Geekbrains

**Интеграция CRM-системы с использованием Python и библиотеки Django для оптимизации управления продажами в сфере государственных закупок**

Программа:

Python-разработчик. Специалист

Аксентьев Сергей Юрьевич

Санкт-Петербург

2024

Содержание:

Введение

Теоретическая и практическая главы

Заключение

Список используемой литературы

Приложения

Оглавление

[Введение 4](#_Toc176174154)

[**Теоретическая часть** 5](#_Toc176174155)

[Что такое государственные закупки 5](#_Toc176174156)

[Виды госзакупок 5](#_Toc176174157)

[Что такое CRM-система 5](#_Toc176174158)

[Классификация по уровню обработки информации 5](#_Toc176174159)

[В чем польза CRM-систем 6](#_Toc176174160)

[CRM-система в сфере государственных закупок 7](#_Toc176174161)

[**Практическая часть** 8](#_Toc176174162)

[Общая структура приложения 8](#_Toc176174163)

[Используемые библиотеки 9](#_Toc176174164)

[Описание ключевой директории проекта 10](#_Toc176174165)

[settings.py 11](#_Toc176174166)

[urls.py 13](#_Toc176174167)

[Директория со статикой 14](#_Toc176174168)

[Директория с базовым шаблоном 14](#_Toc176174169)

[Приложения проекта 17](#_Toc176174170)

[Приложение procedures\_app 17](#_Toc176174171)

[Файл apps.py 17](#_Toc176174172)

[Файл моделей (models.py) 17](#_Toc176174173)

[Файл admin.py 19](#_Toc176174174)

[Файл views.py 19](#_Toc176174175)

[Файл urls.py 25](#_Toc176174176)

[Директория с шаблонами html страниц (templates\procedures\_app) 26](#_Toc176174177)

[Директория с миграциями (migrations) 31](#_Toc176174178)

[Приложение user\_app 31](#_Toc176174179)

[Файл apps.py 31](#_Toc176174180)

[Файл models.py 32](#_Toc176174181)

[Файл admin.py 32](#_Toc176174182)

[Файл views.py 34](#_Toc176174183)

[Файл forms.py 35](#_Toc176174184)

[Файл urls.py 37](#_Toc176174185)

[Директория с шаблонами html страниц (templates\user\_app) 38](#_Toc176174186)

# Введение

**Темой проекта** является Интеграция CRM-системы с использованием Python и библиотеки Django для оптимизации управления продажами в сфере государственных закупок.

**Цель:** Создание и интеграция CRM-системы для оптимизации управления продажами в сфере государственных закупок с использованием Python и библиотеки Django

**Актуальность проблемы:** CRM-системы предназначены для эффективного планирования и управления продажами, отслеживания сроков выполнения задач и событий, контроля качества работы сотрудников, а также упрощения процесса работы с тендерами. Разработка такой системы позволит значительно улучшить качество обслуживания тендеров и повысить общую эффективность работы в данной сфере.

**Задачи:**

1. Изучить литературу, касающуюся темы исследования, для определения современных тенденций и практик.
2. Выявить ключевые потребности пользователей для успешного внедрения системы.
3. Разработать CRM-систему, учитывающую специфические потребности в сфере государственных закупок.
4. Настроить и запустить систему на сервере для дальнейшей эксплуатации.

**Инструменты:** Python 3.11, Django 4.2, Git, requests, beautifulsoup4,

**Состав команды:** Аксентьев Сергей Юрьевич

# **Теоретическая часть**

## Что такое государственные закупки

Государственные закупки — главный способ для бюджетных учреждений и госкомпаний найти товары и услуги по максимально выгодной цене. Для этого они размещают заказ с конкретными характеристиками работ или нужной продукции на электронных торговых площадках, где предприниматели, желающие участвовать в торгах, могут оставить заявку. Поставщика определяют в ходе тендера.

Все гостендеры можно найти в единой информационной системе (ЕИС) на сайте **zakupki.gov.ru**. Там публикуются данные об актуальных и состоявшихся торгах. Найти необходимый тендер можно с помощью фильтров и ключевых слов.

