения на сайте;
- работать с базой данных сайта;
- просматривать статистику.

Так как административная панель также написана на Python, то её можно оптимизировать или переделать под задачи проекта.

Резюмируя, можно отметить следующие преимущества фреймворка Django:

- Скорость разработки. Django с самого начала создавался как фреймворк для быстрой сборки. С ним за короткие сроки возможно собрать рабочую версию сайта, протестировать её и определить дальнейшие действия.
- Масштабирование. В Django почти всё основано на модулях. Если нужно расширить сайт или масштабироваться под посещение миллиона человек в день, можно один модуль заменить другим. Другие модули при этом продолжат свою работу без изменений.

- Безопасность. В Django встроено много готовых решений для безопасной работы: система аутентификации, защита от инъекций кода, межсайтовых запросов, подмены заголовка хоста и от других потенциальных уязвимостей.

- Библиотеки и расширения. Если по умолчанию чего-то нет в Django, этот компонент можно установить отдельной библиотекой: новую административную панель, плагин для работы с биометрическими данными или валидатор полей в формах и т.п.

В настоящее время на Django работает достаточно много популярных сервисов, таких как YouTube, Google (для вывода результатов по шаблону), Dropbox, Spotify и другие.

Выбор Django при разработке web-приложения может быть обусловлен, если необходимо собрать большой сайт со множеством возможностей. Для одностраничных сайтов на подобие блога или формы для обратной связи рекомендуется использовать другие фреймворки, например, Flask.

В данном проекте выбор Django обусловлен достаточно широким функционалом, а также необходимостью расширения проекта в дальнейшем.

1.3 Концепция MVT

Концепция Модель – Представление – Шаблон (MVT, от английского Model – View - Template) является основой фреймворка Django. Она разделяет приложение на три основных компонента: модель (Model), представление (View) и шаблон (Template).

Модель – это объект, который представляет данные в приложении и отвечает за их хранение и манипуляцию. Модели Django обычно соответствуют таблицам в базе данных.

Представление – это компонент, который отвечает за обработку запросов и формирование ответов на основе данных из моделей. В Django представления обычно реализуются в виде функций или классов.

Шаблон – это компонент, который отвечает за отображение данных на странице. Шаблоны Django используют язык шаблонов, который позволяет вставлять данные из моделей и представлений в HTML-код.

Отдельно стоит упомянуть маршруты. Маршруты в Django играют роль посредника между пользователем и представлением. Они определяют, какой URL-адрес должен быть связан с каким представлением. Когда пользователь запрашивает URL-адрес, Django использует маршруты для определения соответствующего представления и передачи запроса ему. Представление обрабатывает запрос и формирует ответ, который затем возвращается пользователю.

Схема реализации концепции MVT представлена на рисунке 1.

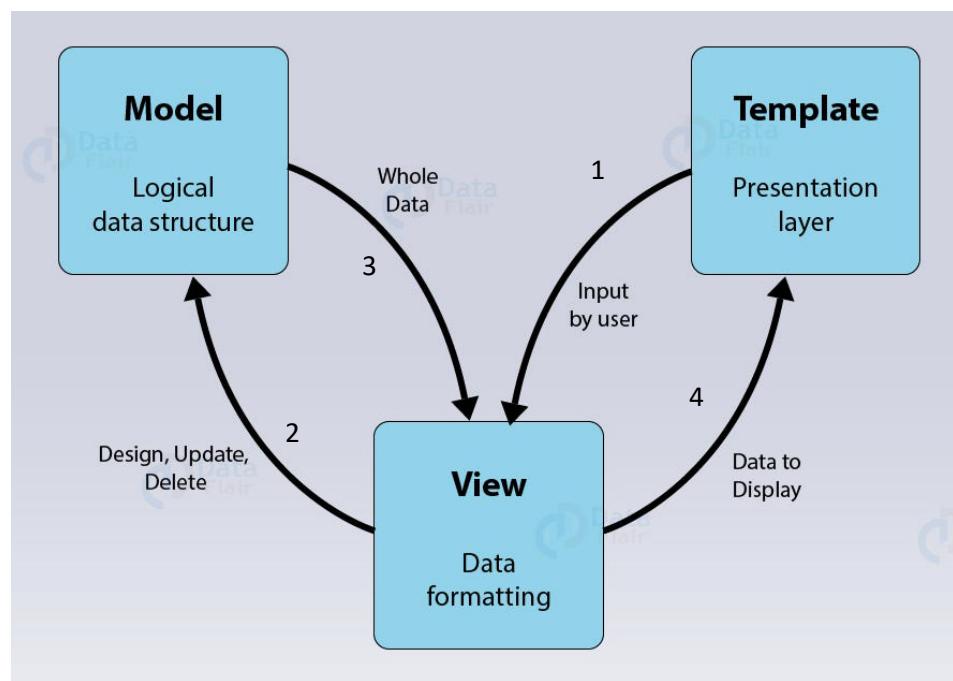


Рисунок 1 – Схема реализации концепции MVT

Глава 2. Подготовка среды и создание проекта

2.1 Виртуальное окружение

Встроенные средства Python позволяют решить проблему управления зависимостями проектов. Если у существует несколько проектов на Python, то вероятность того, что придется работать с разными версиями библиотек или самого Python, очень высока. Но использование автономных виртуальных сред позволяет создавать независимые группы библиотек для каждого проекта, что предотвращает конфликты между версиями и не дает одному проекту повлиять на другой. В Python уже есть модуль `venv` для создания виртуальных сред, который можно использовать как в разработке, так и в производстве.

Проект создавался на платформе Windows 10, с помощью средства разработки Visual Studio Code.

Выбор Visual Studio Code обусловлен следующими факторами:

- множество настроек (как всей программы, так и интерфейса);
- расширяемая библиотека дополнений и готовых решений;
- мультифункциональность (редактор поддерживает почти все языки, используемые для создания приложений);
- простота и гибкость.

Для создания виртуального окружения необходимо создать и перейти в папку проекта. Для осуществления данной операции в терминале Visual Studio Code необходимо ввести следующие строки:

```
mkdir virtual_office  
Cd virtual_office  
python -m venv venv
```

Далее активируем виртуальное окружение, чтобы все дальнейшие действия выполнялись внутри него:

```
.\venv\Scripts\activate
```

2.2 Установка и настройка Django

Для установки Django рекомендуется использовать менеджер пакетов pip.

А также все необходимые дополнения:

```
pip install django
```

Файл requirements.txt содержит информацию обо всех необходимых дополнениях, собранных в процессе разработки проекта.

Помимо самого фреймворка будет установлено несколько обязательных зависимостей. Сам Django содержит в себе около 15 пакетов, позволяющих решать большинство задач разработки без установки дополнительных пакетов и модулей.

В Django различаются термины проект и приложение. Проект представляет из себя пакет Python с базовыми настройками. Приложение также является пакетом. Но он входит в состав проекта. При этом каждый проект может состоять из нескольких приложений. А приложения можно переносить из одного проекта в другой.

2.3 Архитектура проекта

Фреймворк Django в логике своей работы использует компонентную «shared-nothing» архитектуру или архитектуру без общих ресурсов (SN) — это распределенная вычислительная архитектура, в которой каждый запрос на обновление удовлетворяется одним узлом (процессором/памятью/устройством хранения) в кластере компьютеров. Цель состоит в том, чтобы устраниć конфликты между узлами. Узлы не используют одну и ту же память или хранилище независимо друг от друга.

Каждая часть приложения при такой архитектуре независима от других и, следовательно, может быть заменена или изменена, если это необходимо. Чёткое разделение частей означает, что Django может масштабироваться при увеличении трафика, путём добавления оборудования на любом уровне: серверы кеширования, серверы баз данных или серверы приложений.

Django опирается на уже созданную архитектуру и шаблоны взаимодействия компонентов. Учитывая это обстоятельство, а также то, что при реализации проекта используется представления на основе функций, а не классов, необходимость создания uml-диаграммы отсутствует. Общая схема взаимодействия компонентов приложений проекта представлена на рисунке 2.

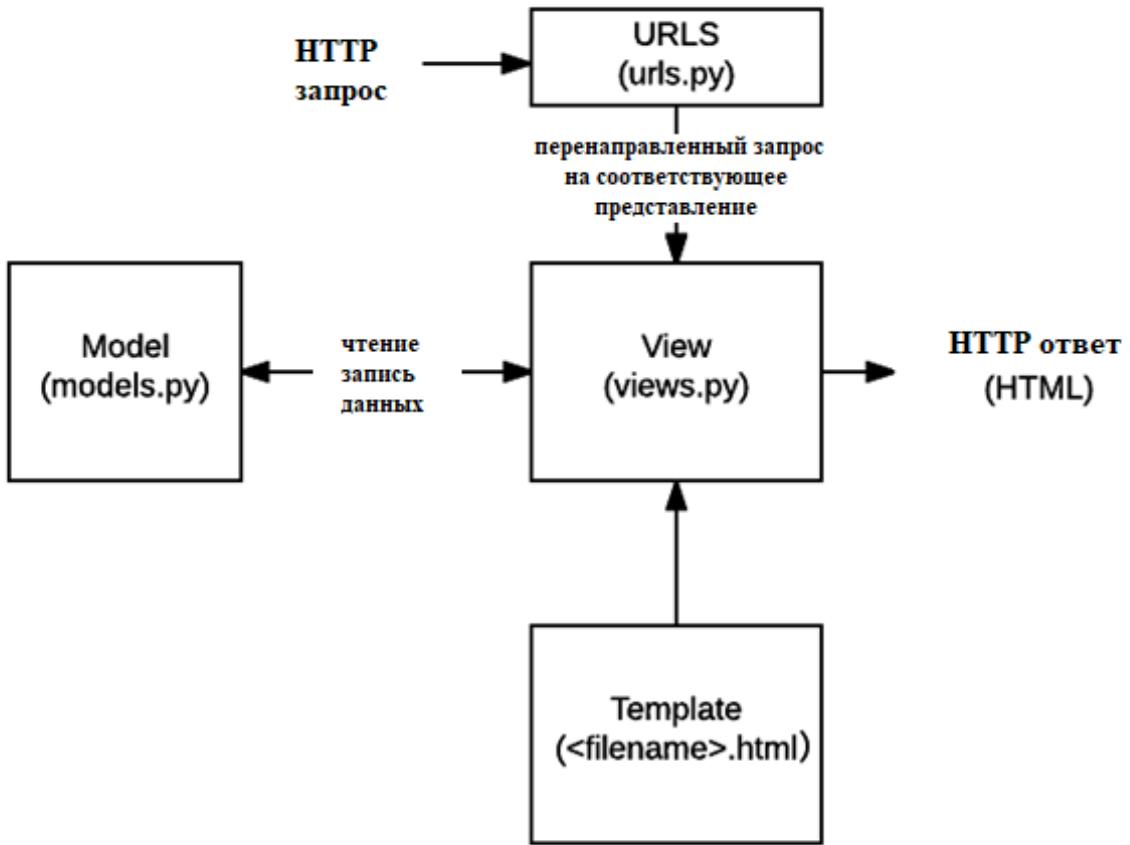


Рисунок 2 – Схема взаимодействия компонентов приложений проекта

Для возможности верной реализации проекта составлена диаграмма UserCase (приложение 1).

Вариант визуализации проекта (UserInterface) представлен в приложении 2.

Диаграмма ERD моделей проекта приведена в приложении 3.

2.4 Создание структуры проекта

Для создания нового проекта в Django необходимо выполнить команду:

```
django-admin startproject virtual_office
```

Эта команда создаст структуру проекта, которая будет содержать все необходимые файлы и папки для работы с фреймворком.

Обзор структуры проекта приведен на рисунке 3.

Каждый из созданных файлов отвечает за определённые действия:

- `manage.py` - файл, который используется для управления проектом. С его помощью можно запустить сервер, создать миграции, создать суперпользователя и т.д.;

- `virtual_office/` - директория, которая содержит основные файлы проекта;

- `__init__.py` - файл, который сообщает Python, что директория `virtual_office` является пакетом;

- `settings.py` - файл, который содержит настройки проекта, такие как базы данных, шаблоны, статические файлы и т.д.

- `urls.py` - файл, который содержит маршруты приложения.

- `asgi.py` - файл, который используется для запуска проекта в ASGI-совместимых серверах.

- `wsgi.py` - файл, который используется для запуска проекта в WSGI-совместимых серверах.

Данный проект предусматривает в своем составе следующие компоненты:

- данные пользователя;
- документы;
- сведения об образовании;
- сведения о собственности;
- контакты;
- планировщик.

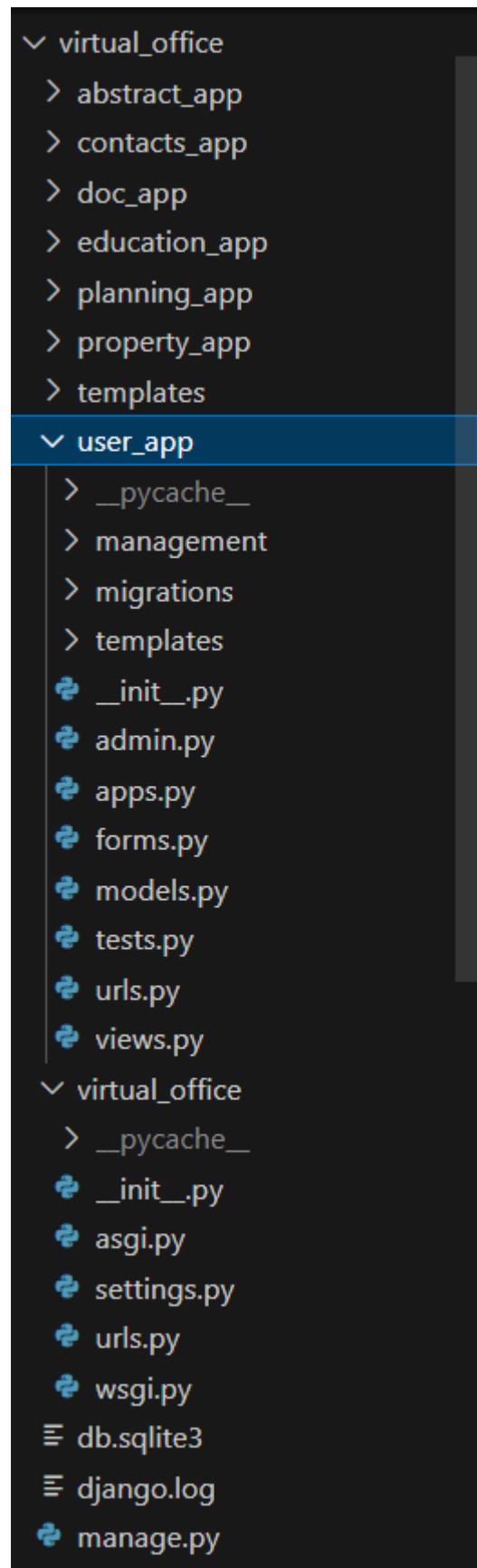


Рисунок 3 – Структура проекта

Для каждого компонента создается отдельное приложение, не зависимое от других. Также необходимо отметить, что для работы данного проекта выбрана следующая структура приложений, представленная на рисунке 1:

- user_app – отвечает за регистрацию, авторизацию пользователя, а также за сбор, изменение, отображение и удаление личных данных пользователя;
- doc_app – отвечает за ввод, изменение, отображение и удаление данных о документах пользователя;
- education_app – отвечает за ввод, изменение, отображение и удаление данных о документах об образовании пользователя;
- property_app – отвечает за ввод, изменение, отображение и удаление данных о собственности пользователя (транспорт и недвижимость);
- contacts_app – отвечает за ввод, изменение, отображение и удаление данных о контактах пользователя;
- planning_app – отвечает за ввод, изменение, отображение и удаление данных о напоминаниях и событиях пользователя.

Стоит отметить, что модель User, создаваемая в приложении user_app, используется для подтверждения аутентификации пользователя.

Приложение abstract_app создано в целях соблюдения принципа DRY (Don't repeat yourself). В нем созданы абстрактные модели, формы, а также функции, константы и переменные, используемые остальными приложениями.

Информация о создании приложений приведена в соответствующем разделе.

2.4 Запуск сервера и проверка работоспособности

Для запуска сервера в каталоге virtual_office выполняем команду:

```
python manage.py runserver
```

Эта команда запустит сервер на локальном хосте и порту 8000.

Первый запуск возвращает следующую информацию в консоль:

```
(venv) PS C:\Alex\Python\diploma\virtual_office> python manage.py runserver 0.0.0.0:8080
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, conte
nntypes, sessions.
Run 'python manage.py migrate' to apply them.
November 01, 2023 - 07:40:09
Django version 4.2.6, using settings 'virtual_office.settings'
Starting development server at http://0.0.0.0:8080/
Quit the server with CTRL-BREAK.
```

Рисунок 4 – Первый запуск сервера

После запуска сервера в браузере по адресу <http://localhost:8000/> отображается информация, представленная на рисунке 5.

В процессе работы над проектом используется режим отладки. В файле настроек settings.py настроены следующий параметр: # SECURITY WARNING: don't run with debug turned on in production! DEBUG = True.

Данный параметр будет отключен при развороте приложения в продакшн.

Для проверки работы приложения на мобильном устройстве, подключенном к той же сети, что и базовый проект, укажем следующий путь для запуска сервера:

```
python manage.py runserver 0.0.0.0:8080
```

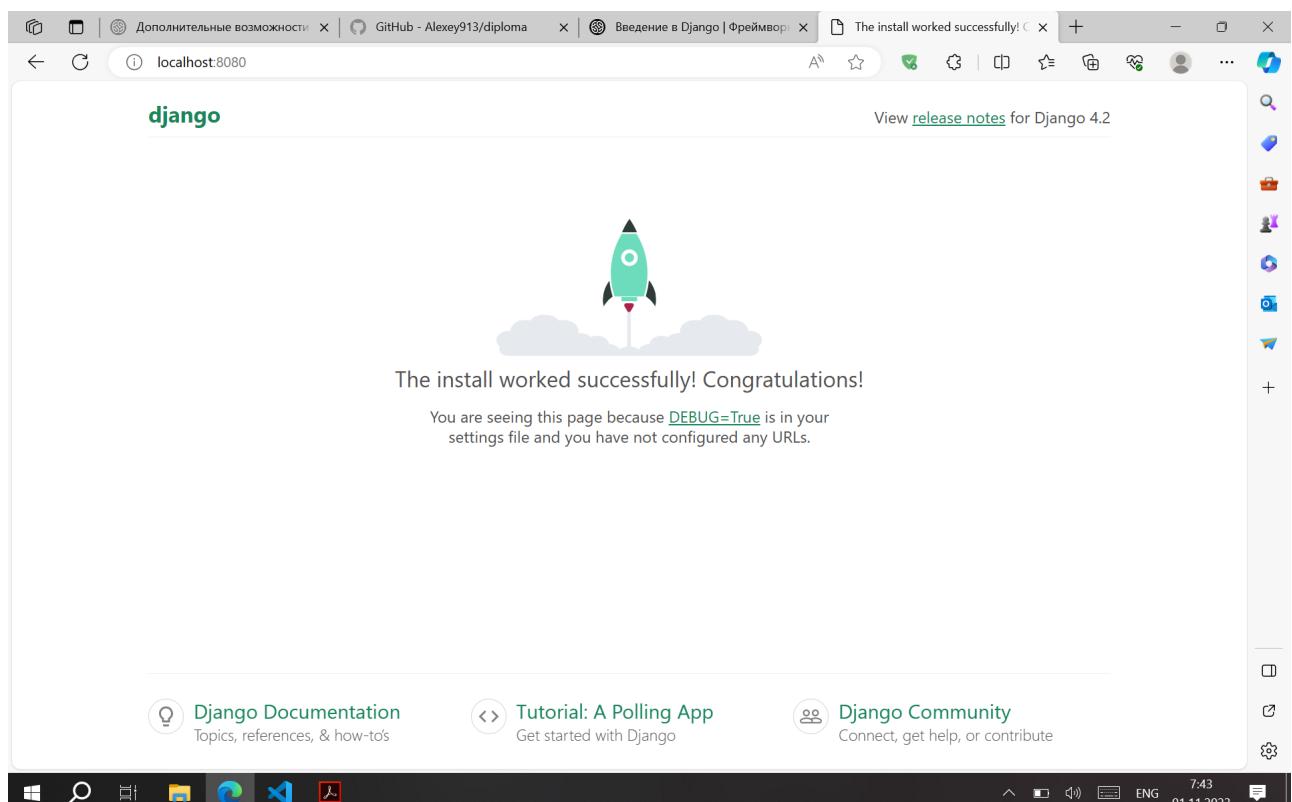


Рисунок 5 – сообщение об удачном запуске сервера

Указав в качестве IP нули, сервер стал прослушивать все адреса. В константу ALLOWED_HOSTS файла settings.py необходимо добавить IP-адрес компьютера, на котором запущен сервис. Теперь, указав в адресной строке браузера мобильного устройства этот IP-адрес, мы можем работать со своим web-приложением.