### Виды госзакупок

В России государственные торги регулируют два федеральных закона:

* закупки федеральных и муниципальных бюджетных учреждений подчиняются требованиям 44-ФЗ;
* закупки компаний с долей государства больше 50 %, их «дочек» и «внучек», субъектов естественных монополий и бюджетных учреждений, которые проводят тендеры за счёт собственных средств, подчиняются ФЗ-223.

Основой нормативно-правовой базы системы госзакупок является Федеральный закон «О контрактной системе в сфере закупок товаров, работ, услуг для обеспечения государственных и муниципальных нужд» от 05.04.2013 № 44-ФЗ (далее ‒ 44-ФЗ), положения которого основываются на Конституции, Гражданском и Бюджетном кодексах.

В данном проекте за основу будут взяты **закупки по 44-ФЗ.**

## Что такое CRM-система

Система управления взаимоотношениями с клиентами (CRM, CRM-система, сокращение от англ. Customer Relationship Management) — прикладное программное обеспечение для организаций, предназначенное для автоматизации стратегий взаимодействия с заказчиками (клиентами), в частности, для повышения уровня продаж, оптимизации маркетинга и улучшения обслуживания клиентов путём сохранения информации о клиентах и истории взаимоотношений с ними, установления и улучшения бизнес-процессов и последующего анализа результатов. ( <https://ru.wikipedia.org/>)

CRM-система может включать:

* фронтальную часть, обеспечивающую обслуживание клиентов на точках продаж с автономной, распределенной или централизованной обработкой информации;
* операционную часть, обеспечивающую авторизацию операций и оперативную отчётность;
* хранилище данных;
* аналитическую подсистему;
* распределенную систему поддержки продаж: реплики данных на точках продаж или смарт-карты.

### Классификация по уровню обработки информации

*Операционный CRM* — регистрация и оперативный доступ к первичной информации по событиям, компаниям, проектам, контактам.

*Аналитический CRM* — отчётность и анализ информации в различных разрезах (воронка продаж, анализ результатов маркетинговых мероприятий, анализ эффективности продаж в разрезе продуктов, сегментов клиентов, регионов и другие возможные варианты).

*Коллаборативный CRM* (англ. collaboration — сотрудничество; совместные, согласованные действия) — уровень организации тесного взаимодействия с конечными потребителями, клиентами, вплоть до влияния клиента на внутренние процессы компании (опросы, для изменения качеств продукта или порядка обслуживания, веб-страницы для отслеживания клиентами состояния заказа, уведомление по SMS о событиях, связанных с заказом или лицевым счётом, возможность для клиента самостоятельно выбрать и заказать в режиме реального времени продукты и услуги, а также другие интерактивные возможности).

В данном проекте будет исследована **Операционная CRM**

### В чем польза CRM-систем

Основное назначение системы – стандартизация и оптимизация работы оператора во взаимодействии с клиентами (задачами).

Любая CRM система направлена на помощь администратору и должна взять на себя часть мелких забот**.**

Внедрение CRM значительно увеличивает количество клиентов, повышает конверсию и приводит к росту повторных продаж. Преимуществ у программного обеспечения много, например:

* В первую очередь, удобный интерфейс для работы с посетителями. CRM-система сохраняет клиентскую базу, собирает историю взаимодействия с ними, анализирует уровень лояльности и автоматизирует процессы транзакций. Благодаря такому функционалу программа поможет не потерять клиента, который проявил интерес к компании.
* Вторым преимуществом является формирование аналитических отчетов в режиме онлайн. С помощью CRM можно управлять некоторыми кадровыми процессами: контролировать занятость сотрудников, планировать график работы и отпусков. Также программа помогает анализировать этапы воронки продаж и избавляться от неэффективных рекламных каналов. За это отвечает специальный модуль, который позволяет отследить путь клиента от этапа посещения сайта до завершения покупки.
* Гибкость систем – для разного рода деятельности можно настроить определенные задачи в системе, разделить на модули или построить индивидуальную CRM-систему под нужны компании.
* Еще одним важным преимуществом программного обеспечения является автоматизация рабочих процессов. Благодаря такой функции, значительно снижается нагрузка на менеджеров и исключаются ошибки по причине невнимательности сотрудников. Система фиксирует все действия в базе и уведомляет менеджеров компании о срочных задачах: совершить важный звонок или отправить письмо. Инструменты CRM-системы также упрощают ведение документооборота за счет внутренних шаблонов и скриптов.

В целом программа позволяет любому руководителю ставить и корректировать задачи, контролировать их выполнение в соответствии с дедлайном и следить за эффективностью работы каждого сотрудника.

## CRM-система в сфере государственных закупок

Основные задачи для CRM-системы в сфере государственных закупок это:

* Поиск закупок по заданным параметрам
* Организация работы на всех стадиях закупки
* 100% контроль всех текущих закупок
* Доступ к контактам заказчика, истории взаимодействия
* Всегда актуальная версия извещения и документации
* Выполнение повседневных процессов компании и автоматизация рутинных задач

# **Практическая часть**

## Общая структура приложения

| .env - *файл для хранения переменных*

| db.sqlite3 – *база данных*

| manage.py - *отвечает за управление проектом*

|

+---crm\_project – *ключевая директория проекта*

| | asgi.py - *используется для налаживания связи между приложением и веб-сервером*

| | settings.py - *содержит в себе все настройки проекта*

| | urls.py - *задаёт ассоциации url адресов с представлениями*

| | wsgi.py - *используется для налаживания связи между приложением и веб-сервером*

| | \_\_init\_\_.py

|

+---procedures\_app – *папка с приложением, отвечающим за обработку процедур*

| | admin.py - *здесь регистрируются модели, для административной части сайта Django*

| | apps.py - *используется для создания конфигурации приложения (имя, инициализация, и др.)*

| | models.py - *файл, который содержит модели базы данных проекта*

| | tests.py – *файл для тестов приложения*

| | urls.py - *для маршрутизации URL-адресов к соответствующим представлениям (views)*

| | views.py - *отвечает за обработку запросов и возврат ответов (логика приложения)*

| | \_\_init\_\_.py

| |

| +---migrations – *папка для файлов миграций (изменения в структуре БД)*

| | | 0001\_initial.py

| | | \_\_init\_\_.py

| |

| +---templates – *папка с шаблонами HTML страниц*

| | \---procedures\_app

| | about.html

| | archive.html

| | index.html

| | new.html

|

+---static - *содержит статические файлы приложения*

| +---admin

| | +---css

| | |

| | +---img

| | |

| | +---js

| |

| +---import\_export

| export.css

| export\_selectable\_fields.js

| guess\_format.js

| import.css

|

+---templates

| base.html

|

+---user\_app – *папка с приложением, отвечающим за обработку сотрудников*

| | admin.py

| | apps.py

| | forms.py

| | models.py

| | tests.py

| | urls.py

| | views.py

| | \_\_init\_\_.py

| |

| +---migrations

| | | 0001\_initial.py

| | | \_\_init\_\_.py

| |

| +---templates

| | \---user\_app

| | login.html

| | registration.html

|

+---user\_procedures\_app – *папка с приложением, отвечающим за обработку процедур сотрудниками*

| admin.py

| apps.py

| forms.py

| models.py

| tests.py

| urls.py

| views.py

| \_\_init\_\_.py

|

+---migrations

| | 0001\_initial.py

| | \_\_init\_\_.py

|

+---templates

| \---user\_procedures\_app

| contracts.html

| my\_procedures.html

| new\_contract.html

| new\_procedure.html

## Используемые библиотеки

1. asgiref==3.8.1\*\*: Утилиты для работы с ASGI (Asynchronous Server Gateway Interface) в Python.

2. beautifulsoup4==4.12.3\*\*: Библиотека для парсинга HTML и XML документов, удобная для извлечения данных из веб-страниц.

3. bs4==0.0.2\*\*: Оболочка для библиотеки Beautiful Soup, обычно не используется отдельно, так как все в Beautiful Soup 4.

4. certifi==2024.2.2\*\*: предоставляет набор безопасных корневых сертификатов для проверки SSL-сертификатов.

5. charset-normalizer==3.3.2\*\*: Библиотека для определения и нормализации кодировки текста.

6. diff-match-patch==20230430\*\*: Библиотека для вычисления различий, создания патчей и других операций с текстом.