2.5 Создание приложений, обзор их структуры и добавление в проект

Создание приложения в Django представляет собой создание отдельного модуля, который будет содержать логику и шаблоны для определенной функциональности. Для создания приложения выполняем команду:

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

Здесь <app_name> - название приложений, указанных в п. 2.3.

Структура проекта после создания приложения имеет вид, указанный на рисунке 3.

Здесь приведена структура для приложения user_app:

- user_app/ - директория приложения;
- migrations/ - директория для хранения миграций базы данных;
- __init__.py - файл, указывающий на то, что директория является пакетом Python;
- admin.py - файл для настройки административного интерфейса приложения;
- apps.py - файл для настройки приложения;
- models.py - файл, содержащий модели данных приложения
- tests.py - файл для написания тестов приложения
- views.py - файл, содержащий представления (views) приложения
- db.sqlite - файл базы данных SQLite

В процессе разработки приложения мы создаем ещё несколько файлов и каталогов:

- файл маршрутов (urls.py);

- файл форм (forms.py);
- каталог для файлов шаблонов (templates/);
- каталог для ввода команд из консоли (management);

Чтобы добавить созданное приложение в проект, указываем его в настройках проекта (файл settings.py). Для этого нужно добавить название приложения в список INSTALLED_APPS (рисунок 6).

При этом Django автоматически создает и вносит в список следующие приложения:

- django.contrib.admin - Сайт администратора;
- django.contrib.auth - Система аутентификации;
- django.contrib.contenttypes - Структура для типов контента;
- django.contrib.sessions - Структура сеанса;
- django.contrib.messages - Фреймворк обмена сообщениями;
- django.contrib.staticfiles - Фреймворк для управления статическими файлами.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'abstract_app',
    'user_app',
    'doc_app',
    'property_app',
    'education_app',
    'planning_app',
    'contacts_app',
]
```

Рисунок 6 – Перечень используемых в проекте приложений

2.6 Выбор метода представлений в проекте

Для работы с приложениями необходимо в первую очередь создать представление.

Представление можно осуществлять двумя методами: с помощью функций и с помощью классов.

Представления на основе функций имеют ряд преимуществ по сравнению с классовыми представлениями:

- легкая читаемость кода, понимание и внедрение представлений;
- явный поток кода;
- простое использование декораторов;
- простая реализация специализированного функционала.

К минусам при этом можно отнести:

- избыточность кода и сложность расширения;
- для обработки методов HTTP используется условное ветвление.
- повторение похожего кода.

Представления на основе классов являются альтернативой представлениям, основанным на функциях. Они реализованы в проектах в виде объектов Python, а не функций. Представления, основанные на классах, имеют определенные преимущества по сравнению с представлениями, основанными на функциях:

- возможность наследования (возможность следовать принципу DRY);
- более простая возможность расширения приложений;
- структурирование кода (избавление от операторов условного ветвления внутри представлений на основе функций);
- встроенные универсальные представления (такие как авторизация или регистрация).

Недостатки представлений на основе классов:

- более сложная реализация и затрудненность чтения кода;
- неявный поток кода;

- требуется дополнительный импорт или переопределение метода в декораторах представлений.

Учитывая плюсы и минусы методов представлений, а также опираясь на то, что проект является учебным, для более углубленного понимания языка Python и фреймворка Django в частности, а также для более простой читаемости кода в дипломном проекте используются представления на основе функций.

2.7 Создание представлений

Для создания представления нужно определить функции в файле views.py, которые будет обрабатывать HTTP-запросы.

В качестве примера на рисунке 7 приведено представление авторизации пользователя.

```
def authorization(request):
    if request.method == 'POST':
        form = UserFormAuth(request.POST)
        if form.is_valid():
            data = form.cleaned_data
            phone_email = data['phone_email']
            password = data['password']
            if check_user_pass(phone_email, password):
                return enter_office(request, phone_email, password)
            logger.debug('Неверный логин или пароль')
            messages.error(request, "Неверный логин или пароль")
        else:
            form = UserFormAuth()
    context = {'form': form,
               'title': 'Авторизация',
               'user_id': None,
               'menu': menu}
    return render(request, 'user_app/authorization.html', context=context)
```

Рисунок 7 – Представление авторизации

Следует отметить, что фреймворк Django имеет встроенный модуль авторизации и регистрации. Однако в целях углубленного изучения работы Django и функционала работы Django с сессиями в проекте реализована пользовательская система регистрации и авторизации, с собственными

валидаторами форм, а также реализована возможность ввода на выбор пользователя e-mail или номера телефона при регистрации и авторизации.

Для настройки URL в Django определены маршруты (routes), которые связывают определенные URL с соответствующими представлениями (views) в приложении. Маршруты определяются в файле urls.py, который находится в корневой директории проекта и в директории приложения.

2.8 Определение маршрутов

Для корректной работы представлений необходимо произвести создание маршрутов в файле urls.py.

При создании проекта файл urls.py в корневой директории проекта уже содержит комментарий с описанием трёх способов добавления маршрутов:

- функции представления;
- классы представления;
- подключение других файлов с настройками URL.

Мы используем представления на основе функций, в связи с чем используем маршруты, приведенные на рисунке 8.

Здесь импортируется модуль admin из пакета django.contrib. Он будет необходим при создании административной панели и работы с ней.

Далее импортируется функция path и модуль include из пакета django.urls. Обе функции нужны для формирования url адресов, на которые будет отвечать сервер. Создается список urlpatterns, который будет содержать маршруты (routes) для обработки URL-адресов.

Модуль pageNotFound необходим для передачи ошибки 404. Он вынесен в отдельное приложение abstract_app, которое содержит все абстрактные модели, формы, функции, константы и переменные, используемые двумя и более приложениями

```
from django.contrib import admin
from django.urls import path, include

from abstract_app.views import pageNotFound

urlpatterns = [
    path('admin/', admin.site.urls),
    path('doc/', include('doc_app.urls')),
    path('', include('user_app.urls')),
    path('property/', include('property_app.urls')),
    path('education/', include('education_app.urls')),
    path('reminds/', include('planning_app.urls')),
    path('contacts/', include('contacts_app.urls')),
]
```

Рисунок 8 – Содержимое файла urls.py корневой директории

Теперь необходимо добавить маршрут для административной панели Django, который будет обрабатывать URL-адрес, начинающийся с префикса "admin/" и передавать управление в модуль admin.site.urls (установлен по умолчанию).

Далее добавляем маршруты для приложений doc, property, education, reminds, contacts, которые будут обрабатывать соответствующие URL-адреса и передавать управление, соответственно, в модули doc_app.urls, property_app, education_app, planning_app, reminds_app, contacts_app. Маршруты включаются с помощью функции include.

Пустой запрос будет направлять в основное приложение проекта – user_app.urls.

Для корректной работы проекты в каждой из соответствующих директориях приложений создается файл urls.py, где прописываются маршруты, необходимые для связывания с представлениями.

Теперь каждое приложение имеет собственный urls.py с локальными маршрутами. А urls.py проекта включает их в глобальный список адресов с помощью include().

Пример содержимого файла urls.py приведен на рисунке 9.

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

urlpatterns = [path('registration/', views.registration, name='registration'),
    path('authorization/', views.authorization, name='authorization'),
    path('logout/', views.logout, name='logout'),
    path('data/<int:user_id>', views.show_data, name='data'),
    path('change_data/<int:user_id>', views.change_data, name='change_data'),
    path('', views.main, name='main')
]
```

Рисунок 9 – Содержимое файла urls.py для приложения user_app.

В данном примере мы определили шесть маршрутов. Функция path() принимает три аргумента: первый аргумент - это URL-адрес, второй - это представление, которое будет обрабатывать запрос на этот URL, а также заданы имена маршрутов с помощью параметра name.

При вводе адреса в браузер сервер начинает сопоставлять введённый путь с содержимым списка urlpatterns из urls.py проекта. Когда совпадение найдено, оно отбрасывается и оставшееся часть пути передаётся в urls.py указанный внутри функции include, например, doc_app.urls.py. Он также просматривает список urlpatterns в поисках совпадений уже в своем файле urls.py. Далее вызывается представление, обозначенное в path.

Таким образом, настройка URL в Django позволяет определить маршруты для обработки запросов на определенные URL и связать их с соответствующими представлениями.

В целом, создание приложения и определение представлений — это основа работы с Django. Далее ведется работа по добавлению моделей, форм, роутинга и других компонентов для создания полноценного web-приложения.

2.9 Логирование в проекте

Логирование — это процесс записи информации о работе приложения. Обычно это запись в файлы. Хотя можно писать логи и в базы данных, и просто выводить в консоль. Логи позволяют отслеживать работу приложения, выявлять ошибки и проблемы, а также анализировать производительность приложения.

В Django логирование используется для записи информации о запросах, ошибках, отладочной информации и т.д. Логи могут быть полезными для разработчиков, чтобы быстро находить и исправлять ошибки, а также для администраторов, чтобы отслеживать работу приложения и производительность.

Django использует модуль `logging` из стандартной библиотеки Python.

Для настройки логирования в Django необходимо изменить файл `settings.py`. В этом файле определяются параметры логирования, такие как уровень логирования, формат вывода, место хранения логов и т.д. Конфигурация логирования в файле `settings.py` в Django происходит через словарь `LOGGING`. На рисунке 10 приведено содержимое словаря `LOGGING` проекта.

Параметры словаря `LOGGING`:

- `version`: версия формата конфигурации логирования. В настоящее время используется версия 1;
- `disable_existing_loggers`: значение `False`, указывает на то, что существующие логгеры будут продолжать работать;
- `handlers`: определяет, какие обработчики будут использоваться для записи логов. Обработчики могут быть консольными или файловыми. В проекте использованы оба вида обработчиков. Логи сохраняются в файл `virtual_office_logger.log`;
- `loggers`: определяет, какие логгеры будут использоваться для записи логов. Логгеры могут быть определены для фреймворка Django (вывод в консоль) в целом и для конкретных приложений (вывод в консоль и файл).

Для каждого обработчика и логгера можно указать следующие параметры:

- `class`: класс, который используется для записи логов. В нашем проекте мы используем классы `StreamHandler` и `FileHandler` для записи логов в консоль и файл соответственно;
- `filename`: путь к файлу, в который записываются логи - `virtual_office_logger.log` в корневой директории проекта;

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'verbose': {
            'format': '{levelname} {asctime} {module} {process} {thread} {message}',
            'style': '{}',
        },
        'simple': {
            'format': '%(levelname)s %(message)s'
        },
    },
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
            'formatter': 'verbose',
        },
        'file': {
            'class': 'logging.FileHandler',
            'filename': 'virtual_office_logger.log',
            'encoding': 'utf-8',
            'formatter': 'verbose',
        },
    },
    'loggers': {
        'django': {
            'handlers': ['console'],
            'level': 'INFO',
        },
        'user_app': {
            'handlers': ['console', 'file'],
            'level': 'DEBUG',
            'propagate': False,
        },
        'doc_app': {
            'handlers': ['console', 'file'],
            'level': 'DEBUG',
            'propagate': False,
        },
        'property_app': {
            'handlers': ['console', 'file'],
            'level': 'DEBUG',
            'propagate': False,
        },
        'education_app': {
            'handlers': ['console', 'file'],
            'level': 'DEBUG',
            'propagate': False,
        },
        'reminds_app': {
            'handlers': ['console', 'file'],
            'level': 'DEBUG',
            'propagate': False,
        },
        'contacts_app': [
            'handlers': ['console', 'file'],
            'level': 'DEBUG',
            'propagate': False,
        ],
    },
}
```

Рисунок 10 – Словарь LOGGING

- propagate: значение False означает, что сообщения не будут передаваться родительским логгерам.

После запуска сервера логирование работает в рамках Django. Перейдите по существующим адресам. Теперь загляните в файл django.log и сравните его содержимое с информацией в консоли.

Для использования логирования в Django необходимо импортировать модуль logging и создать объект логгера (рисунок 11).

Теперь мы видим логи уровня приложения и в консоли, и в файле.

Логирование позволяет нам отслеживать работу приложения и находить ошибки. Мы можем задавать различные уровни логирования для разных частей приложения и выбирать, какие сообщения будут записываться в файл или выводиться на консоль. Если происходит ошибка, мы можем быстро найти ее в логах и исправить.

Таким образом, логирование является важным инструментом для отладки и анализа работы приложения, поэтому рекомендуется использовать его в своих проектах.

```
import logging

logger = logging.getLogger(__name__)

def registration(request):
    if request.method == 'POST':
        form = UserFormReg(request.POST)
        if form.is_valid():
            data = form.cleaned_data
            phone_email = data['phone_email']
            password = data['password']
            hash_password, salt = pass_to_hash(password)
            if not get_user(phone_email):
                user = User(phone=fill_phone(phone_email),
                            email=fill_email(phone_email),
                            salt=salt,
                            hash_password=hash_password)
                user.save()
                create_data(user)
                logger.info(f'Создан пользователь {user}')
                messages.success(request, "Пользователь успешно создан!")
                return enter_office(request, phone_email, password)
            else:
                logger.debug('Ошибка создания пользователя - пользователь существует')
                messages.error(request, "Пользователь с таким e-mail или телефоном уже существует!")
        else:
            form = UserFormReg()
    context = {'form': form,
               'title': 'Регистрация',
               'user_id': None,
               'menu': menu}
    return render(request, 'user_app/registration.html', context=context)
```

Рисунок 11 – Импорт, объявление и использование логирования

Глава 3. Организация работы с моделями проекта

3.1 Общие сведения о моделях в Django

Одной из ключевых функций фреймворка Django являются модели, которые обеспечивают удобный способ работы с базами данных.

Модели в Django — это классы Python, которые определяют структуру таблиц базы данных. Каждый атрибут класса соответствует полю таблицы, а экземпляры класса представляют записи в таблице. Django использует ORM (Object-Relational Mapping), чтобы автоматически создавать SQL-запросы для работы с базой данных.

Модели в Django обеспечивают удобный способ работы с базами данных. Они позволяют создавать, изменять и удалять записи в базе данных без необходимости написания SQL-запросов. Кроме того, модели позволяют описывать отношения между таблицами, что делает работу с базой данных еще более удобной и эффективной.

ERD-диаграмма, определяющая взаимодействие сущностей баз данных проекта приведена в приложении 3.

3.2 Определение моделей

Для создания модели в Django в файле models.py приложения (в качестве примера представлен файл models.py приложения user_app) класс Python User, который наследуется от базового класса models.Model.

Учитывая, что в проекте используются модели с общими полями (например, поля «Номер» и «Серия» документа для паспорта, ИИН, водительского удостоверения и т.п., или поля «Фамилия» и «Имя» для пользователя, супруга или детей в паспорте), целесообразно создать отдельное приложение abstract_app, где в файле models.py разместить необходимые

абстрактные модели. Так как всё в Python является объектом, с помощью вложенного класса Meta класса Model, мы имеем возможность наследовать поля абстрактной модели в текущую модель (рисунки 12 и 13).

```
from django.db import models

from .views import (module) models

class AbstractUser(models.Model):

    class Meta:
        abstract = True

    email = models.EmailField(null=True, blank=True)
    phone = models.IntegerField(null=True, blank=True)
    hash_password = models.BinaryField()
    salt = models.BinaryField()

class People(models.Model):

    class Meta:
        abstract = True

    surname = models.CharField(
        max_length=50, null=True, blank=True, verbose_name='Фамилия')
    name = models.CharField(max_length=50, null=True,
                           blank=True, verbose_name='Имя')
    patronymic = models.CharField(
        max_length=50, null=True, blank=True, verbose_name='Отчество')
    birthday = models.DateField(
        null=True, blank=True, verbose_name='Дата рождения')
    gender = models.CharField(max_length=7, null=True,
                             blank=True, choices=GENDERS, verbose_name='Пол')
```

Рисунок 12 – Абстрактные модели в приложении abstract_app

Рассмотрим создание модели на примере класса Data приложения user_app.

Сначала необходимо определить несколько полей абстрактной модели People, от которой будет наследоваться класс Data: «surname», «name», «patronymic», «birthday», «gender»; также прописываем их свойства (см. рисунок 12):

- CharField – поле для хранения строк, указываем обязательный параметр – максимальную длину строки с помощью аргумента max_length;
- DateField – поле для хранения даты.