7. Django==4.2.11\*\*: веб-фреймворк для создания сложных веб-приложений на Python.

8. django-import-export==4.0.0\*\*: Расширение для Django, позволяющее легко импортировать и экспортировать данные из/в различные форматы.

9. idna==3.7\*\*: Утилиты для работы с IDNA (Internationalized Domain Names in Applications) для корректной обработки доменных имен.

10. lxml==5.2.1\*\*: Библиотека для работы с XML и HTML, обеспечивающая высокую производительность.

11. python-dotenv==1.0.1\*\*: Позволяет загружать переменные окружения из .env файлов, для настройки конфигураций.

12. requests==2.31.0\*\*: Библиотека для работы с HTTP-запросами.

13. soupsieve==2.5\*\*: Поисковая библиотека, используемая с Beautiful Soup для операций по выбору элементов в HTML/XML.

14. sqlparse==0.5.0\*\*: Простая библиотека для разбора SQL-запросов в удобочитаемый формат.

15. tablib==3.5.0\*\*: Библиотека для работы с табличными данными, поддерживает множество форматов (CSV, JSON, Excel и т.д.).

16. tzdata==2024.1\*\*: Библиотека, содержащая актуальную информацию о временных зонах.

17. urllib3==2.2.1\*\*: Расширенная библиотека для работы с HTTP-запросами, обеспечивающая дополнительные функции и улучшения.

## Описание ключевой директории проекта

Файлы asgi.py и wsgi.py создаются автоматически при создании проекта и не требуют изменений.

**asgi.py** - Этот файл используется для настройки приложения с ASGI-сервером, что позволяет обрабатывать веб-сокеты и асинхронные запросы.

**wsgi.py** - Этот файл настраивает WSGI-сервер для запуска вашего приложения, поддерживая стандартные HTTP-запросы.

Файл **\_\_init\_\_.py** - Указывает Python, что данная директория является пакетом. Он может быть пустым или содержать код инициализации. (далее про него не будет упоминаться, так как выполняет ту же функцию)

### settings.py

Файл **settings.py** - Содержит основные настройки проекта, такие как базы данных, файлы статик, параметры безопасности и другие конфигурации.

Разберем файл более подробно:

**import** os

**import** dotenv

**from** pathlib **import** Path

# Переменная, которая указывает на корневую директорию проекта Django, используется для построения путей к другим папкам и файлам в проекте.

BASE\_DIR **=** Path**(**\_\_file\_\_**).***resolve***().***parent***.***parent*

# Загрузка переменные окружения из файла .env. Это обычно используется для хранения конфиденциальных данных, таких как ключи API или настройки базы данных

dotenv\_file **=** os**.***path***.***join***(**BASE\_DIR**,** ".env"**)**

**if** os**.***path***.***isfile***(**dotenv\_file**):**

dotenv**.***load\_dotenv***(**dotenv\_file**)**

# ПРЕДУПРЕЖДЕНИЕ ПО технике БЕЗОПАСНОСТИ: храните секретный ключ, используемый при запуске, в секрете! Значение переменной SECRET\_KEY берется из переменных окружения операционной системы (из файла .env)

SECRET\_KEY **=** os**.***environ***[**"SECRET\_KEY"**]**

# DEBUG используется в настройках веб-приложений на Django для включения режима отладки

DEBUG **=** **False**

# Это список разрешенных хостов для приложения. В продакшн-режиме здесь указываются домены и IP-адреса, с которых приложение сможет принимать запросы

ALLOWED\_HOSTS **=** **[]**

# Настройка указывает, куда перенаправлять пользователей после успешного входа в систему. В данном случае, после логина пользователь будет перенаправлен на главную страницу ("/")

LOGIN\_REDIRECT\_URL **=** "/"

# Список всех приложений, которые используются в проекте

INSTALLED\_APPS **=** **[**

"django.contrib.admin"**,**

"django.contrib.auth"**,**

"django.contrib.contenttypes"**,**

"django.contrib.sessions"**,**

"django.contrib.messages"**,**

"django.contrib.staticfiles"**,**

"procedures\_app"**,**

"import\_export"**,**

"user\_app"**,**

"user\_procedures\_app"**,**

**]**

# Посредники (middleware) обрабатывают запросы и ответы на разных стадиях. Они могут выполнять такие действия, как обработка сессий, аутентификация, и защита от CSRF-атак