```

from django.db import models

from abstract_app.models import People, AbstractUser

class User(AbstractUser):

    def __str__(self):
        return f'{self.phone if self.phone else self.email}'

    class Meta:
        db_table = "user_app_user"

class Data(People):
    birth_place = models.CharField(max_length=100, null=True, blank=True, verbose_name='Место рождения')
    place_residense = models.CharField(max_length=100, null=True, blank=True, verbose_name='Место жительства')
    user = models.OneToOneField(
        'User', on_delete=models.CASCADE, primary_key=True)

    def __str__(self):
        attrs = vars(self)
        return ', '.join("%s: %s" % item for item in attrs.items())

    class Meta:
        db_table = "user_app_data"

```

Рисунок 13 – Создание модели в приложении user_app

Для всех полей залаем параметры blank=True (означает, что поле не является обязательным к заполнению), null=True (поле может содержать значение Null (None)), а также verbose_name, определяющий название поля при отображении в форме.

Для поля gender определяем параметр choices=GENDERS, означающий, что поле может принимать только значения из списка GENDERS, созданного в файле views.py приложения abstract_app, а также то, что при загрузке данных в форму будет отображаться выпадающий список из состава списка GENDERS.

С помощью вложенного класса Meta определяем параметр abstract=True, означающий, что мы не можем создавать экземпляры данной модели, она является абстрактной.

Кроме указанных типов полей, в проекте используются следующие типы:

- EmailField – поле для хранения электронных адресов. Django проверяет, что введенный адрес имеет правильный формат;
- IntegerField – поле для хранения целых чисел;
- TextField – поле для хранения текстовых данных большой длины.
- BooleanField – поле для хранения логических значений (True/False);
- TimeField – поле для хранения времени;

- ForeignKey – поле для связи с другой моделью;
- ManyToManyField – поле для связи с другой моделью в отношении «многие-ко-многим»;
- OneToOneField – поле для связи с другой моделью в отношении «один-к-одному»;
- DecimalField – поле для хранения десятичных чисел.

Стоит отметить несколько особенностей указанных типов полей:

- для использования в моделях Django поля ImageField необходимо установить дополнительный модуль Pillow:

```
pip install Pillow
```

- для работы с полями OneToOne используется параметр primary_key=True при связке с моделью User. Это связано со спецификой приложения (один пользователь может иметь только один паспорт или ИНН) и продиктовано необходимостью для удобства работы с моделями, так как в данном случае ID записи в конкретной модели будет равен ID пользователя модели User.

После создания абстрактной модели необходимо в соответствующем файле models.py (в данном случае это приложение user_app) создать еще одну модель, унаследовав ее от абстрактной (см. рисунок 13).

В модель People добавляется несколько полей, в том числе, поле user (OneToOne). Связка моделей будет осуществляться в представлениях из данных, полученных при авторизации пользователя.

В классе Meta также определяем необязательный параметр db_table='user_app_data', где обозначаем название таблицы в базе данных.

Класс __str__ необходим для корректного вывода данных класса Data.

3.3 Миграции

В Django миграции представляют собой автоматически сгенерированные скрипты для изменения структуры базы данных в соответствии с изменениями в

моделях приложения. Миграции позволяют легко и безопасно изменять базу данных, не теряя данные.

Чтобы создать миграции для модели Data и User, мы должны выполнить следующую команду:

```
python manage.py makemigrations user_app
```

Команда **makemigrations** анализирует все изменения в моделях приложения и создает миграционный файл, который содержит инструкции для изменения базы данных.

При этом если опустить имя приложения, команда **python manage.py makemigrations** попытается найти изменения во всех приложениях проекта и создать миграции для каждого из них.

После выполнения команды в каталоге `/migrations` приложения `user_app` появляется новый файл вида `0001_initial.py`. Содержимое файла сгенерировано автоматически и не требует ручных правок в данном проекте.

В базу данных автоматически добавлено поле `id` как первичный ключ.

Создание миграций не изменяет таблицы базы данных. Это лишь подготовка к изменениям.

После создания миграционного файла мы должны применить его к базе данных с помощью команды `migrate`:

```
python manage.py migrate
```

Команда `migrate` выполняет все накопленные миграции в порядке их создания и применяет изменения к базе данных.

База данных при этом наполняется таблицами, включая `data_user_app` и `user_user_app`, которые мы создали ранее.

В дальнейшем при каждом изменении модели приложений создаем новые миграции для этих изменений. Затем применяем их к базе данных, и мигрируем в базу. Это позволяет обновлять базу данных без потери данных и сохранять целостность базы данных.

3.4 Создание собственных команд manage.py

В фреймворке Django имеется возможность по созданию собственных команд. Они необходимы при тестировании работы с моделями данных без создания представлений, добавления маршрутов и отрисовки шаблонов.

Чтобы создать собственную команду, необходимо:

- создать структуру каталогов: в каталоге приложения необходимо создать следующие вложенные друг в друга каталоги: management/ и commands/ (рисунок 14);

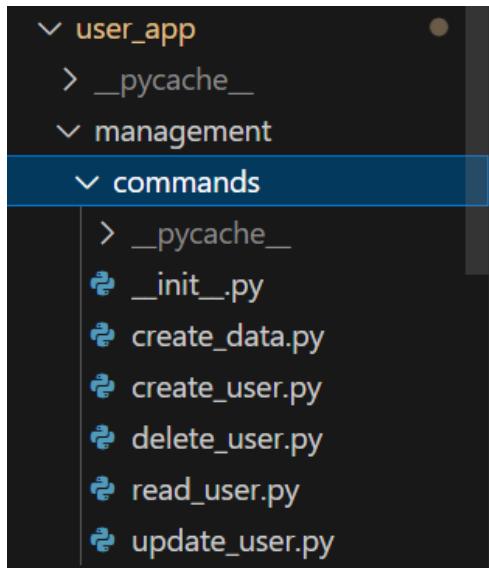


Рисунок 14 – Структура каталога management

- создать файл с кодом команды с помощью класса Command(BaseCommand): для примера здесь приведен файл команды заполнения базы данных категорий для водительского удостоверения, которая заполняется единожды и в дальнейшем не изменяется (рисунок 15).

Переменная help выведет справку по работе команды. А метод handle отработает при вызове команды в консоли.

Для печати информации в стандартный поток вывода – консоль вместо print необходимо использовать self.stdout.write.

Далее вызываем файл fill_driver_category_db из терминала командой:

```
python manage.py fill_driver_category_db
```

Команда соответствует названию файла папки management.

```

from django.core.management.base import BaseCommand

from doc_app.models import DriverCategory
from virtual_office.common_data import DRIVER_CATEGORIES


class Command(BaseCommand):
    help = "Fill database 'driver_category'"

    def handle(self, *args, **kwargs):
        for cat, des in DRIVER_CATEGORIES.items():
            category = DriverCategory(name=cat, description=des)
            category.save()
            self.stdout.write(f'{category}')

```

Рисунок 15 – Скрипт команды заполнения БД водительских категорий

В результате мы получаем заполненную базу данных категорий для водительского удостоверения из списка DRIVER_CATEGORIES в файле view.py приложения abstract_app.

3.5 Работа с данными в моделях

В Django работа с данными в моделях осуществляется через объекты моделей, которые представляют отдельные записи в базе данных. При работе с данными в базах данных используются принципы CRUD - акронима, обозначающий четыре базовые функции, используемые при работе с базами данных: создание (create), чтение (read), модификация (update), удаление (delete).

Все эти операции рассмотрим на основе примера из проекта - модель User из приложения user_app.

Создание объекта реализуется через функцию представления registration (рисунок 16).

Для создания нового объекта модели необходимо создать экземпляр класса модели и заполнить его поля значениями. Практически все объекты в данном проекте создаются с помощью форм и данных, которые вводит пользователь. Более подробно о формах изложено в главе 5.

```

80  def registration(request):
81      if request.method == 'POST':
82          form = UserFormReg(request.POST)
83          if form.is_valid():
84              data = form.cleaned_data
85              phone_email = data['phone_email']
86              password = data['password']
87              hash_password, salt = pass_to_hash(password)
88              if not get_user(phone_email):
89                  user = User(phone=fill_phone(phone_email),
90                             email=fill_email(phone_email),
91                             salt=salt,
92                             hash_password=hash_password)
93                  user.save()
94                  create_data(user)
95                  logger.info(f'Создан пользователь {user}')
96                  messages.success(request, "Пользователь успешно создан!")
97                  return enter_office(request, phone_email, password)
98              else:
99                  logger.debug('Ошибка создания пользователя - пользователь существует')
100                 messages.error(request, "Пользователь с таким e-mail или телефоном уже существует!")
101         else:
102             form = UserFormReg()
103         context = {'form': form,
104                    'title': 'Регистрация',
105                    'user_id': None,
106                    'menu': menu}
107     return render(request, 'user_app/registration.html', context=context)

```

Рисунок 16 – Функция представления add_realty (создание объекта)

После ввода данных в форму и проверки их встроенными и пользовательскими валидаторами, мы получаем данные для создания объекта модели. На рисунке 16 это строки 85 – 93. Для получения хэш-значения пароля и соли используется функция `pass_to_hash`, а для заполнения полей телефона или e-mail функции `fill_phone` и `fill_email`, соответственно.

Мы создаем новый объект модели "User" с заданными значениями полей телефона, e-mail и пароля и сохраняем его в базе данных с помощью метода `save()`.

Для получения объектов модели из базы данных можно использовать методы `all()` и `get()` класса модели. Метод "`all()`" возвращает все объекты модели, а метод "`get()`" возвращает один объект, соответствующий заданным условиям (рисунок 17).

На рисунке 17 показано чтение данных из модели Data. Мы выводим только поля, указанные в методе `values()`, поиск при этом осуществляется по `id` пользователя, который в нашем случае равен `id` Data (связь OneToOne).

При этом для сокращения кода в шаблонах используется метод `get_data_with_verbose_name` (более подробно в главе 4).

```
def show_data(request, user_id):
    data = Data.objects.values('surname', 'name', 'patronymic', 'birthday',
                               'birth_place', 'place_residense', 'gender').filter(user_id=user_id).first()
    object_data = get_object_or_404(Data, user_id=user_id)
    context = get_data_with_verbose_name(object_data, data)
    user = object_data.user
    if user.email: context['E-mail'] = user.email
    if user.phone: context['Телефон'] = user.phone
    context_to_template = {'context': context,
                          'title': 'Данные пользователя',
                          'user_id': user_id,
                          'menu': menu}
    return render(request, 'user_app/data.html', context=context_to_template)
```

Рисунок 17 – Чтение данных модели

Однако, во избежание ошибки `user_app.models.User.DoesNotExist: User matching query does not exist`, принято решение использовать функцию `filter` в сочетании с методом `first()` или `all()`. Вместе с тем, уходим от использования `id` для имени переменной в базе данных во избежание конфликта со встроенной в Python функцией `id()`, и используем при чтении по `id` параметр `pk` – primary key, первичный ключ в таблице, т.е. ID.

При этом если в базе несколько записей, вернётся одна, первая из результата запроса, а если запись одна, `first` вернёт эту единственную запись. При отсутствии совпадений, вместо ошибки возвращается `None`.

Также с помощью функции `filter` в приложении `planning_app` при поиске событий по дате используется фильтрация по дате (рисунок 18), используя двойное подчёркивание и оператор `date`.

```
@check_authorization
def reminds(request, user_id):
    today = datetime.today()
    reminds = Remind.objects.filter(user_id=user_id,
                                    date__year=today.year,
                                    date__month=today.month,
                                    date__day=today.day).all()
    mes = 'На сегодня событий нет'
    return get_reminds(request, user_id, reminds, mes)
```

Рисунок 18 – Фильтрация событий по дате (сегодняшний день)

Для изменения объектов модели также использовался метод filter() в сочетании с save() экземпляра класса модели. Пример приведен на рисунке 19.

```
@check_authorization
def change_data(request, user_id):
    if request.method == 'POST':
        form = ChangeDataForm(request.POST)
        if form.is_valid():
            current_data = Data.objects.get(user_id=user_id)
            current_user = User.objects.get(pk=user_id)
            if current_data:
                data_by_form = form.cleaned_data
                for field, value in data_by_form.items():
                    if value and field != 'phone' and field != 'email':
                        setattr(current_data, field, value)
                current_data.save()
                phone = data_by_form['phone']
                email = data_by_form['email']
                if phone:
                    if get_user(phone):
                        messages.error(request, "Номер телефона зарегистрирован другим пользователем")
                    else:
                        current_user.phone = phone
                if email:
                    if get_user(email):
                        messages.error(request, "E-mail зарегистрирован другим пользователем")
                    else:
                        current_user.email = email
                current_user.save()
            else:
                form.save()
            logger.info(f'Успешное сохранение данных пользователя {user_id}')
            messages.success(request, "Данные успешно изменены")
            return redirect(show_data, user_id=user_id)
            messages.error(request, "Неверные данные")
            logger.warning(f'Ошибка ввода данных пользователя {user_id}')
        else:
            form = ChangeDataForm()
            context = {'form': form,
                       'title': 'Изменение личных данных',
                       'user_id': user_id,
                       'user_name': Data.objects.get(user_id=user_id),
                       'menu': menu}
            return render(request, 'user_app/change_data.html', context=context)
```

Рисунок 19 – Изменение данных пользователя с применением метода save()

После ввода данных через форму функция change_data проверяет, не передано ли пустое поле форме. Если оно пустое, то в соответствующее поле модели изменения не вносятся. Если же поле формы корректно заполнено, то значение меняется на новое.

Все изменения сохраняются с помощью метода save().

Удаление осуществляется не для всех сущностей. В проекте можно удалить супруга из паспорта, записи о детях, контакты, напоминания и т.п. При

необходимости возможно добавить удаление документов. На данном этапе проекта этого не требуется.

Удаление записей выполняется с помощью функции filter() и метода delete() (рисунок 20).

Часто удаление данных из базы не является хорошим решением. Такие данные можно восстановить только из резервной копии, а это затратная операция. Для упрощения таких ситуаций целесообразно помечать данные удалёнными. Но учитывая учебный статус проекта, а также в целях углубления знаний о методах CRUD в проекте использовано полноценное удаление записей.

```
@check_authorization
def del_spouse(request, user_id):
    try:
        Spouse.objects.filter(passport_id=user_id).delete()
        logger.info(f"Удалены данные о супруге пользователя {user_id}")
        messages.success(request, "Данные успешно удалены")
    except:
        logger.debug(
            f"Ошибка удаления данных о супруге пользователя {user_id}")
        messages.error(request, "Данных не существует")
    finally:
        return redirect('passport', user_id=user_id)
```

Рисунок 20 – Пример реализации удаления сущности

Глава 4. Работа с шаблонами на основе представлений

4.1 Сопоставление представления с маршрутом

Для работы web-приложения необходимо определить шаблоны, в которых будут отображаться формы и выводиться данные для пользователя. Они представляют собой HTML-страницы, связанные с представлениями.

Использование представлений осуществляется путем указания их имени в URL-адресе приложения. Далее подключаются маршруты приложения к маршрутам проекта.

Преобразования пути — это процесс преобразования URL-адреса запроса в формат, понятный для Django. Django преобразует пути запроса в параметры, которые передаются обработчику. При этом мы можем передавать в URL-запросе параметры, необходимые для работы представлений.

Например, практически все функции представлений получают в urls-запросе id пользователя, после чего с помощью декоратора `check_authorization` осуществляется проверка данных пользователя и осуществляется переход к нужному представлению с параметром `user_id`.

Для этого в маршрутах после символа ‘/’ прописывается значение `<int:user_id>` (рисунок 21).

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

urlpatterns = [
    path('<int:user_id>/', views.docs, name='docs'),

    path('passport/<int:user_id>', views.get_passport, name='passport'),
    path('change_passport/<int:user_id>', views.change_passport, name='change_passport'),

    path('edit_spouse/<int:user_id>', views.edit_spouse, name='edit_spouse'),
    path('del_spouse/<int:user_id>', views.del_spouse, name='del_spouse'),
```

Рисунок 21 – Маршруты с передачей параметров

Если необходимо передать несколько параметров, используется форма, приведённая на рисунке 22.

```
path('change_children/<int:user_id>/<int:children_id>', views.change_children, name='change_children'),
path('del_children/<int:user_id>/<int:children_id>', views.del_children, name='del_children'),
```

Рисунок 22 – Маршруты с передачей нескольких параметров

Когда пользователь делает запрос к веб-приложению, Django преобразует URL-адреса запроса в параметры, которые передаются обработчику (представлению)

Таким образом, диспетчер URL и преобразования пути являются важными компонентами фреймворка Django, которые позволяют обрабатывать запросы и направлять их на соответствующие обработчики.

4.2 Шаблоны

Шаблоны в Django представляют собой файлы, содержащие HTML-код с дополнительными тегами и переменными, которые могут быть заменены на значения из контекста.

Шаблоны в Django используются для генерации HTML-страниц на основе данных, полученных из представлений. Они позволяют создавать динамические страницы, которые могут меняться в зависимости от данных, полученных от пользователя или из базы данных.

Для корректной работы шаблонов необходима правильная структура каталогов.

Внутри каталога приложения создан каталог templates. Далее в нём создаётся каталог с именем приложения (рисунок 23).

В каждом приложении необходим каталог templates. При этом в корневой директории проекта также создан каталог templates, в котором размещен базовый шаблон base.html, от которого наследуются либо базовые шаблоны приложений (при наличии), либо остальные шаблоны приложений.

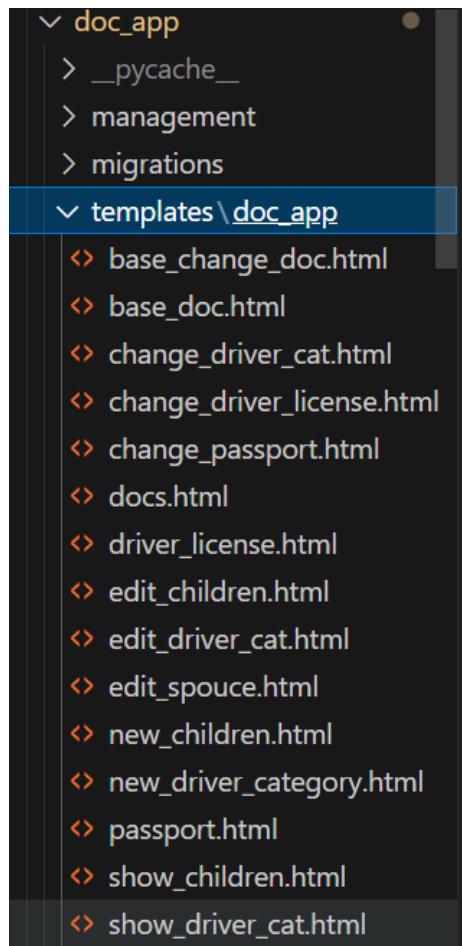


Рисунок 23 – Структура каталогов шаблонов

Внутри шаблонов проекта используются переменные из контекста, а также конструкции цикла `{% for %}` и условия `{% if %}`. Переменные внутри шаблона заключаются в двойные фигурные скобки с пробелами до и после имени переменной.

Чтобы передать данные в шаблон, используется функция `render` из модуля `django.shortcuts` (рисунок 24).

Функция `render` заменяет переменные в шаблоне, передаваемые с помощью словаря `context`, на значения из контекста и возвращает готовую HTML-страницу.

Во фреймворке Django используются условные операторы в шаблонах имеют синтаксис похожий на Python и заключаются в фигурные скобки вида `{% %}`.

Кроме того, шаблоны Django поддерживают сложные условия благодаря конструкциям `{% elif %}` и `{% else %}`. Например, при выводе информации о

событиях, если на сегодняшнюю дату они отсутствуют, шаблон выдаст соответствующее сообщение (рисунок 25).

```
def get_reminds(request, user_id, reminds, mes):
    context = {'date_dict': show_calendar(),
               'user_id': user_id,
               'reminds': reminds,
               'mes': mes,
               'menu': menu}
    return render(request, 'planning_app/reminds.html', context=context)

@check_authorization
def reminds(request, user_id):
    today = datetime.today()
    reminds = Remind.objects.filter(user_id=user_id,
                                    date__year=today.year,
                                    date__month=today.month,
                                    date__day=today.day).all()
    mes = 'На сегодня событий нет'
    return get_reminds(request, user_id, reminds, mes)
```

Рисунок 24 – Передача переменных в шаблон reminds.html (get_reminds)

В Django для вывода данных в цикле используется тег for. Он позволяет перебирать элементы списка, словаря или QuerySet'a и выводить их на страницу. Как и для условия if, цикл for завершается оператором {%- endfor %}.

```
{% if reminds %}
    {% for remind in reminds %}
        {% if not forloop.last %}
            <li><a href="{% url 'show_remind' user_id remind.pk %}">{{ remind.title }}</a></li>
        {% else %}
            <li class="last"><a href="{% url 'show_remind' user_id remind.pk %}">{{ remind.title }}</a></li>
        {% endif %}
    {% endfor %}
    {% else %}
        <h2>{{ mes }}</h2>
    {% endif %}
```

Рисунок 25 – Использование переменных, условий и циклов в шаблоне

4.3 Наследование шаблонов

В Django есть возможность наследовать шаблоны, что позволяет создавать базовый шаблон и на его основе создавать другие, которые будут иметь общий вид и функциональность. Наследование шаблонов позволяет избежать

дублирования кода и упростить процесс разработки. Мы перестаём нарушать принцип DRY.