MIDDLEWARE **=** **[**

"django.middleware.security.SecurityMiddleware"**,**

"django.contrib.sessions.middleware.SessionMiddleware"**,**

"django.middleware.common.CommonMiddleware"**,**

"django.middleware.csrf.CsrfViewMiddleware"**,**

"django.contrib.auth.middleware.AuthenticationMiddleware"**,**

"django.contrib.messages.middleware.MessageMiddleware"**,**

"django.middleware.clickjacking.XFrameOptionsMiddleware"**,**

**]**

# Указывает, какой файл будет использоваться для маршрутизации (URL routing). В данном случае это crm\_project.urls, где определяются все маршруты (URL) приложения

ROOT\_URLCONF **=** "crm\_project.urls"

# Определяет настройки для обработки шаблонов. Указаны директории, где можно найти шаблоны, и необходимые контекстные процессоры, которые позволяют использовать разные переменные в 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"**,**

**],**

**},**

**},**

**]**

WSGI\_APPLICATION **=** "crm\_project.wsgi.application"

# Определяем настройки для базы данных. "ENGINE" - Указывает, какой тип базы данных используется (в данном случае SQLite). "NAME" - Путь к файлу базы данных в корне проекта

DATABASES **=** **{**

"default"**:** **{**

"ENGINE"**:** "django.db.backends.sqlite3"**,**

"NAME"**:** BASE\_DIR **/** "db.sqlite3"**,**

**}**

**}**

# список валидаторов паролей, которые используются для проверки надежности вводимых паролей

AUTH\_PASSWORD\_VALIDATORS **=** **[**

**{**

"NAME"**:** "django.contrib.auth.password\_validation.UserAttributeSimilarityValidator"**,**

**},**

**{**

"NAME"**:** "django.contrib.auth.password\_validation.MinimumLengthValidator"**,**

**},**

**{**

"NAME"**:** "django.contrib.auth.password\_validation.CommonPasswordValidator"**,**

**},**

**{**

"NAME"**:** "django.contrib.auth.password\_validation.NumericPasswordValidator"**,**

**},**

**]**

# Интернационализация. Устанавливает язык (русский) и часовой пояс (Московский.

LANGUAGE\_CODE **=** "ru-Ru"

TIME\_ZONE **=** "Europe/Moscow"

USE\_I18N **=** **True**

USE\_TZ **=** **True**

# URL для статики (CSS, JS, изображения)

STATIC\_URL **=** "static/"

# Папка, в которую будут собираться статические файлы для продакшн-окружения

STATIC\_ROOT **=** **(**"static"**,)**

# Определяет тип поля автоинкремента для первичных ключей, что помогает избежать проблем с ограничениями базы данных

# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field

DEFAULT\_AUTO\_FIELD **=** "django.db.models.BigAutoField"

# Модель пользователя для аутентификации, что позволяет использовать пользовательскую модель пользователя вместо стандартной

AUTH\_USER\_MODEL **=** "user\_app.User"

### urls.py

Файл **urls.py** в Django отвечает за маршрутизацию (routing) URL-адресов приложения.

Рассмотрим код файла urls.py ключевой директории:

# импортируем админку Django и функции маршрутизации

**from** django**.***contrib* **import** admin

**from** django**.***urls* **import** path**,** include

urlpatterns **=** **[**

path**(**"admin/"**,** admin**.***site***.***urls***),**

path**(**""**,** include**(**"procedures\_app.urls"**,** namespace**=**"procedures\_app"**)),** # домашняя страница

path**(**"user/"**,** include**(**"user\_app.urls"**,** namespace**=**"user\_app"**)),** # работа с пользователями

path**(**

"procedures/"**,** include**(**"user\_procedures\_app.urls"**,** namespace**=**"user\_procedures\_app"**)**

**),** # работа с закупками

**]**

Определение маршрутов:

- path("admin/", admin.site.urls): Этот маршрут устанавливает, что по адресу /admin/ будет доступен интерфейс администрирования.

Включение маршрутов приложений:

- path("", include("procedures\_app.urls", namespace="procedures\_app")): Этот маршрут обрабатывает домашнюю страницу, перенаправляя на URL-ы, определенные в *procedures\_app*.

- path("user/", include("user\_app.urls", namespace="user\_app")): Этот маршрут отвечает за все URL, связанные с работой с пользователями, перенаправляя запросы на *user\_app*.

- path("procedures/", include("user\_procedures\_app.urls", namespace="user\_procedures\_app")): Этот маршрут обрабатывает запросы, связанные с закупками, перенаправляя их на user\_procedures\_app.

Этот файл маршрутизации связывает основные части приложения и позволяет добавлять новые функции в будущем.

## Директория со статикой

В папке «static» проекта Django хранятся статические файлы — изображения, JS и CSS-файлы.

В проекте используется библиотека «django-import-export», поэтому большая часть файлов статики подгружена автоматически для корректной работы данной библиотеки.

Библиотека «django-import-export» предоставляет удобные инструменты для импорта и экспорта данных в Django-приложениях. Она позволяет работать с различными форматами файлов, такими как CSV, Excel и JSON, и включает в себя следующие функции:

1. Импорт данных: легко загружать данные из файлов в вашу модель Django.

2. Экспорт данных: Вы можете экспортировать данные моделей в различные форматы.

3. Настройка полей: Возможность настройки полей, которые будут импортированы или экспортированы.

4. Адаптация к интерфейсу администрирования: Удобная интеграция с Django Admin, включая кнопки для загрузки и скачивания файлов.

Применение данной библиотеки будет указано далее в проекте.

## Директория с базовым шаблоном

Базовый шаблон HTML в проекте Django — это основной файл, который используется для определения структуры и оформления страниц веб-приложения. Он обычно содержит общий HTML-код, такой как заголовок, меню навигации, подвал и ссылки на стили и скрипты. Затем другие шаблоны могут расширять этот базовый шаблон, добавляя свой контент.

Код базового шаблона (base.html):

**{% comment %} Базовая страница с общими настрйоками {% endcomment %}**

<!DOCTYPE html>

<html lang=**"ru"**>

<head>

<meta charset=**"UTF-8"**>

<meta name=**"viewport"** content=**"width=device-width, initial-scale=1.0"**>

<link rel=**"preconnect"** href=**"https://fonts.googleapis.com"**>

<link rel=**"preconnect"** href=**"https://fonts.gstatic.com"** crossorigin>

<link href=**"https://fonts.googleapis.com/css2?family=Reddit+Sans:ital,wght@0,200..900;1,200..900&display=swap"** rel=**"stylesheet"**>

<link rel=**"stylesheet"** href=**"https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/css/bootstrap.min.css"** integrity=**"sha384-xOolHFLEh07PJGoPkLv1IbcEPTNtaed2xpHsD9ESMhqIYd0nLMwNLD69Npy4HI+N"** crossorigin=**"anonymous"**>

<title>**{% block title %}{% endblock %}**</title>

</head>

<body>

<header>

<div class=**"d-flex gap-2 justify-content-center py-5"**>

<a href=**'{% url "procedures\_app:index" %}'** class=**"btn btn-primary btn-lg active rounded-pill px-3"** role=**"button"** aria-pressed=**"true"** >**Главная**</a>

**{% if not user.is\_authenticated %}**

<a href=**'{% url "user\_app:login" %}'** class=**"btn btn-info btn-lg active rounded-pill px-3"** role=**"button"** aria-pressed=**"true"**>**Войти**</a>

**{% else %}**

<a href=**'{% url "procedures\_app:new" %}'** class=**"btn btn-secondary btn-lg active rounded-pill px-3"** role=**"button"** aria-pressed=**"true"**>**Новые закупки**</a>

<a href=**'{% url "user\_procedures\_app:my\_procedures" %}'** class=**"btn btn-success btn-lg active rounded-pill px-3"** role=**"button"** aria-pressed=**"true"**>**Мои закупки**</a>

<a href=**'{% url "user\_procedures\_app:contracts" %}'** class=**"btn btn-warning btn-lg active rounded-pill px-3"** role=**"button"** aria-pressed=**"true"**>**Контракты**</a>