В базовом шаблоне создаем следующие блоки:

- {`% block title %`} – ответственный за отображение заголовка;
- {`% block mini_title %`} – отвечает за отображение подзаголовка,
- {`% block login %`} – отвечает за отображение кнопки «Выйти», если пользователь авторизован и кнопок «Авторизация», «Регистрация», если пользователь не авторизован;
- {`% block menu %`} – ответственный за отображение меню;
- {`% block messages %`} – в нем отображаются flash-сообщения, если они передаются в представлении;
- {`% block content %`} – основное наполнение страницы;
- {`% block footer %`} – отображение футера сайта.

Часть этих блоков остается неизменными для всех страниц, часть меняется, в зависимости от наполнения.

Для наследования шаблонов используется тег `extends` для указания базового шаблона. Затем мы переопределяем необходимы блоки (рисунок 26).

Фреймворк Django позволяет создать базовый шаблон на уровне проекта, который размещен в корневой директории проекта в каталоге `templates`.

Для создания базового шаблона проекта необходимо внести изменения в файл `settings.py`, показанные на рисунке 27.

Добавляем в список `DIRS` путь до каталога шаблона проекта - `BASE_DIR / 'templates'`. Далее создаём каталог `templates` в каталоге `BASE_DIR`. Это каталог верхнего уровня. В нём находится файл `manage.py`, каталог проекта и каталоги приложений. В каталог помещаем базовый шаблон приложения `base.html`.

Теперь команда расширения в дочерних шаблонах записывается без указания имени приложения: `{% extends 'base.html' %}`

Однако в случае наследования из шаблона приложения необходимо прописать адрес типа `{% extends 'doc_app/base_doc.html' %}`

The screenshot shows two code editor windows side-by-side.

base.html (Top Window):

```

1 <% block content %>
2     <h2>Наполнение страницы</h2>
3     <% endblock %>
4
5     </td>
6     </tr>
7     <tr border="1" class="gradient_footer" height="20%">
8         <td colspan="3">
9             <% block footer %>
10            <h3 align = "right">Информация</h3>
11            <% endblock %>
12        </td>
13    </tr>
14
15    </table>
16
17    </div>
18
19    <br>
20
21
22
23
24
25
26
27
28
29
30
31

```

show_property.html (Bottom Window):

```

1 <% extends 'base.html' %>
2 <% block content %>
3
4
5     <div><h1>{{ title }}</h1></div>
6     <div>
7         <table cellpadding="3">
8             <% for pr in property %>
9                 <tr>
10                    <td align="left" width="200">
11                        <a href="{% url change_property user_id pr.pk %}">
12                            {{ pr.type_property }}:
13                            {% if pr.area %}
14                                площадь {{ pr.area }} кв.м.
15                            {% else %}
16                                {{ pr.brand }}&ampnbsp{{ pr.model }}
17                            {% endif %}
18
19                        </a>
20                    </td>
21                </tr>
22            <% endfor %>
23            <tr>
24                <td>
25                    <a href="{% url add_property user_id %}"> Добавить данные документа об образовании </a>
26                </td>
27            </tr>
28        </table>
29        <br>
30    </div>
31

```

Рисунок 26 – `{% block content %}` в базовом и наследуемом шаблонах

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR/'templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Рисунок 27 – Настройка settings.py для создания базового шаблона проекта

Глава 5. Работа с формами

5.1 Создание форм

Формы в Django — это инструмент, который позволяет создавать и обрабатывать HTML-формы. Формы используются для сбора данных от пользователей и отправки их на сервер для дальнейшей обработки.

Плюсы использования форм в Django заключаются в том, что они позволяют упростить процесс сбора данных от пользователей и обработки их на сервере.

Формы автоматически проверяют данные на корректность, что позволяет избежать ошибок при обработке данных. Кроме того, формы могут быть переиспользованы в разных частях приложения, что упрощает разработку и поддержку кода.

В целом, использование форм в Django — это удобный и эффективный способ сбора и обработки данных от пользователей, который позволяет упростить разработку и поддержку web-приложений. Однако, при использовании форм необходимо учитывать их ограничения и принимать меры для защиты от возможных уязвимостей.

Определение формы в Django начинается с создания класса формы, который наследуется от класса `forms.Form` или `forms.ModelForm`.

Для создания форм авторизации и регистрации использованы формы, наследуемые от класса `forms.Form`.

В рамках реализации проекта принято решение создать собственные модули авторизации и регистрации, в целях углубленного изучения работы Django и функционала работы Django с сессиями, а также реализации возможности ввода на выбор пользователя e-mail или номера телефона при регистрации и авторизации.

В классе `forms.Form` определяются поля формы, их типы, валидация и другие атрибуты. Поля имеют текстовый формат, а также созданы валидаторы

проверки ввода номера телефона или e-mail, соответствия пароля определенному формату, а также соответствия подтверждения пароля. (рисунок 28).

```
22 class UserFormAuth(forms.Form):
23     phone_email = forms.CharField(max_length=50, min_length=6, label='Телефон / E-mail',
24                                     error_messages={'required': 'Обязательно введите телефон или e-mail'},
25                                     widget=forms.TextInput(attrs={
26                                         'class': 'form-control', 'placeholder': 'Телефон или e-mail'}))
27
28     password = forms.CharField(max_length=50, min_length=6, label='Пароль',
29                                widget=forms.PasswordInput(attrs=[
30                                         'class': 'form-control', 'placeholder': 'Пароль']))
31
32     def clean_phone_email(self):
33         phone_email: str = self.cleaned_data['phone_email']
34         if check_phone_email(phone_email):
35             raise forms.ValidationError(
36                 'Введите корректный номер телефона или адрес электронной почты')
37         return phone_email
38
39     def clean_password(self):
40         password: str = self.cleaned_data['password']
41         if not any(map(str.isdigit, password)) or \
42             not any(map(lambda x: x in low_alpha, password)) or \
43             not any(map(lambda x: x in up_alpha, password)):
44             raise forms.ValidationError(
45                 'Пароль должен содержать цифры, а также заглавные и строчные латинские буквы')
46         return password
```

Рисунок 28 – Создание UserFormAuth формы и валидаторов полей

В данной форме создается класс UserFormAuth, который наследуется от класса forms.Form. В классе определены два поля формы: phone_email и password. Каждое поле представлено отдельным классом, который определяет тип поля и его атрибуты:

- поле phone_email определено с помощью класса CharField, который представляет текстовое поле. Аргумент max_length указывает максимальную длину текста, которую можно ввести в поле:

- поле password определено также с помощью класса CharField добавлен аргумент min_length, указывающий минимальную длину вводимого текста.

Оба поля имеют атрибут label, устанавливающий отображение название поля в шаблоне, а также атрибут widget, необходимый для настройки отображения полей.

По сути, каждое поле формы — это экземпляр класса <Name>Field из модуля forms, где <Name> — имя класса поля.

В данном проекте используются следующие классы Field:

- CharField — используется для создания текстовых полей, таких как имя,

- фамилия, электронная почта и т.д.;
- EmailField – используется для создания поля электронной почты;
 - IntegerField – используется для создания поля для ввода целых чисел;
 - FloatField – используется для создания поля для ввода чисел с плавающей точкой;
 - BooleanField – используется для создания поля флажка;
 - DateField – используется для создания поля даты;
 - TimeField – используется для создания поля времени;
 - ChoiceField – используется для создания выпадающего списка с выбором одного из нескольких вариантов;
 - CheckboxSelectMultiple – используется для выбора одного или нескольких вариантов;

Кроме класса forms.Form в проекте используются формы на основе классов forms.ModelForm. Его использование связано с тем, что весь проект работает на основе базы данных и большая часть форм близко соответствуют моделям. Данный класс позволяет создавать форму из модели (рисунок 29).

С помощью класса Meta мы задаем параметр fields – поля, которые необходимо заполнить. Остальные поля, если они необходимы (например, user_id) заполняются в представлении.

Здесь же с помощью параметра widgets мы добавляем необходимые виджеты полей.

Наследование форм в проекте выполняется аналогично наследованию в моделях с небольшими дополнениями (рисунок 30).

Для того, чтобы наследовать форму также необходимо наследовать вложенный класс Meta и использовать его атрибуты для задания атрибутов дочерней формы (представлены, как объединение списков fields и словарей widgets).

Виджеты форм определяют внешний вид и поведение полей формы на странице. В проекте использованы следующие виджеты:

- TextInput – используется для создания текстового поля ввода;

- EmailInput – используется для создания поля ввода электронной почты;
- PasswordInput – используется для создания поля ввода пароля;
- CheckboxInput – используется для создания флажка;
- DateInput – используется для создания поля ввода даты;
- TimeInput – используется для создания поля ввода времени;
- Textarea – используется для создания многострочного текстового поля ввода.

```

from abstract_app.views import REPEAT_LIST

class Remind(models.Model):
    title = models.CharField(max_length=20, verbose_name='Заголовок', default=f'Напоминание')
    date = models.DateField(verbose_name='Дата', default=now)
    time = models.TimeField(verbose_name='Время', default=now)
    all_day = models.BooleanField(verbose_name='Весь день', default=False)
    repeat = models.CharField(max_length=20, choices=sorted(REPEAT_LIST),
                              verbose_name='Повтор', default='Никогда')
    description = models.TextField(null=True, blank=True, verbose_name='Описание события')
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    repeat_id = models.IntegerField(null=True)

✓ from django import forms

    from .models import Remind

✓ class RemindForm(forms.ModelForm):

    class Meta:
        model = Remind

        fields = ['title', 'date', 'time', 'all_day', 'repeat', 'description']

        widgets = {
            'title': forms.TextInput(attrs={'class': 'form-control', 'placeholder': 'Заголовок'}),
            'date': forms.DateInput(attrs={'class': 'form-control', 'type': 'date'}),
            'time': forms.TimeInput(attrs={'class': 'form-control', 'type': 'time'}),
            'all_day': forms.CheckboxInput(attrs={'class': 'form-check-input'}),
            'description': forms.Textarea(
                attrs={'class': 'form-control', 'placeholder': 'Описание события'}),
        }

```

Рисунок 29 – Модель Reminds и созданная на ее основе форма

В каждом виджете задан атрибут class со значениями form-control, form-check-input, что позволяет использовать стили Bootstrap для оформления полей формы. Атрибут placeholder, который определяет текст-подсказку, отображаемый в поле формы до того, как пользователь введет данные.

```

from django import forms

from .models import Property, Transport, Realty

class PropertyForm(forms.ModelForm):

    class Meta:
        model = Property

        fields = ['date_registration', 'description']

        widgets = {
            'date_registration': forms.DateInput(attrs={
                'class': 'form-control', 'type': 'date'}),
            'description': forms.Textarea(attrs={
                'class': 'form-control', 'placeholder': 'Описание'}),
        }

    class RealtyForm(PropertyForm):
        class Meta(PropertyForm.Meta):
            model = Realty

            fields = ['type_property', 'cadastral_number', 'cadastral_cost',
                      'adress', 'area'] + PropertyForm.Meta.fields

            widget_list = {
                'cadastral_number': forms.TextInput(attrs={
                    'class': 'form-control', 'placeholder': 'Кадастровый номер'}),
                'cadastral_cost': forms.TextInput(attrs={
                    'class': 'form-control', 'placeholder': 'Кадастровая стоимость'}),
                'adress': forms.TextInput(attrs={
                    'class': 'form-control', 'placeholder': 'Адрес объекта'}),
                'area': forms.TextInput(attrs={
                    'class': 'form-control', 'placeholder': 'Площадь объекта'}),
            }

            widgets = PropertyForm.Meta.widgets
            for k, v in widget_list.items():
                widgets[k] = v

```

Рисунок 30 – Пример наследования полей

Для ввода даты в виджетах используется параметр 'type': 'date', благодаря которому поле позволяет выбрать дату из календаря.

При вводе данных формы для создания события определен дополнительный параметр initial=datetime.date.today, оставляющий по умолчанию сегодняшнюю дату при заполнении формы.

5.2 Представление и маршруты для форм

Следующий этап работы с формами – использовать представление для перевода формы из класса в видимый пользователем HTML, а также для обработки данных, которые пользователь введёт в форму и отправит на сервер.

Для вывода формы по GET запросу и обработки данных по POST запросу в проекте использован код по примеру на рисунке 31.

```
@check_authorization
def change_realty(request: HttpResponse, user_id: int, property_id: int) -> Callable:
    return change_property(request, user_id, property_id,
                           RealtyForm, Realty, 'realty',
                           'объекте недвижимости')

def change_property(request: HttpResponse, user_id: int, property_id: int,
                    type_form: forms.ModelForm, entity: models.Model,
                    kind:str, in_title: str) -> HttpResponse:
    current_property = entity.objects.filter(pk=property_id).first()
    if request.method == 'POST':
        form = type_form(request.POST)
        if form.is_valid():
            data_by_form = form.cleaned_data
            for field, value in data_by_form.items():
                if value:
                    setattr(current_property, field, value)
            current_property.save()
            logger.info(f"Изменение данных о {in_title} {property_id} \
пользователя {user_id}")
            messages.success(request, "Данные успешно изменены")
            return redirect('get_'+kind, user_id=user_id, property_id=property_id)
        logger.debug(
            f"Ошибка сохранения данных о {in_title} {property_id} \
пользователя {user_id}")
            messages.error(request, "Неверные данные")
    else:
        form = type_form()
    context = {'form': form,
               'title': f'Изменение данных о {in_title}',
               'user_id': user_id,
               'property_id': property_id,
               'change_property': 'change_'+kind,
               'del_property': 'del_'+kind,
               'property': current_property,
               'menu': menu}
    return render(request, 'property_app/edit_property.html', context)
```

Рисунок 31 – Представление для формы изменения данных о недвижимости

В данном случае мы импортируем модуль render для рендеринга шаблона, а также импортируем форму RealtyForm. Далее определяем функцию change_property, которая будет обрабатывать запросы на адрес /change_realty/<int:user_id>/<int:property_id>/, а также /change_transport/<int:user_id>/<int:property_id>/.

Здесь следует отметить, что в связи с использованием однотипных объектов в проекте использованы общие функции, принимающие в себя классы моделей и форм, которые вызываются конкретными функциями представлений (например, для объектов недвижимости или для документов).

Если запрос пришел методом POST, то мы создаем экземпляр формы RealtyForm с переданными данными из запроса. Если форма проходит валидацию (все поля заполнены корректно), то мы получаем данные из формы и можем с ними работать.

Если запрос пришел методом GET, то мы просто создаем пустой экземпляр формы RealtyForm и передаем его в шаблон edit_property.html.

В шаблоне edit_property.html мы выводим нашу форму с помощью тега {{ form }} и добавляем кнопку «Изменить данные». Необходимо также добавить тег формы, указав что при нажатии кнопки отправить нужно использовать тот же url адрес для отправки данных и метод POST для пересылки.

При этом для выравнивания формы мы используем ее вывод с помощью таблицы, выводя в одном столбце название полей, в другом сами поля. Также добавляем под каждым полем вывод сообщения об ошибке валидации, если они присутствуют. Пример реализации такого вывода приведен на рисунке 32.

```

<form action="{% url change_property user_id property.pk %}" method="post">
    {% csrf_token %}
    <table cellpadding="3">
        {% for f in form %}
            <tr>
                <td align="left" width="200">
                    <label for="{{ f.id_for_label }}>{{f.label}}: </label>
                </td>
                <td> {{ f }} </td>
            </tr>
            <tr>
                <td colspan="2" align="right">
                    <font color="FFAA00">{{ f.errors }}</font>
                </td>
            </tr>
        {% endfor %}
    </table>
    <br>
    <input class="c-button" type="submit" value="Изменить данные">

```

Рисунок 32 – Шаблон для вывода формы ReltyForm

Так же был добавлен тег `csrf_token`. Он обеспечивает защиту данных формы от изменений злоумышленниками. Так называемая защита от CSRF атак.

5.3 Валидация данных форм

При заполнении форм во избежание ошибок пользователя предусмотрена встроенная и пользовательская валидация данных.

Встроенная валидация осуществляется в процессе передачи данных из браузера на сервер. Это проверка соответствия введенных данных типу поля, соответствия длины введенной строки и т.п.

Пользовательская валидация данных осуществляется с помощью метода `clean()`. В качестве примера на рисунке 28 приведены методы `clean_phone_email` и `clean_password`.

После прохождения встроенной валидации поле `phone_email` проверяется на наличие только цифр и знака «+» во введенных данных или на соответствие данных шаблону `mail@mail.ru`.

Если условие ввода не соблюдено, то выбрасывается исключение ValidationError с соответствующим сообщением под полем с ошибкой в соответствии с шаблоном (см. рисунок 32).

5.4 Сохранение данных форм в базу данных

После того как форма заполнена пользователем, отправлена Django, проверена и прошла валидацию данные можно использовать. Использование данных из форм в проекте реализует сохранение или изменение записей в базе данных.

Сохранение данных при изменении записей в базе данных приведено на рисунке 33.

Сохранение данных при создании новой записи в базе данных с помощью формы, наследованной от класса forms.Forms представлено на рисунке 34.

Сохранение данных при создании новой записи в базе данных с помощью формы, наследованной от класса forms.ModelForms представлено на рисунке 35.

```
@check_authorization
def change_diploma(request: HttpResponse, user_id: int, diploma_id: int) -> HttpResponse:
    current_diploma = Diploma.objects.filter(pk=diploma_id).first()
    if request.method == 'POST':
        form = DiplomaForm(request.POST)
        if form.is_valid():
            data_by_form = form.cleaned_data
            for field, value in data_by_form.items():
                if value:
                    setattr(current_diploma, field, value)
            current_diploma.save()
            logger.info(f"Изменение данных о дипломе {diploma_id} пользователя {user_id}")
            messages.success(request, "Данные успешно изменены")
            return redirect('education', user_id=user_id)
        logger.debug(
            f"Ошибка сохранения данных диплома {diploma_id} пользователя {user_id}")
        messages.error(request, "Неверные данные")
    else:
        form = DiplomaForm()
    context = {'form': form,
               'title': 'Изменение данных об образовании',
               'user_id': user_id,
               'diploma': current_diploma,
               'menu': menu}
    return render(request, 'education app/edit diploma.html', context)
```

Рисунок 33 – Сохранение данных при изменении данных

```

@check_authorization
def change_data(request, user_id):
    if request.method == 'POST':
        form = ChangeDataForm(request.POST)
        if form.is_valid():
            current_data = Data.objects.get(user_id=user_id)
            current_user = User.objects.get(pk=user_id)
            if current_data:
                data_by_form = form.cleaned_data
                for field, value in data_by_form.items():
                    if value and field != 'phone' and field != 'email':
                        setattr(current_data, field, value)
                current_data.save()
                phone = data_by_form['phone']
                email = data_by_form['email']
                if phone:
                    if get_user(phone):
                        messages.error(request, "Номер телефона зарегистрирован другим пользователем")
                    else:
                        current_user.phone = phone
                if email:
                    if get_user(email):
                        messages.error(request, "E-mail зарегистрирован другим пользователем")
                    else:
                        current_user.email = email
                current_user.save()
            else:
                form.save()
            logger.info(f'Успешное сохранение данных пользователя {user_id}')
            messages.success(request, "Данные успешно изменены")
            return redirect(show_data, user_id=user_id)
        messages.error(request, "Неверные данные")
        logger.warning(f'Ошибка ввода данных пользователя {user_id}')
    else:
        form = ChangeDataForm()
    context = {'form': form,
               'title': 'Изменение личных данных',
               'user_id': user_id,
               'user_name': Data.objects.get(user_id=user_id),
               'menu': menu}
    return render(request, 'user_app/change_data.html', context=context)

```

Рисунок 34 – Сохранение данных (форма forms.Forms)

```

@check_authorization
def add_driver_category(request, user_id):
    if request.method == 'POST':
        form = DriverCategoryAddForm(request.POST)
        if form.is_valid():
            category = form.save(commit=False)
            category.driver_license_id = user_id
            category.save()

```

Рисунок 35 – Сохранение данных (форма forms.ModelForms)

Если форма отправлена как POST запрос и проверки данных прошли успешно, мы получаем переданные данные и создаём экземпляр класса модели либо изменяем данные уже существующего экземпляра. Метод save() сохраняет запись в таблицу базы данных.

Глава 6. Административная панель и Django Debug Toolbar

6.1 Работа с административной панелью Django

Административная панель в Django — это готовый интерфейс, который позволяет управлять данными вашего приложения. Она предоставляет возможность создавать, редактировать и удалять записи в базе данных, а также осуществлять множество других действий, связанных с управлением приложением.

Адрес, по которому мы можем зайти в админ панель автоматически настроен при создании проекта.

Для изменения языка панели в файле settings.py изменена языковая константа с английского на русский: LANGUAGE_CODE = 'ru-ru' (рисунок 36).

```
# Internationalization
# https://docs.djangoproject.com/en/4.2/topics/i18n/

LANGUAGE_CODE = 'ru'
```

Рисунок 36 – Изменение языковой константы

По умолчанию доступ к административной панели имеет только суперпользователь (superuser). Выполним в командной строке команду:

```
python manage.py createsuperuser
```

Далее введём имя пользователя, электронную почту, пароль и повтор пароля (рисунок 37).

```
• Имя пользователя (leave blank to use 'al913'): Alex
• Адрес электронной почты: al913@mail.ru
Password:
Password (again):
```

Рисунок 37 – Ввод данных суперпользователя

В базе данных в таблице auth_user появились новые данные (рисунок 38).

<code>id</code>	<code>INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT</code>	<code>password</code>	<code>varchar(128) NOT NULL</code>	<code>last_login</code>	<code>datetime</code>	<code>is_superuser</code>	<code>bool NOT NULL</code>	<code>username</code>	<code>varchar(150) NOT NULL UNIQUE</code>	<code>last_name</code>	<code>varchar(150) NOT NULL</code>	<code>email</code>	<code>va NOT</code>
1		<code>pbkdf2_sha256\$6000...</code>	<code>NULL</code>			1		Alex				<code>a1913@mail.ru</code>	

Рисунок 38 – Данные суперпользователя в базе данных

Теперь, перейдя по адресу `http://localhost:8000/admin/`, и введя имя пользователя и пароль суперпользователя (рисунок 39) мы можем войти в административную панель проекта (рисунок 40).

Рисунок 39 – Вход в административную панель

Рисунок 40 – Главная страница административной панели

Раздел “Пользователи и группы” доступен по умолчанию. Администратор сайта может делать всю цепочку CRUD: создавать, просматривать, изменять и удалять с группами и пользователями.

CRUD доступен для любых моделей, которые зарегистрированы в административной панели.

Административная панель позволяет, помимо операций CRUD с моделями, осуществлять следующие действия:

- добавление пользовательских моделей в административную панель;
- подключение моделей из разных приложений;
- персонализация моделей (настройка отображаемой информации);
- изменение существующих моделей;
- сортировка строк баз данных;
- текстовый поиск;
- добавление новых действий с записями в базе данных.

6.2 Панель Django Debug Toolbar

В комплекте Django нет встроенного инструмента для профилирования кода. Но фреймворк позволяет подключать пакеты, созданные другими разработчиками. И в качестве пакета для профилирования обычно используют Django Debug Toolbar.

Django Debug Toolbar — это инструмент для отладки Django-приложений, который предоставляет дополнительную информацию о запросах и ответах, используя различные панели, которые можно настроить. Он может быть полезен для оптимизации производительности, отслеживания ошибок и улучшения качества кода.

Основные возможности Django Debug Toolbar:

- панель запросов: отображает время выполнения каждого запроса к базе данных, а также общее количество запросов;
- панель шаблонов: показывает, какие шаблоны были использованы для генерации страницы и сколько времени заняло их выполнение;
- панель SQL: отображает все SQL-запросы, выполненные приложением, включая параметры запросов;
- панель кэша: показывает, какие запросы были сохранены в кэше и сколько времени они там находились;
- панель профилирования: предоставляет информацию о времени выполнения каждой функции приложения;

- панель HTTP-заголовков: отображает HTTP-заголовки запросов и ответов;
- панель среды выполнения: показывает информацию о системе, на которой работает приложение;
- панель логирования: позволяет просматривать сообщения журнала приложения.

Кроме того, Django Debug Toolbar (далее – DjDT) также позволяет создавать свои собственные панели, чтобы отслеживать дополнительную информацию, которая может быть полезна для вашего приложения.

Для установки DjDT воспользуемся компонентом PIP для установки. Выполним команду:

```
pip install django-debug-toolbar
```

Для корректной работы DjDT необходимо проверить файл настроек settings.py и заполнить четыре параметра, влияющие на DjDT:

- django.contrib.staticfiles включён в список INSTALLED_APPS;
- в качестве значения для ключа BACKEND в списке TEMPLATES установлен django.template.backends.django.DjangoTemplates;
- в качестве значения для ключа APP_DIRS в списке TEMPLATES установлена истина – True;
- задано значение константы STATIC_URL. В проекте это каталог static/.

Далее необходимо внести в файл настроек settings.py следующие данные:

- в список INTERNAL_IPS добавим локальный адрес компьютера – 127.0.0.1;

Панель отображается только в случае совпадения адреса.

- в список INSTALLED_APPS добавим DjDT как приложение debug_toolbar;
- в список MIDDLEWARE добавим промежуточный слой debug_toolbar.middleware.DebugToolbarMiddleware

Далее в файле urls.py проекта добавляем следующий маршрут:

```
path('__debug__/', include("debug_toolbar.urls")).
```

Теперь при запуске проекта в браузере отображается панель DjDT, с помощью которой проведена оптимизация кода и работа по уменьшению времени работы приложений (рисунок 41).

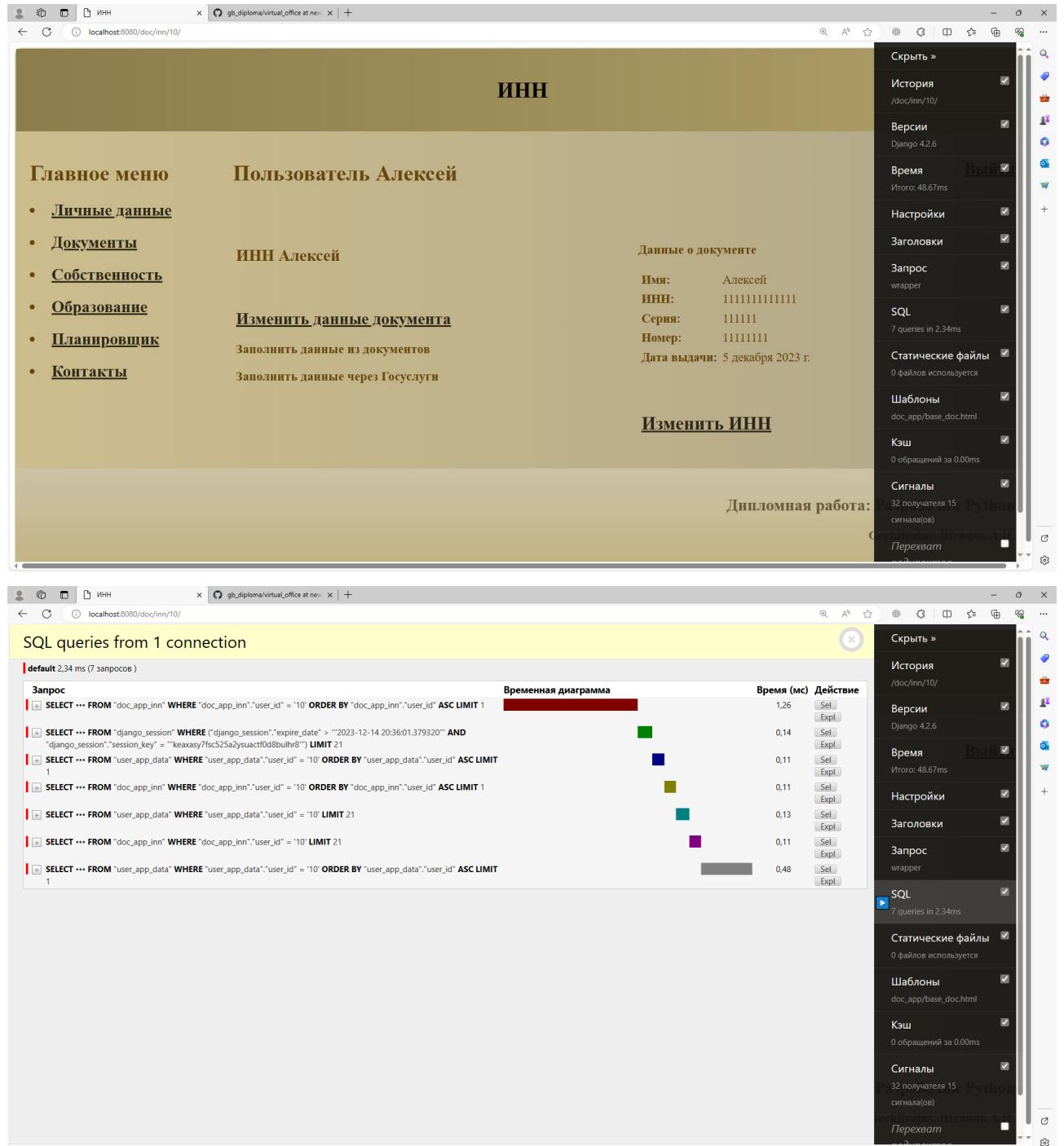


Рисунок 41 – Панель Django Debug Toolbar

Глава 7. Развёртывание проекта

7.1 Настройки безопасности

В качестве площадки выбрана платформа <https://www.pythonanywhere.com/> (рисунок 42).

The screenshot shows the PythonAnywhere dashboard. At the top, there's a logo for python anywhere by ANACONDA, a navigation bar with links to Dashboard, Consoles, Files, Web, Tasks, and Databases, and a welcome message "Welcome, al913". Below the header, there are sections for CPU Usage (2% used), File storage (26% full), and recent activity.

- Recent Consoles:** Shows a list of recent consoles with buttons for Bash, Python, and More...
 - You have no recent consoles.
- Recent Files:** Shows a list of recent files with preview links:
 - /home/al913/django_server/myproject...base.html
 - /home/al913/django_server/myproject...client_orders.html
 - /home/al913/django_server/myproject...settings.py
 - /home/al913/django_server/myproject...success.html
 - /home/al913/django_server/myproject...views.py
- Recent Notebooks:** Shows a message: "Your account does not support Jupyter Notebooks. Upgrade your account to get access!" with a link to upgrade.
- All Web apps:** Shows a message: "You don't have any web apps." with a button to Open Web tab.

Рисунок 42 – Главная страница сервиса <https://www.pythonanywhere.com/>

Для настройки безопасности в первую очередь выключаем режим отладки в файле settings.py:

DEBUG = False

Здесь же добавляем две константы и секретный ключ. Так повышается безопасность работы с сессиями и с csrf токенами:

SESSION_COOKIE_SECURE = True

CSRF_COOKIE_SECURE = True

Секретный ключ безопаснее хранить не в файле настроек, а в переменных окружения. Поэтому строку с ключом от Django необходимо заменить на строки:

```
import os
```

```
SECRET_KEY = os.getenv('SECRET_KEY')
```

Секретный ключ будет сгенерирован и добавлен в переменные окружения на следующих этапах.

Также добавляем адрес сайта в список доступных хостов и ещё одну константу для правильной настройки работы со статическими файлами на сервере (рисунок 43).

```
SECRET_KEY = os.getenv('SECRET_KEY')

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = False

SESSION_COOKIE_SECURE = True
|
CSRF_COOKIE_SECURE = True

ALLOWED_HOSTS = ['0.0.0.0',
                 'localhost',
                 '192.168.0.108',
                 '192.168.0.13',
                 'al913.pythonanywhere.com',]

INTERNAL_IPS = ['127.0.0.1',]

STATIC_URL = 'static/'
STATIC_ROOT = BASE_DIR / 'static/'
```

Рисунок 43 – Редактирование файла settings.py

7.2 Настройки подключения к базе данных

Для подключения базы данных заходим на страницу Базы данных на сайте pythonanywhere. В проекте используем базу данных MySQL.

После ввода пароля по умолчанию создана база al913\$default. Кликаем по имени, чтобы открыть консоль MySQL. Вводим команду смены кодировки с латиницы по умолчанию, на UTF-8:

```
ALTER DATABASE al913$default CHARACTER SET utf8 COLLATE
utf8_general_ci;
```

После выполнения команды выключаем консоль командой exit (рисунок 44).

The screenshot shows a terminal window titled "MySQL: al913\$default". It displays the MySQL monitor startup information, including the server version (8.0.33 Source distribution) and copyright notice. A command is run to alter the database character set to utf8_general_ci, followed by the "exit" command to close the connection. The terminal window has a standard browser-like header with tabs and sharing options.

```
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 70641843
Server version: 8.0.33 Source distribution

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> ALTER DATABASE al913$default CHARACTER SET utf8 COLLATE utf8_general_ci;
Query OK, 1 row affected, 2 warnings (0.01 sec)

mysql> exit
Bye

Console closed.
```

Рисунок 44 – Смена кодировки базы данных

Далее в файле settings.py необходимо настроить подключение к MySQL. Настройки приведены на рисунке 45.

The screenshot shows a code editor displaying the contents of the settings.py file. The code defines a 'DATABASES' dictionary containing a single entry for the 'default' database. This entry specifies the engine as 'django.db.backends.mysql', the name as 'al913\$default', the user as 'al913', and the host as 'al913.mysql.pythonanywhere-services.com'. The 'OPTIONS' key is set to a dictionary with two items: 'init_command' (containing the MySQL command to set NAMES to utf8mb4 and sql_mode to STRICT_TRANS_TABLES) and 'charset' (set to utf8mb4). The code uses Python's f-string notation for the 'HOST' value.

```
DATABASES = [
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'al913$default',
        'USER': 'al913',
        'PASSWORD': os.getenv('MYSQL_PASSWORD'),
        'HOST': 'al913.mysql.pythonanywhere-services.com',
        'OPTIONS': {
            'init_command': "SET NAMES 'utf8mb4';SET sql_mode='STRICT_TRANS_TABLES'",
            'charset': 'utf8mb4',
        },
    }
]
```

Рисунок 45 – Настройки базы данных

Данные об именах пользователя, базы данных и хосте взяты на странице доступа к базе данных. Пароль к БД был создан на предыдущем шаге. Он так же, как и секретный ключ, будет сгенерирован далее.

Значения ключа OPTIONS понадобятся для правильной обработки кириллицы и сохранности данных.

7.3 Формирование списка пакетов

Список пакетов необходим для установки требуемых пакетов на сервере.

В терминале вводим команду:

```
pip freeze > requirements.txt
```

В результате в каталоге проекта появился файл requirements.txt следующего содержания:

```
1  asgiref==3.7.2
2  Django==4.2.6
3  django-debug-toolbar==4.2.0
4  Pillow==10.1.0
5  sqlparse==0.4.4
6  typing_extensions==4.9.0
7  tzdata==2023.3
8  numpy
9  python-dateutil
```

Рисунок 46 – Установленные пакеты

Для правильной работы проекта на сервере в файл requirements.txt добавляем еще два пакета:

mysqlclient

python-dotenv

Эти пакеты необходимы для установки связи Django с базой данных и получения доступа к секретным паролям из переменных окружения.

Так же на данном этапе можно удалить из списка django-debug-toolbar. Также отключаем маршрут DjDT в urls.py и удаляем все созданные настройки для работы DjDT в главе 6.

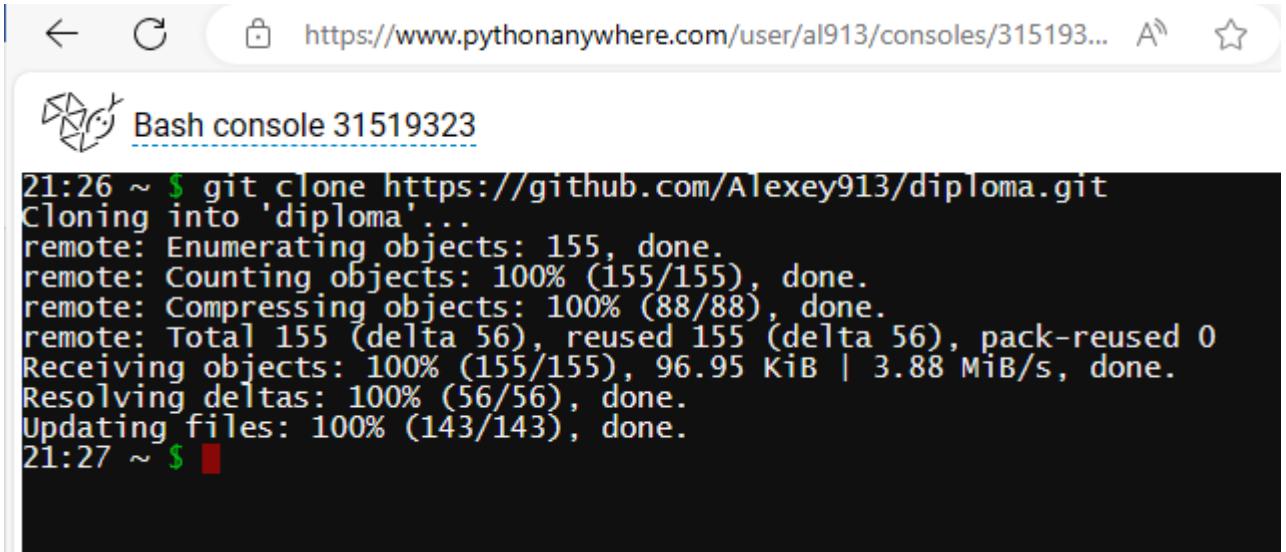
После данных действий всё готово для переноса кода на сервер.

7.4 Создание репозитория GitHub

Репозиторий проекта на GitHub находится по адресу: <https://github.com/Alexey913/diploma.git>. После завершения настроек изменения загружены на сервер GitHub.

Далее необходимо клонировать репозиторий в pythonanywhere командой
git clone https://github.com/Alexey913/diploma.git.

Для её ввода необходимо перейти на вкладку Dashboard и открыть Bash консоль. (рисунок 47).