<a href=**'{% url "procedures\_app:archive" %}'** class=**"btn btn-light btn-lg active rounded-pill px-3"** role=**"button"** aria-pressed=**"true"**>**Архив**</a>

<a href=**'{% url "procedures\_app:about" %}'** class=**"btn btn-dark btn-lg active rounded-pill px-3"** role=**"button"** aria-pressed=**"true"**>**О нас**</a>

**{% if user.is\_admin or user.is\_staff %}**

<a href=**'{% url "admin:index" %}'** class=**"btn btn-info btn-lg active rounded-pill px-3"** role=**"button"** aria-pressed=**"true"**>**Админка**</a>

**{% endif %}**

<a href=**'{% url "user\_app:logout" %}'** class=**"btn btn-danger btn-lg active rounded-pill px-3"** role=**"button"** aria-pressed=**"true"**>**Выйти**</a>

**{% endif %}**

</div>

</header>

<main>

<div class=**'container'**>

**{% block content %}{% endblock %}**

</div>

</main>

</body>

</html>

Рассмотрим более подробно строки:

* Тег <head>:

**<meta charset="UTF-8">** - Задает кодировку документа как UTF-8, что позволяет корректно отображать многоязычный контент.

**<meta name="viewport" content="width=device-width, initial-scale=1.0">** - Устанавливает параметры отображения на мобильных устройствах, чтобы сайт был адаптивным.

**<link rel="preconnect" href="https://fonts.googleapis.com">**

**<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>** - Предварительное соединение с Google Fonts для ускорения загрузки шрифтов.

**<link href="https://fonts.googleapis.com/css2?family=Reddit+Sans:ital,wght@0,200..900;1,200..900&display=swap" rel="stylesheet">** - Подключает шрифт Reddit Sans из Google Fonts с возможностью использования разных начертаний.

**<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/css/bootstrap.min.css" integrity="sha384-xOolHFLEh07PJGoPkLv1IbcEPTNtaed2xpHsD9ESMhqIYd0nLMwNLD69Npy4HI+N" crossorigin="anonymous">** - Подключает стили Bootstrap версии 4.6.2 для быстрого и удобного оформления

**<title>{% block title %}{% endblock %}</title>** - Заголовок страницы, который будет заменяться в зависимости от контекста

В Django шаблонах конструкции {% block %} и {% endblock %} используются для определения областей, которые могут быть переопределены в дочерних шаблонах. Это часть механизма наследования шаблонов.

* Тег <body>:

В тегах <body> содержится основное содержимое веб-страницы.

В теге <header> (это верхняя часть сайта, расположенная выше блока с основным контентом и отображаемая на всех страницах)

- Содержит навигационное меню с кнопками для перехода на разные страницы, такие как "Главная", "Войти", "Новые закупки", "Мои закупки" и другие.

- Кнопки оформлены с использованием Bootstrap классов для красивого визуального отображения.

- Видимость кнопок зависит от того, аутентифицирован ли пользователь: если нет, доступна кнопка для входа; если да, показываются дополнительные кнопки, включая административные, если пользователь имеет соответствующие права.

В теге <main> (основной контент страницы)

- Содержит основной контент страницы, обернутый в контейнер.

- Использует {% block content %}{% endblock %} для динамической вставки содержимого из других шаблонов.

# Приложения проекта

## Приложение procedures\_app

Приложение procedures\_app, отвечает за обработку процедур (закупок).

Данное приложение производит поиск новых процедур, добавление их в базу данных, формирование шаблонов главной страницы, страниц с процедурами и прочее.

Рассмотрим подробно код файлов приложения.

### Файл apps.py

Apps.py — это модуль с настройками приложения в Django.

**from** django**.***apps* **import** AppConfig

**class** **ProceduresAppConfig(**AppConfig**):**

default\_auto\_field **=** "django.db.models.BigAutoField"

name **=** "procedures\_app"

verbose\_name **=** "Новые закупки И Архив"

Описание полей:

- default\_auto\_field: устанавливает тип автоматически создаваемого идентификатора для моделей. В данном случае используется BigAutoField, что позволяет использовать более широкий диапазон значений для ID.

- name: указывает имя приложения в Django. Здесь оно обозначено как "procedures\_app".