The screenshot shows a web-based terminal interface. At the top, there's a browser-style header with back, forward, and refresh buttons, and a URL field containing <https://www.pythonanywhere.com/user/al913/consoles/315193...>. Below the header, the title bar says "Bash console 31519323". The main area is a black terminal window displaying the following text:

```
21:26 ~ $ git clone https://github.com/Alexey913/diploma.git
Cloning into 'diploma'...
remote: Enumerating objects: 155, done.
remote: Counting objects: 100% (155/155), done.
remote: Compressing objects: 100% (88/88), done.
remote: Total 155 (delta 56), reused 155 (delta 56), pack-reused 0
Receiving objects: 100% (155/155), 96.95 KiB | 3.88 MiB/s, done.
Resolving deltas: 100% (56/56), done.
Updating files: 100% (143/143), done.
21:27 ~ $
```

Рисунок 47 – Клонирование репозитория Git на сервер

Дожидаемся завершения клонирования. В разделе Dashboard, при нажатии на кнопку Browse files, открывается клонированный каталог проекта.

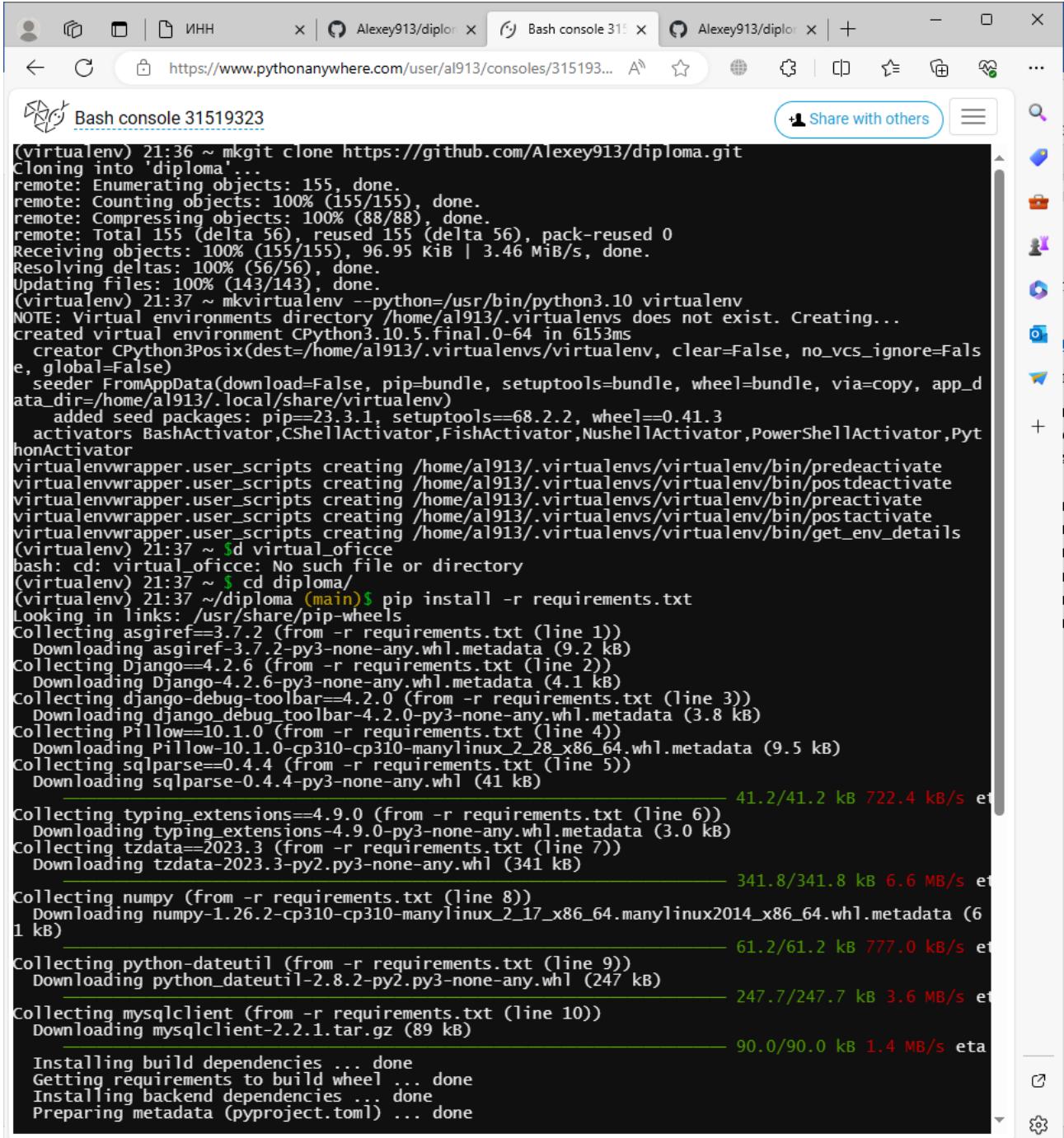
7.5 Настройка проекта на сервере

После завершения клонирования необходимо создать виртуальное окружение командой bash:

```
mkvirtualenv --python=/usr/bin/python3.10 virtualenv
```

Активация виртуального окружения происходит автоматически после создания. Не закрывая консоль, устанавливаем необходимые пакеты (рисунок 48):

```
cd diploma
pip install -r requirements.txt
```



```
(virtualenv) 21:36 ~ mkgit clone https://github.com/Alexey913/diploma.git
Cloning into 'diploma'...
remote: Enumerating objects: 155, done.
remote: Counting objects: 100% (155/155), done.
remote: Compressing objects: 100% (88/88), done.
remote: Total 155 (delta 56), reused 155 (delta 56), pack-reused 0
Receiving objects: 100% (155/155), 96.95 KiB | 3.46 MiB/s, done.
Resolving deltas: 100% (56/56), done.
Updating files: 100% (143/143), done.
(virtualenv) 21:37 ~ mkvirtualenv --python=/usr/bin/python3.10 virtualenv
NOTE: Virtual environments directory /home/al913/.virtualenvs does not exist. Creating...
created virtual environment CPython3.10.5.final.0-64 in 6153ms
  creator CPython3Posix(dest=/home/al913/.virtualenvs/virtualenv, clear=False, no_vcs_ignore=False, global=False)
    seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=/home/al913/.local/share/virtualenv)
      added seed packages: pip==23.3.1, setuptools==68.2.2, wheel==0.41.3
  activators BashActivator,CShellActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator
virtualenvwrapper.user_scripts creating /home/al913/.virtualenvs/virtualenv/bin/predeactivate
virtualenvwrapper.user_scripts creating /home/al913/.virtualenvs/virtualenv/bin/postdeactivate
virtualenvwrapper.user_scripts creating /home/al913/.virtualenvs/virtualenv/bin/preactivate
virtualenvwrapper.user_scripts creating /home/al913/.virtualenvs/virtualenv/bin/postactivate
virtualenvwrapper.user_scripts creating /home/al913/.virtualenvs/virtualenv/bin/get_env_details
(virtualenv) 21:37 ~ $d virtual_oficce
bash: cd: virtual_oficce: No such file or directory
(virtualenv) 21:37 ~ $ cd diploma/
(virtualenv) 21:37 ~/diploma (main)$ pip install -r requirements.txt
Looking in links: /usr/share/pip-wheels
Collecting asgiref==3.7.2 (from -r requirements.txt (line 1))
  Downloading asgiref-3.7.2-py3-none-any.whl.metadata (9.2 kB)
Collecting Django==4.2.6 (from -r requirements.txt (line 2))
  Downloading Django-4.2.6-py3-none-any.whl.metadata (4.1 kB)
Collecting django-debug-toolbar==4.2.0 (from -r requirements.txt (line 3))
  Downloading django_debug_toolbar-4.2.0-py3-none-any.whl.metadata (3.8 kB)
Collecting Pillow==10.1.0 (from -r requirements.txt (line 4))
  Downloading Pillow-10.1.0-cp310-cp310-manylinux_2_28_x86_64.whl.metadata (9.5 kB)
Collecting sqlparse==0.4.4 (from -r requirements.txt (line 5))
  Downloading sqlparse-0.4.4-py3-none-any.whl (41 kB)
Collecting typing_extensions==4.9.0 (from -r requirements.txt (line 6))
  Downloading typing_extensions-4.9.0-py3-none-any.whl.metadata (3.0 kB)
Collecting tzdata==2023.3 (from -r requirements.txt (line 7))
  Downloading tzdata-2023.3-py2.py3-none-any.whl (341 kB)
Collecting numpy (from -r requirements.txt (line 8))
  Downloading numpy-1.26.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (61 kB)
Collecting python-dateutil (from -r requirements.txt (line 9))
  Downloading python_dateutil-2.8.2-py2.py3-none-any.whl (247 kB)
Collecting mysqlclient (from -r requirements.txt (line 10))
  Downloading mysqlclient-2.2.1.tar.gz (89 kB)
Installing build dependencies ... done
Getting requirements to build wheel ... done
Installing backend dependencies ... done
Preparing metadata (pyproject.toml) ... done
41.2/41.2 kB 722.4 kB/s eta
341.8/341.8 kB 6.6 MB/s eta
61.2/61.2 kB 777.0 kB/s eta
247.7/247.7 kB 3.6 MB/s eta
90.0/90.0 kB 1.4 MB/s eta
```

Рисунок 48 – Установка необходимых пакетов в виртуальном окружении

7.6 Создание web-приложения

После окончания установки пакетов на вкладке Dashboard необходимо выбрать пункт Web apps создаём новое приложение кнопкой Add a new web app:

1. Подтверждаем доменное имя для бесплатного профиля кнопкой Next

2. Выбираем пункт Manual configuration (including virtualenvs)
3. Выбираем последнюю из доступных версий Python
4. Подтверждаем выбор очередным нажатием Next
5. All done! Your web app is now set up. Details below.

После завершения работы сервиса мы оказываемся на вкладке настроек web-приложения (рисунок 49).

7.7 Настройка web-приложения

В первую очередь в разделе Virtualenv необходимо указать путь до созданного окружения:

/home/al913/.virtualenvs/virtualenv.

The screenshot shows the PythonAnywhere web application configuration interface. At the top, there's a header with the Python logo, 'pythonanywhere' text, and 'by ANACONDA.' Below the header, a green banner displays the message 'All done! Your web app is now set up. Details below.' On the left, a blue sidebar button says 'al913.pythonanywhere.com' and has a '+ Add a new web app' link. The main content area has a title 'Configuration for al913.pythonanywhere.com'. It includes a 'Reload:' button and a 'Reload al913.pythonanywhere.com' button. A section titled 'Best before date:' explains the site's free hosting terms. It states that the site will be disabled on 'Monday 11 March 2024' if not run until three months from today. A yellow button labeled 'Run until 3 months from today' is present. Below this, a note says 'Paying users' sites stay up forever without any need to log in to keep them running.' In the 'Traffic:' section, it shows traffic statistics for the current month, yesterday, and the previous hour. The table is as follows:

	This month (previous month)	18 (0)
Today (yesterday)	3 (0)	
Hour (previous hour)	0 (0)	

Рисунок 49 – Настройки web-приложения

Теперь отредактируем wsgi файл, ссылка на который находится в разделе Code.

После этого созданное web-приложение Django будет работать по адресу al913.pythonanywhere.com.

В файле wsgi находим раздел Django (74-90 строки) и удаляем всё лишнее до и после него, а также дополняем (рисунок 50).

Следующий этап настройки – сохранение секретного ключа. Для этого в консоли запускаем интерпретатор Python и воспользуемся функцией token_hex из модуля secrets (рисунок 51).

Копируем токен и выходим из режима интерпретатора командой exit()

В консоли bash переходим в директорию ‘~/diploma/virtual_office’ и выполняем команды добавления паролей (рисунок 52).



```
/var/www/al913_pythonanywhere_com_wsgi.py

1 # ++++++ DJANGO ++++++
2 # To use your own django app use code like this:
3 import os
4 import sys
5
6 from dotenv import load_dotenv
7
8
9 project_folder = os.path.expanduser('~/diploma') # adjust as appropriate
10 load_dotenv(os.path.join(project_folder, '.env'))
11
12 ## assuming your django settings file is at '/home/al913/mysite/mysite/settings.py'
13 ## and your manage.py is at '/home/al913/mysite/manage.py'
14 path = '/home/al913/diploma'
15 #if path not in sys.path:
16 sys.path.append(path)
17
18 os.environ['DJANGO_SETTINGS_MODULE'] ='virtual_office.settings'
19
20 ## then:
21 from django.core.wsgi import get_wsgi_application
22 application = get_wsgi_application()
```

Рисунок 50 – Содержание WSGI-файла



Ipython3.10 console 31464637

```
IPython 8.3.0 -- An enhanced Interactive Python. Type '?' for help.  
In [1]: import secrets  
In [2]: secrets.token_hex()  
Out[2]: 'e904108aa4c8126c19cce6d11257d835ac7864278faee86b'  
In [3]: exit()  
Console closed.
```

Рисунок 51 – Генерация секретного ключа

```
Successfully installed Django-4.2.6 Pillow-10.1.0  
(virtualenv) 03:00 ~/qb_diploma/virtual_office (main)$ echo "export SECRET_KEY=e904108aa4c8126c19cce6d11257d835ac7864278faee86b" >> .env  
(virtualenv) 05:43 ~/gb_diploma/virtual_office (main)$ echo "export MYSQL_PASSWORD=2d835ac7864278faee86b" >> .env  
(virtualenv) 05:43 ~/gb_diploma/virtual_office (main)$
```

Рисунок 52 – Добавление паролей

7.8 Создание баз данных

Для создания баз данных необходимо связать консоль с паролями.

Вводим команду

```
echo 'set -a; source ~/myproject/.env; set +a' >>  
~/.virtualenvs/virtualenv/bin/postactivate
```

Таким образом при старте консоли мы будем получать доступ к переменным окружения. Выходим из консоли командой exit и открываем снова на закладке Web в разделе Virtualenv. Кликаем по ссылке Start a console in this virtualenv.

Далее применяем миграции к базе данных (рисунок 53):

```
python manage.py migrate
```

Также необходимо заполнить базу данных водительских категорий (рисунок 54):

```
python manage.py fill_driver_category_db
```

Далее собираем статические файлы проекта и приложений в одном месте.

Для этого выполним команду **python manage.py collectstatic** (рисунок 55).

```
(virtualenv) 06:26 ~/diploma (main)$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contacts_app, contenttypes, doc_app, education_app, planning_app, property_app, sessions, user_app
Running migrations:
  Applying contenttypes_0001_initial... OK
  Applying auth_0001_initial... OK
  Applying admin_0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes_0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying user_app.0001_initial... OK
  Applying contacts_app.0001_initial... OK
  Applying contacts_app.0002_alter_email_email_alter_phone_phone... OK
  Applying doc_app.0001_initial... OK
  Applying doc_app.0002_alter_militariyticket_category... OK
  Applying doc_app.0003_alter_militariyticket_description_and_more... OK
  Applying education_app.0001_initial... OK
  Applying education_app.0002_alter_diploma_description... OK
  Applying education_app.0003_alter_diploma_year_of_finish_edu_and_more... OK
  Applying planning_app.0001_initial... OK
  Applying planning_app.0002_alter_remind_repeat_id... OK
  Applying property_app.0001_initial... OK
  Applying property_app.0002_rename_descripion_realty_description_and_more... OK
  Applying property_app.0003_alter_realty_descripion_alter_realty_type_property_and_more... OK
  Applying property_app.0004_alter_transport_registration_number_and_more... OK
  Applying sessions.0001_initial... OK
(virtualenv) 06:27 ~/diploma (main)$
```



Рисунок 53 – Применение миграций

```
(virtualenv) 06:38 ~/diploma (main)$ python manage.py fill_driver_category_db
A - Мотоциклы
A1 - Легковые мотоциклы
B - Легковые автомобили, небольшиегрузовики (до 3,5 тонн)
B1 - Трициклы, квадрициклы
C - Грузовые автомобили от 3,5 тонн
C1 - Средние грузовики (от 3,5 до 7,5 тонн)
D - Автобусы
D1 - Небольшие автобусы
BE - Легковые автомобили с прицепом
CE - Грузовые автомобили с прицепом
C1E - Средние грузовики с прицепом
DE - Автобусы с пицепом
D1E - Небольшие автобусы с прицепом
M - Мопеды
Tm - Трамваи
Tb - Троллейбусы
(virtualenv) 06:39 ~/diploma (main)$
```

Рисунок 54 – Заполнение базы данных водительских категорий

```
(virtualenv) 06:48 ~/diploma (main)$ python manage.py collectstatic
132 static files copied to '/home/al913/diploma/static'.
(virtualenv) 06:48 ~/diploma (main)$
```

Рисунок 55 – Сбор статических файлов

После завершения сбора переходим на вкладку Web в раздел Static files.

В поле URL записываем /static/. Это содержимое константы STATIC_URL в настройках проекта. В поле Directory необходимо ввести абсолютный путь до каталога со статикой. Это информация получена в результате работы команды collectstatic. Это путь /home/al913/diploma/static (рисунок 56).

После перезагрузки сервера статические данные начали автоматически раздаваться.

После того, как проект настроен и успешно работает, остается создать суперпользователя, чтобы получить доступ к административной панели.

Открываем консоль и выполняем команду **python manage.py createsuperuser**. Далее производим действия, указанные в главе 6 для создания суперпользователя.

После финальной перезагрузки переходим по ссылке <https://al913.pythonanywhere.com/> в приложение Виртуальный офис (рисунок 57).

Static files:

Files that aren't dynamically generated by your code, like CSS, JavaScript or uploaded files, can be served much faster straight off the disk if you specify them here. You need to Reload your web app to activate any changes you make to the mappings below.

URL	Directory	Delete
/static/	/home/al913/diploma/static	
<i>Enter URL</i>	<i>Enter path</i>	

Рисунок 56 – Настройка путей статических файлов

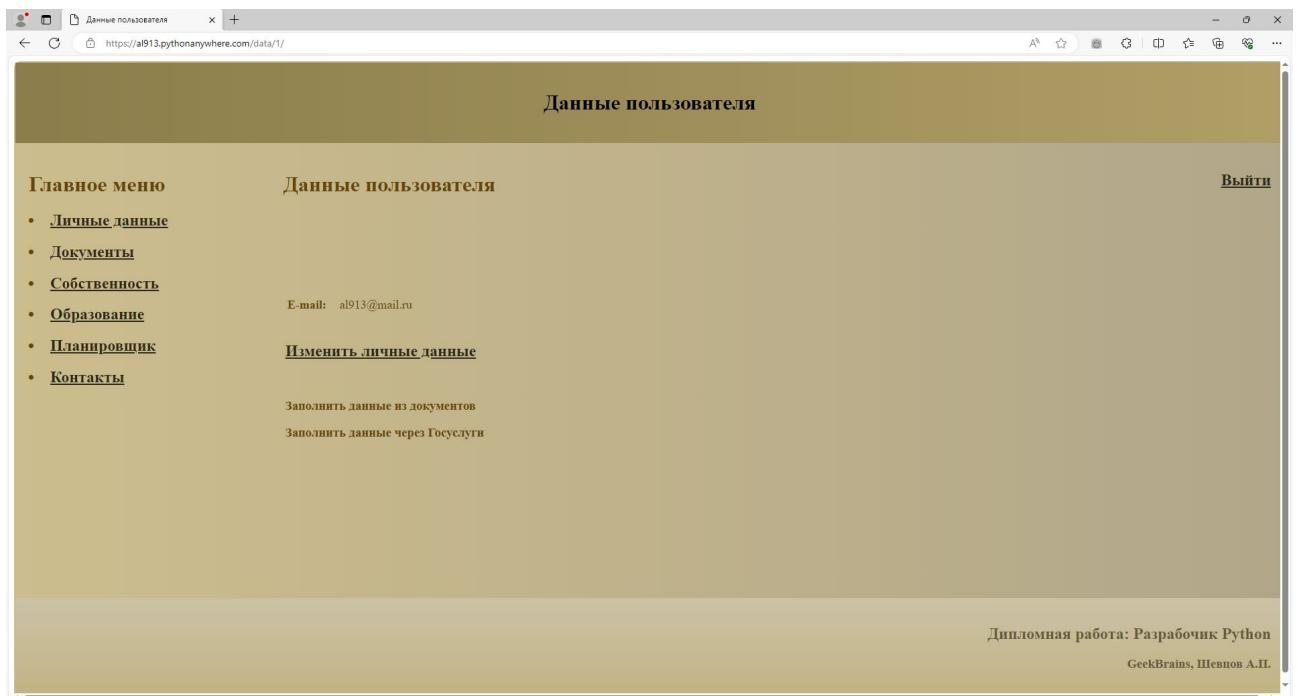


Рисунок 57 – Работа приложения на сервере

Глава 8. Тестирование

Тестирование выступает завершающим этапом в разработке сайта. В результате тестирования выявляются ошибки в проекте, оценивается соответствие требованиям заказчика. На основе тестирования принимается решение о готовности проекта к сдаче заказчику.

На начальном этапе проектирования проверяется весь функционал сайта. В данном проекте были протестированы следующие элементы:

- регистрация и авторизация пользователей;
- добавление документов;
- редактирование документов;
- удаление документов;
- добавление контактов;
- редактирование контактов;
- удаление контактов;
- добавление события;
- редактирование события;
- удаление событий;
- редактирование позиций из интерфейса администратора.

В тестировании безопасности сайт не нуждается, так как Django обеспечивает защиту от основных видов атак.

Тестирование быстродействия (проверка скорости загрузки сайта для оценки скорости работы скриптов, загрузки изображений и контента). Данный вид тестирования позволяет выявлять проблемные участки сайта, чтобы в дальнейшем оптимизировать процесс загрузки. А также результаты позволяют оценить соответствие сайта поставленным требованиям быстродействия. Показательными результатами будет время ответа сервера при загрузке страницы категории, так как на данной странице находятся фильтры,

формирование которых наиболее ресурсозатратно. Результаты отображены на рисунке 54.

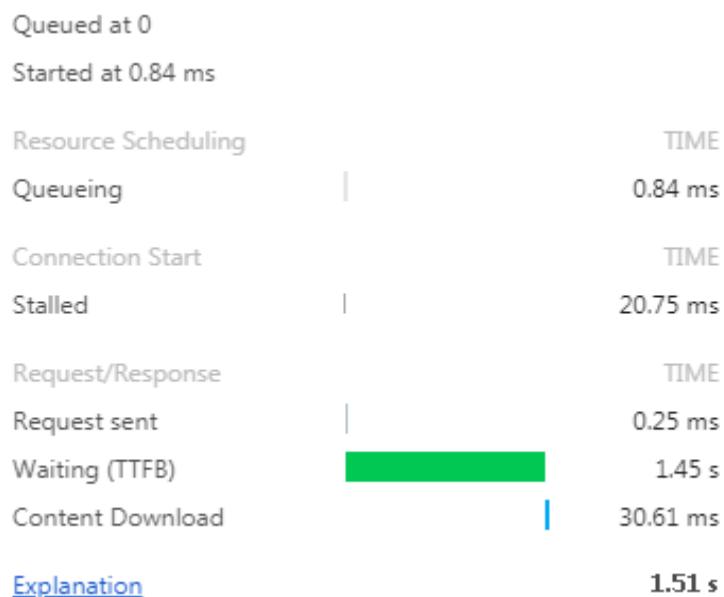


Рисунок 54 – Время загрузки страницы создания события

Сайт тестировался при создании события с ежедневным напоминанием. По результатам тестирования видно, что время ответа сервера составляет 1.45 секунды, остальное время можно не принимать во внимание, так как оно напрямую зависит от скорости интернет-соединения.

Заключение

При работе над дипломным проектом изучена литература в соответствии со списком. Были рассмотрены основные принципы работы языка Python во взаимодействии с фреймворком Django, а также принцип работы паттерна «Model-View-Template» (MVT).

Результатом выполнения выпускной квалификационной работы является web-сайт, развернутый на сервере. Данное web-приложение ориентировано на физические лица и позволяет заменить более нагруженные приложения, такие как Госуслуги или Личный кабинет налогоплательщика. С его помощью можно загружать, просматривать и редактировать различные документы. Также приложение позволяет добавлять и редактировать контакты и события в календаре.

При разработке проекта были проанализированы современные технологии и паттерны программирования, позволяющие создавать конкурентоспособные web-приложения. Отличным решением для выполнения поставленной задачи оказались язык программирования Python и фреймворк Django.

В дальнейшем возможна доработка сайта добавлением выгрузки личных данных, данных о документах, собственности и образовании с сайта Госуслуги. В данный момент реализация данного функционала не возможна по причине ограничений интеграции API со стороны Госуслуг (см. приложение 6).

Кроме того, в целях расширения функционала возможно подключение приложения загрузки сканов документов, а также реализовать возможность сохранения данных пользователя на устройстве для возможности просмотра данных при отсутствии интернет-соединения.

Список использованной литературы и ресурсов

1. Python [Электронный ресурс]

Режим доступа: <https://www.python.org/>

2. Django The web framework for perfectionists with deadlines [Электронный ресурс] – Режим доступа: <https://www.djangoproject.com/>

3. Документация Django [Электронный ресурс]

Режим доступа: <https://djangodoc.ru/>

4. Python. Высокоуровневый язык программирования [Электронный ресурс] – Режим доступа: <https://habr.com/ru/hubs/python/articles/>

5. Описание архитектуры Django [Электронный ресурс]

Режим доступа: <https://metanit.com/python/django/1.1.php>

6. Про Python [Электронный ресурс]

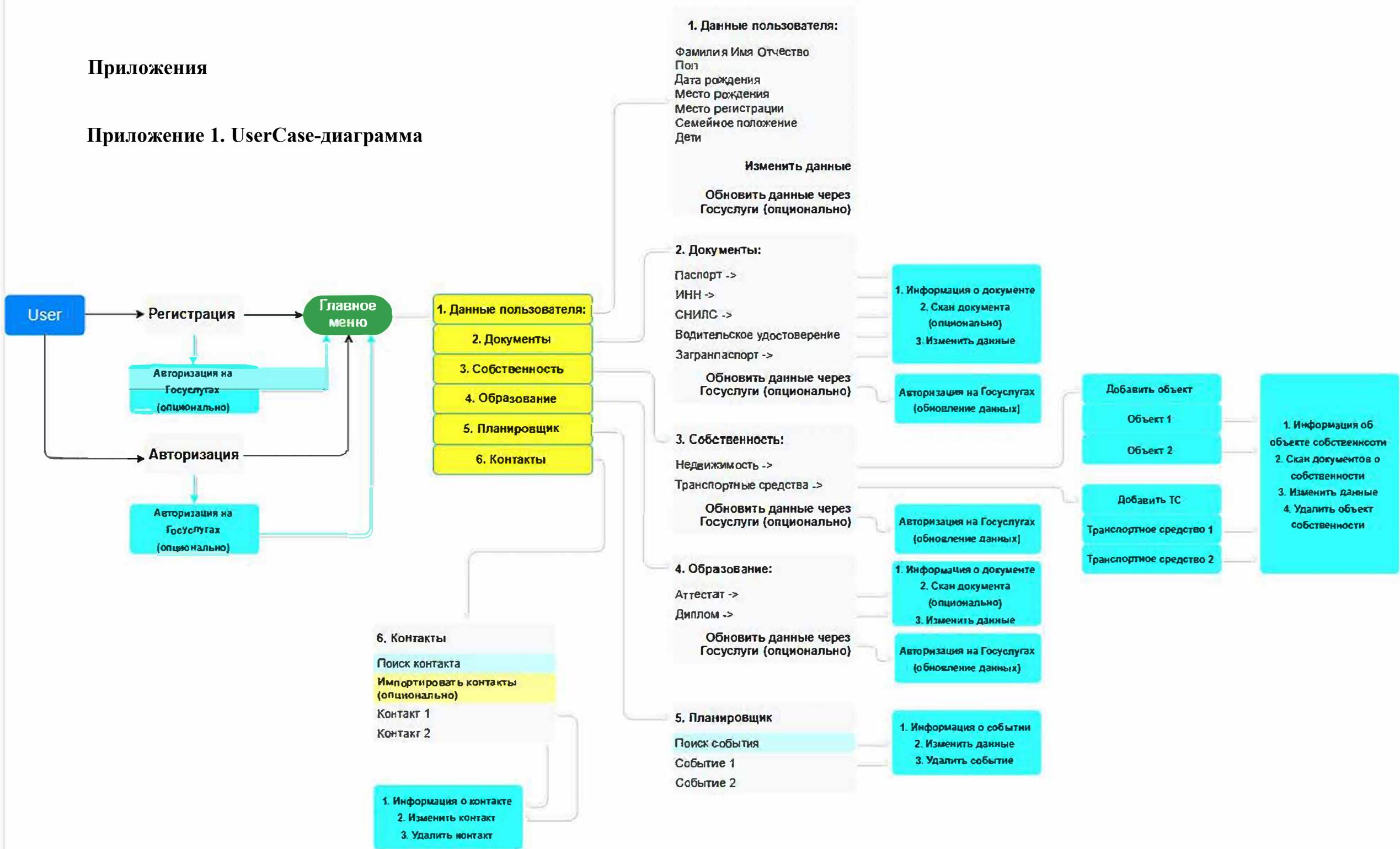
Режим доступа: <https://proprogs.ru/python>

7. Тестирование проектов Django [Электронный ресурс]

Django <https://habr.com/ru/articles/122156/>

Приложения

Приложение 1. UserCase-диаграмма



Приложение 2. User Interface

Авторизация

Авторизация

[Регистрация | Авторизация](#)

Для работы в виртуальном
офисе необходимо
зарегистрироваться или
авторизоваться

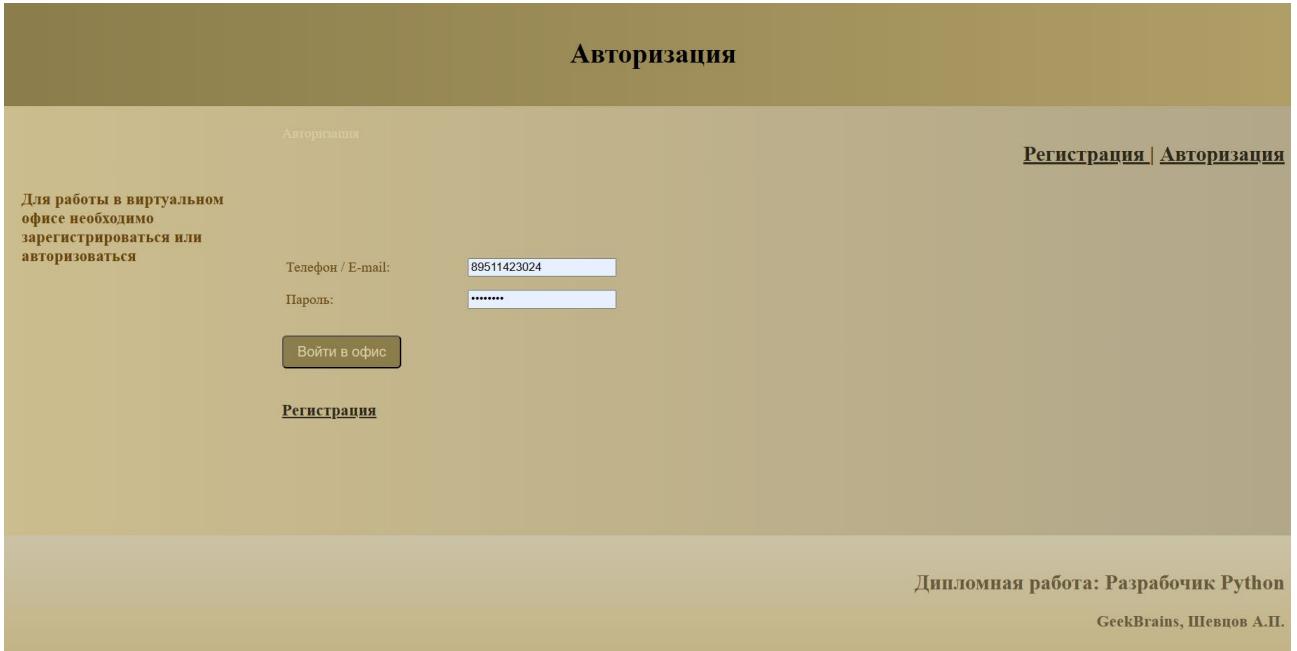
Телефон / E-mail:

Пароль:

[Войти в офис](#)

[Регистрация](#)

Дипломная работа: Разработчик Python
GeekBrains, Шевцов А.П.



Регистрация

Регистрация

[Регистрация | Авторизация](#)

Для работы в виртуальном
офисе необходимо
зарегистрироваться или
авторизоваться

Телефон / E-mail:

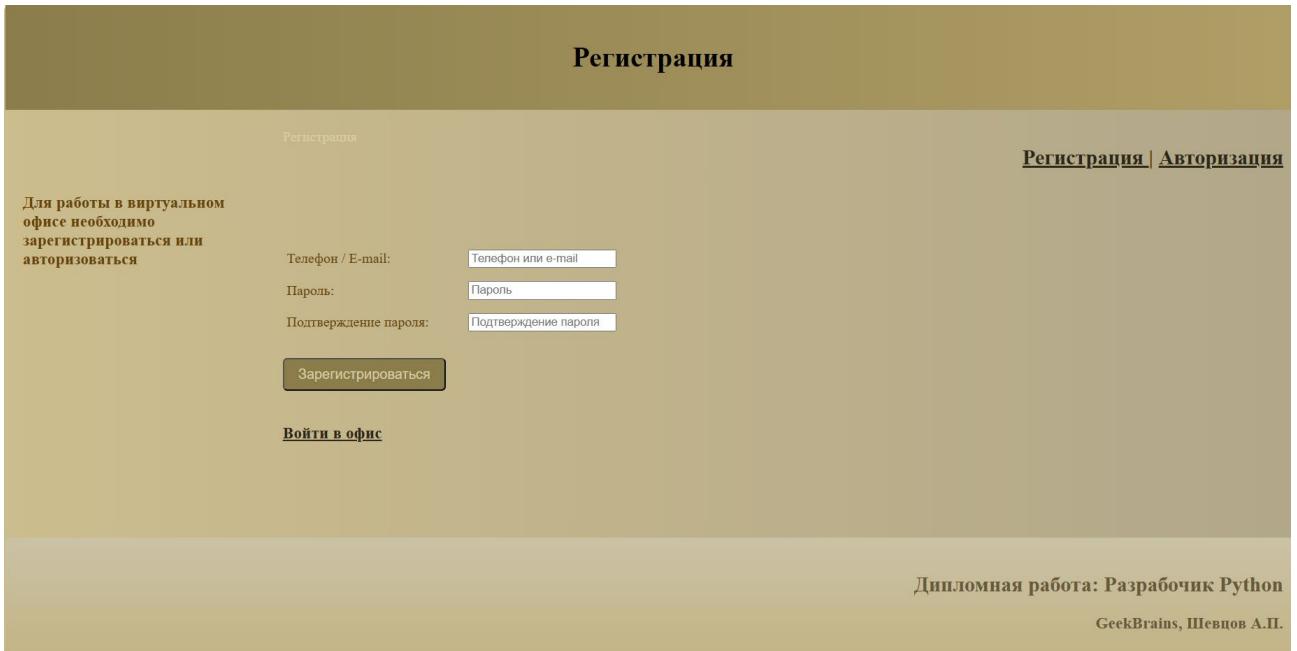
Пароль:

Подтверждение пароля:

[Зарегистрироваться](#)

[Войти в офис](#)

Дипломная работа: Разработчик Python
GeekBrains, Шевцов А.П.



Данные пользователя

Главное меню

- [Личные данные](#)
- [Документы](#)
- [Собственность](#)
- [Образование](#)
- [Планировщик](#)
- [Контакты](#)

Данные пользователя Шевцов

Выход

Фамилия: Шевцов
Дата рождения: 24 сентября 1990 г.
Пол: Мужской
Телефон: 89511423024

[Изменить личные данные](#)

Заполнить данные из документов
Заполнить данные через Госуслуги

Дипломная работа: Разработчик Python
GeekBrains, Шевцов А.П.

Изменение личных данных

Главное меню

- [Личные данные](#)
- [Документы](#)
- [Собственность](#)
- [Образование](#)
- [Планировщик](#)
- [Контакты](#)

Пользователь Шевцов

Выход

Изменение личных данных

Фамилия:
Имя:
Отчество:
Пол:

Дата рождения:

Место рождения:
Место жительства:
Телефон:
E-mail:

Дипломная работа: Разработчик Python
GeekBrains, Шевцов А.П.

Документы

Главное меню <ul style="list-style-type: none">• Личные данные• Документы• Собственность• Образование• Планировщик• Контакты	Документы пользователя Шевцов <table border="0"><tr><td>Паспорт</td><td>ИИН</td></tr><tr><td>СНИЛС</td><td>Водительское удостоверение</td></tr><tr><td>Заграничный паспорт</td><td>Военный билет</td></tr></table>	Паспорт	ИИН	СНИЛС	Водительское удостоверение	Заграничный паспорт	Военный билет	Выйти
Паспорт	ИИН							
СНИЛС	Водительское удостоверение							
Заграничный паспорт	Военный билет							

[Заполнить данные из документов](#)
[Заполнить данные через Госуслуги](#)

Дипломная работа: Разработчик Python
GeekBrains, Шевцов А.П.

Паспорт

Главное меню <ul style="list-style-type: none">• Личные данные• Документы• Собственность• Образование• Планировщик• Контакты	Пользователь Шевцов <p>Паспорт Шевцов</p> <p>Изменить данные документа</p> <p>Заполнить данные из документов Заполнить данные через Госуслуги</p>	Выйти
--	--	-----------------------

Данные о документе	Сведения о супруге Сведения о детях
Фамилия: Шевцов Дата рождения: 24 сентября 1990 г. Пол: Мужской Серия: 1410 Номер: 103300 Дата выдачи: 5 октября 2010 г. Код подразделения: 310025	Ребенок 1 Фамилия: Шевцов

[Изменить Паспорт](#)

Дипломная работа: Разработчик Python

Собственность пользователя

Главное меню

- [Личные данные](#)
- [Документы](#)
- [**Собственность**](#)
- [Образование](#)
- [Планировщик](#)
- [Контакты](#)

Собственность пользователя

[Недвижимость](#)

[Транспорт](#)

[Заполнить данные из документов](#)

[Заполнить данные через Госуслуги](#)

[Выход](#)

Дипломная работа: Разработчик Python
GeekBrains, Шевцов А.П.

Добавление транспортного средства

Главное меню

- [Личные данные](#)
- [Документы](#)
- [Собственность](#)
- [Образование](#)
- [Планировщик](#)
- [Контакты](#)

Новый объект

Раздел Транспорт пуст. Добавьте Транспорт

Добавление транспортного средства

Тип транспортного средства: Автомобиль

Марка ТС: Марка

Модель ТС: Модель

Год выпуска ТС:

Мощность двигателя, л.с.: Мощность двигателя

Масса ТС, кг: Масса

Грузоподъемность ТС, кг: Грузоподъемность

Дата постановки на учет: dd.mm.yyyy

Описание:

Дипломная работа: Разработчик Python
GeekBrains, Шевцов А.П.

Документ об образовании

Главное меню <ul style="list-style-type: none"> • Личные данные • Документы • Собственность • Образование • Планировщик • Контакты 	Новый документ <p style="color: red;">• Нет сведений об образовании. Добавьте документ</p> <p>Документ об образовании</p> <table border="0" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 15%;">Вид документа:</td> <td style="width: 85%;"><input type="button" value="Диплом"/></td> </tr> <tr> <td>Серия:</td> <td><input type="text" value="Серия"/></td> </tr> <tr> <td>Номер:</td> <td><input type="text" value="Номер"/></td> </tr> <tr> <td>Дата выдачи:</td> <td><input type="text" value="Ад. №№. гггг"/> <input type="button" value="..."/></td> </tr> <tr> <td>Регистрационный номер:</td> <td><input type="text" value="Регистрационный номер"/></td> </tr> <tr> <td>Название учебного заведения:</td> <td><input type="text" value="Учебное заведение"/></td> </tr> <tr> <td>Год начала обучения:</td> <td><input type="text" value="Начало обучения"/></td> </tr> <tr> <td>Год окончания обучения:</td> <td><input type="text" value="Окончание обучения"/></td> </tr> <tr> <td>Специальность:</td> <td><input type="text" value="Специальность"/></td> </tr> <tr> <td>Специализация:</td> <td><input type="text" value="Специализация"/></td> </tr> <tr> <td colspan="2" style="border: 1px solid black; height: 40px; vertical-align: top;">Дополнительные сведения:</td> </tr> </table> <p style="text-align: center;"><input type="button" value="Добавить объект"/></p>	Вид документа:	<input type="button" value="Диплом"/>	Серия:	<input type="text" value="Серия"/>	Номер:	<input type="text" value="Номер"/>	Дата выдачи:	<input type="text" value="Ад. №№. гггг"/> <input type="button" value="..."/>	Регистрационный номер:	<input type="text" value="Регистрационный номер"/>	Название учебного заведения:	<input type="text" value="Учебное заведение"/>	Год начала обучения:	<input type="text" value="Начало обучения"/>	Год окончания обучения:	<input type="text" value="Окончание обучения"/>	Специальность:	<input type="text" value="Специальность"/>	Специализация:	<input type="text" value="Специализация"/>	Дополнительные сведения:		Выход
Вид документа:	<input type="button" value="Диплом"/>																							
Серия:	<input type="text" value="Серия"/>																							
Номер:	<input type="text" value="Номер"/>																							
Дата выдачи:	<input type="text" value="Ад. №№. гггг"/> <input type="button" value="..."/>																							
Регистрационный номер:	<input type="text" value="Регистрационный номер"/>																							
Название учебного заведения:	<input type="text" value="Учебное заведение"/>																							
Год начала обучения:	<input type="text" value="Начало обучения"/>																							
Год окончания обучения:	<input type="text" value="Окончание обучения"/>																							
Специальность:	<input type="text" value="Специальность"/>																							
Специализация:	<input type="text" value="Специализация"/>																							
Дополнительные сведения:																								
Дипломная работа: Разработчик Python																								

Данные об образовании

Главное меню <ul style="list-style-type: none"> • Личные данные • Документы • Собственность • Образование • Планировщик • Контакты 	Данные об образовании пользователя Шевцов <p style="color: red;">• Документ об образовании создан</p> <p><u>Диплом № 1410 1010101</u></p> <p>Добавить Данные об образовании</p>	Выход
Дипломная работа: Разработчик Python <small>GeekBrains, Шевцов А.П.</small>		

Данные об образовании

Главное меню <ul style="list-style-type: none"> • Личные данные • Документы • Собственность • Образование • Планировщик • Контакты 	Диплом № 1410 1010101 Добавить Данные об образовании	Выйти												
Диплом <table border="0" style="width: 100%;"> <tr> <td style="width: 30%;">Вид документа:</td> <td>Диплом</td> </tr> <tr> <td>Номер:</td> <td>1010101</td> </tr> <tr> <td>Серия:</td> <td>1410</td> </tr> <tr> <td>Дата выдачи:</td> <td>12 декабря 2023 г.</td> </tr> <tr> <td colspan="2">Название учебного заведения: БГТУ им. Шухова</td> </tr> <tr> <td colspan="2">Дополнительные сведения:</td> </tr> </table> Изменить данные Удалить Диплом Добавить Данные об образовании			Вид документа:	Диплом	Номер:	1010101	Серия:	1410	Дата выдачи:	12 декабря 2023 г.	Название учебного заведения: БГТУ им. Шухова		Дополнительные сведения:	
Вид документа:	Диплом													
Номер:	1010101													
Серия:	1410													
Дата выдачи:	12 декабря 2023 г.													
Название учебного заведения: БГТУ им. Шухова														
Дополнительные сведения:														

Дипломная работа: Разработчик Python
GeekBrains, Шевцов А.П.

События на сегодня

Главное меню <ul style="list-style-type: none"> • Личные данные • Документы • Собственность • Образование • Планировщик • Контакты 	<table border="0" style="width: 100%;"> <tr> <td style="width: 50%; vertical-align: top;"> Поиск событий по названию <input type="text" value="Заголовок"/> <input type="button" value="Найти события"/> </td> <td style="width: 50%; vertical-align: top;"> Поиск событий по дате <input type="text" value="dd.mm.yyyy"/> <input type="text" value="dd.mm.yyyy"/> <input type="button" value="Найти события"/> </td> </tr> </table>	Поиск событий по названию <input type="text" value="Заголовок"/> <input type="button" value="Найти события"/>	Поиск событий по дате <input type="text" value="dd.mm.yyyy"/> <input type="text" value="dd.mm.yyyy"/> <input type="button" value="Найти события"/>	Выйти
Поиск событий по названию <input type="text" value="Заголовок"/> <input type="button" value="Найти события"/>	Поиск событий по дате <input type="text" value="dd.mm.yyyy"/> <input type="text" value="dd.mm.yyyy"/> <input type="button" value="Найти события"/>			
События на сегодня <table border="0" style="width: 100%;"> <tr> <td style="width: 50%; vertical-align: top;"> События в текущем месяце <div style="background-color: #f0f0f0; padding: 2px; border-radius: 5px; display: inline-block;"> Декабрь 2023 Пн Вт Ср Чт Пт Сб Вс 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 </div> </td> <td style="width: 50%; vertical-align: top;"> Добавить событие На сегодня событий нет Сегодня: 15, Декабрь, 2023 Текущее время: 02:25 Заполнить данные из документов Заполнить данные через Госуслуги </td> </tr> </table>			События в текущем месяце <div style="background-color: #f0f0f0; padding: 2px; border-radius: 5px; display: inline-block;"> Декабрь 2023 Пн Вт Ср Чт Пт Сб Вс 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 </div>	Добавить событие На сегодня событий нет Сегодня: 15, Декабрь, 2023 Текущее время: 02:25 Заполнить данные из документов Заполнить данные через Госуслуги
События в текущем месяце <div style="background-color: #f0f0f0; padding: 2px; border-radius: 5px; display: inline-block;"> Декабрь 2023 Пн Вт Ср Чт Пт Сб Вс 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 </div>	Добавить событие На сегодня событий нет Сегодня: 15, Декабрь, 2023 Текущее время: 02:25 Заполнить данные из документов Заполнить данные через Госуслуги			

Дипломная работа: Разработчик Python
GeekBrains, Шевцов А.П.

Найдено по названию Напом:

Главное меню

- [Личные данные](#)
- [Документы](#)
- [Собственность](#)
- [Образование](#)
- [Планировщик](#)
- [Контакты](#)

Поиск событий по названию

Поиск событий по дате

Найти события

Найти события

Выйти

События в текущем месяце

Декабрь 2023
Пн Вт Ср Чт Пт Сб Вс
1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31

Сегодня: 15, Декабрь, 2023
Текущее время: 02:26

Добавить событие

По Вашему запросу ничего не найдено

[Заполнить данные из документов](#)
[Заполнить данные через Госуслуги](#)

Дипломная работа: Разработчик Python
GeekBrains, Шевцов А.П.

Список контактов

Главное меню

- [Личные данные](#)
- [Документы](#)
- [Собственность](#)
- [Образование](#)
- [Планировщик](#)
- [Контакты](#)

Поиск контактов

Найти контакты

Список контактов

Список контактов пуст

Добавить запись

Дипломная работа: Разработчик Python
GeekBrains, Шевцов А.П.

Сведения о контакте

Главное меню

- [Личные данные](#)
- [Документы](#)
- [Собственность](#)
- [Образование](#)
- [Планировщик](#)
- [Контакты](#)

Поиск контактов

[Выход](#)

Сведения о контакте Шевцов Алексей

Фамилия:	Шевцов	Изменить контакт Добавить номер телефона Добавить e-mail Удалить контакт
Имя:	Алексей	
Телефон:	87929002702	
E-mail:	al913@mail.ru	

Дипломная работа: Разработчик Python
GeekBrains, Шевцов А.П.

Найдено по запросу Шевцов:

Главное меню

- [Личные данные](#)
- [Документы](#)
- [Собственность](#)
- [Образование](#)
- [Планировщик](#)
- [Контакты](#)

Поиск контактов

[Выход](#)

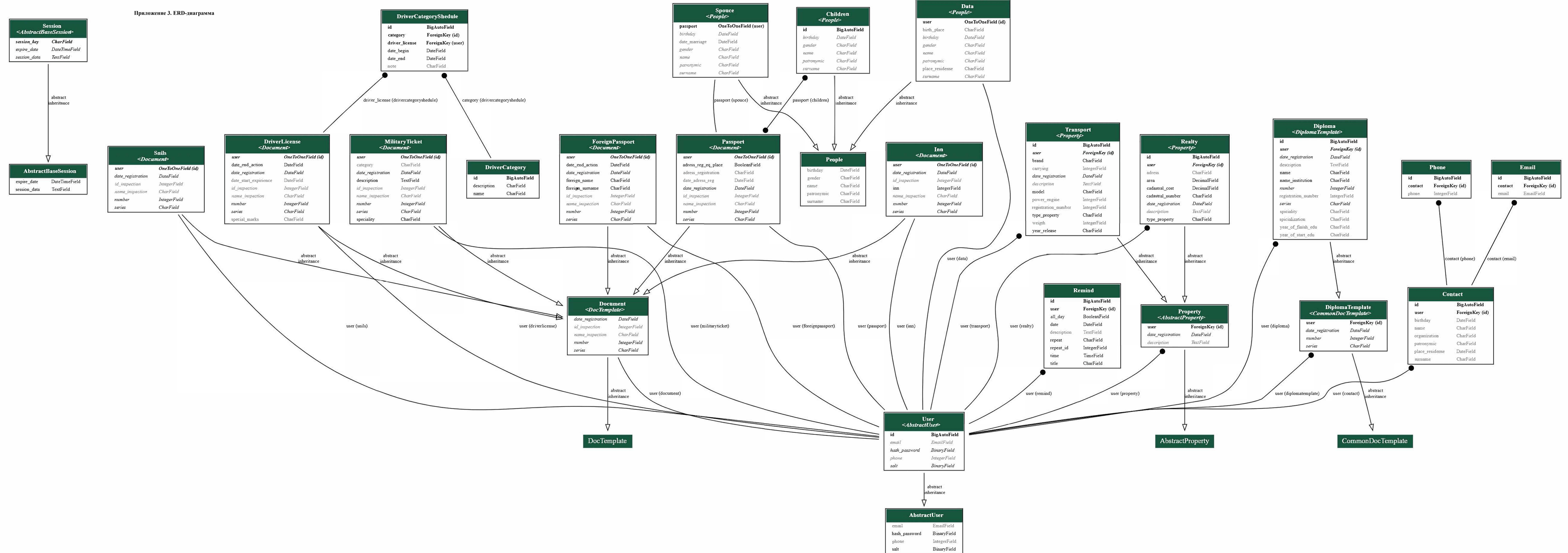
Найдено по запросу Шевцов:

- [Шевцов Алексей](#)

Добавить запись

Дипломная работа: Разработчик Python
GeekBrains, Шевцов А.П.

Приложение 3. ERD-диаграмма



Приложение 4. Код приложения, описание кода, ссылка на онлайн-сервер

Код приложения в полном объеме приведен в репозитории GitHub по ссылке:

<https://github.com/Alexey913/diploma.git>

Особенности реализации кода:

1. Приложение abstract_app – отвечает за создание абстрактных моделей, а также хранит в себе методы, функции, переменные и константы, необходимые для использования всеми приложениями.

В нем размещены списки для реализации меню и выпадающих списков, функции проверок авторизации пользователя (реализуется в виде декоратора для всех функций-представлений, задействованных в urls.py), проверки существования документов (декоратор), функция get_data_with_verbose_name для вывода в шаблоны сущностей из БД с полями.

Также размещена функция pageNotFound, отвечающая за отображение страницы ошибки 404.

2. Приложение contacts_app – отвечает за создание, редактирование, просмотр и удаление контактов. Телефон и E-mail вынесены в отдельные БД, поэтому для контакта, телефона и e-mail реализованы отдельные функции. Реализована функция поиска контактов (закреплена в базовом шаблоне с помощью добавления в переменную TEMPLATES файла settings.py строки 'contacts_app.views.inject_form').

3. Приложение doc_app – отвечает за создание, редактирование, просмотр документов. Реализованы паспорт, ИНН, СНИЛС, водительское удостоверение, заграничный паспорт, военный билет. Приложение не предусматривает удаление документа.

4. Приложение education_app – отвечает за создание, редактирование, просмотр и удаление документов об образовании. Реализована возможность выбора типа документа с помощью выпадающего списка.

5. Приложение planning_app – отвечает за создание, редактирование, просмотр и удаление событий. Так же, как и в приложении contacts_app, реализована функция поиска по заголовку и по диапазону дат событий.

6. Приложение property_app – отвечает за создание, редактирование, просмотр и удаление документов о собственности. Реализована возможность выбора типа собственности – транспорт и недвижимость, с которыми совершаются операции CRUD.

7. Приложение user_app – основное приложение, отвечает за создание, редактирование и просмотр данных о пользователе. Реализованы функции авторизации и регистрации пользователей. В целях углубленного изучения работы Django и функционала работы Django с сессиями в проекте реализована пользовательская система регистрации и авторизации, с собственными валидаторами форм, а также реализована возможность ввода на выбор пользователя e-mail или номера телефона при регистрации и авторизации.

Для однотипных объектов в целях соблюдения принципа DRY реализованы общие функции к которым обращаются конкретные функции-представления (более подробное описание в главе 2).

Приложение развернуто на сайте <https://www.pythonanywhere.com/> и доступно по ссылке:

<http://al913.pythonanywhere.com/>

Приложение 5. Краткая информация по работе с Git.

В целях оптимизации процесса написания кода использовалась система контроля версий Git, консольно реализуемая в Virtual Studio Code.

Последовательность основных команд, используемых при реализации проекта:

git init – инициализация репозитория;

git add – добавление файлов в репозиторий;

git commit -m “” – подтверждение сохранения файлов

git branch – вывод списка веток;

git remote add – добавление удаленного репозитория;

git push – сохранение файлов в удаленный репозиторий из локального;

git pull – сохранение файлов в локальный репозиторий из удаленного;

git status – текущее состояние проекта.

Перечисленный перечень не является исчерпывающим при реализации проекта, часть из них используется с параметрами.

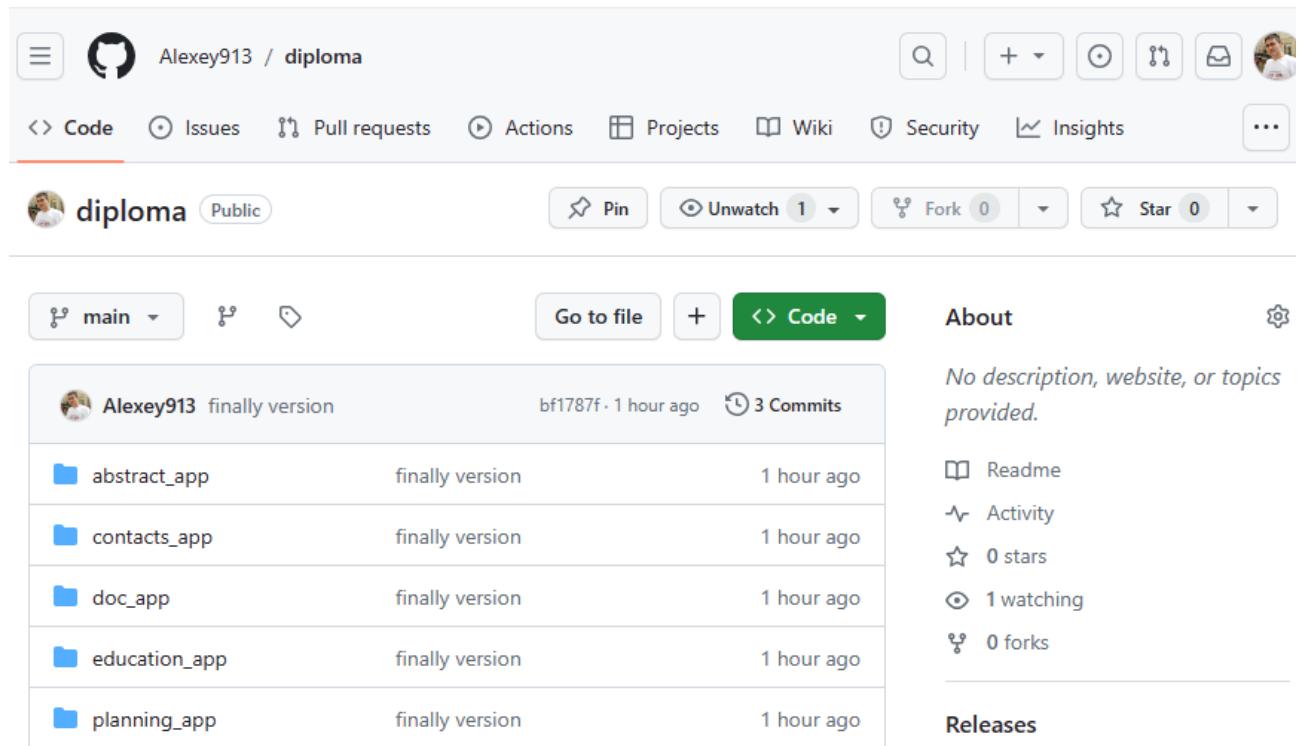


Рисунок – Удаленный репозиторий проекта на GitHub

Приложение 6. Ответ по использованию API Госуслуги



Ваше обращение рассмотрено

Кому: Шевцов Алексей Павлович
От: МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И
МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ
ФЕДЕРАЦИИ
Г. Москва, Пресненская наб., д.10, стр.2

Ваше сообщение 188308510 по теме Портал госуслуг от 27 октября 2023 г. рассмотрено.

Ответ ведомства:

Уважаемый Алексей Павлович! Мы сожалеем, что Вы столкнулись с трудностями при использовании Портала. В рамках рассмотрения Вашего обращения Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации (далее – Минцифры России) сообщает следующее. API могут использовать организации и ИП. Также советуем Вам обратиться с вопросом на api@digital.gov.ru и описать, как вы бы хотели использовать API Госуслуг. Мы постараемся Вам помочь. Приносим извинения за доставленные неудобства. С уважением, Минцифры России.