- verbose\_name: определяет понятное и человеко-читаемое имя приложения, которое будет отображаться в админ-панели и других местах. В данном случае это "Новые закупки И Архив".

### Файл моделей (models.py)

Модели в Django описывают структуру используемых данных.

Данные в программе хранятся в базах данных, и с помощью моделей осуществляется взаимодействие с базой данных.

**import** datetime

**from** django**.***db* **import** models

# БД с новыми процедурами

**class** **NewProcedures(**models**.***Model***):**

method **=** models**.***CharField***(**"Способ определния поставщика"**,** max\_length**=**30**)**

number **=** models**.***CharField***(**"Номер процедуры"**,** max\_length**=**25**)**

purchase\_name **=** models**.***CharField***(**"Предмет закупки"**,** max\_length**=**150**)**

customer **=** models**.***CharField***(**"Заказчик"**,** max\_length**=**200**)**

price **=** models**.***CharField***(**"Начальная цена"**,** max\_length**=**150**)**

start **=** models**.***DateField***(**"Размещено"**)**

update **=** models**.***DateField***(**"Обновлено"**)**

end **=** models**.***DateField***(**"Окончание подачи заявок"**,** max\_length**=**25**)**

purchase\_link **=** models**.***URLField***(**"Ссылка на процедуру"**,** max\_length**=**150**)**

# Имена табличек в админке

**class** **Meta:**

verbose\_name **=** "Новую процедуру"

verbose\_name\_plural **=** "Новые процедуры"

**def** \_\_str\_\_**(**self**)** **->** **str:**

**return** f"№ {self**.***number*}, НМЦК: {self**.***price*}, Предмет закупки: {self**.***purchase\_name*}"

# БД с архивными процедурами

**class** **Archive(**models**.***Model***):**

method **=** models**.***CharField***(**"Способ определния поставщика"**,** max\_length**=**30**)**

number **=** models**.***CharField***(**"Номер процедуры"**,** max\_length**=**25**)**

purchase\_name **=** models**.***CharField***(**"Предмет закупки"**,** max\_length**=**150**)**

customer **=** models**.***CharField***(**"Заказчик"**,** max\_length**=**200**)**

price **=** models**.***CharField***(**"Начальная цена"**,** max\_length**=**150**)**

start **=** models**.***DateField***(**"Размещено"**)**

update **=** models**.***DateField***(**"Обновлено"**)**

end **=** models**.***DateField***(**"Окончание подачи заявок"**,** max\_length**=**25**)**

purchase\_link **=** models**.***URLField***(**"Ссылка на процедуру"**,** max\_length**=**150**)**

**def** \_\_str\_\_**(**self**)** **->** **str:**

**return** f"№ {self**.***number*} {self**.***end*} {self**.***purchase\_name*}"

# Проверка на "срок годности"

**def** end\_submission**(**self**)** **->** **bool:**

delta\_days **=** 90

**return** datetime**.***datetime***.***fromisoformat***(str(**self**.***end***))** **>=** **(**

datetime**.***datetime***.***today***()** **-** datetime**.***timedelta***(**days**=**delta\_days**)**

**)**

**class** **Meta:**

verbose\_name **=** "Процедуру"

verbose\_name\_plural **=** "Архив"

В файле описаны две модели: NewProcedures и Archive

Модель NewProcedures отвечает за формирование в базе данных таблицы с новыми процедурами. Имеет следующие поля:

- method: способ определения поставщика.

- number: номер процедуры.

- purchase\_name: предмет закупки.

- customer: заказчик.

- price: начальная цена.

- start: дата размещения.

- update: дата обновления (внесение изменений) процедуры.

- end: дата окончания подачи заявок.

- purchase\_link: ссылка на процедуру.

Определены метаданные для отображения в админке:

- Единичное verbose\_name и множественное verbose\_name\_plural.

Метод \_\_str\_\_ возвращает строку с основными данными о процедуре.

Модель Archive - хранит информацию об всех добавленных процедурах. Поля аналогичны классу NewProcedures.

- Метод \_\_str\_\_ отдаёт строку с номером, датой окончания подачи заявок и предметом закупки.

- Метод end\_submission() проверяет, прошел ли срок подачи заявок с использованием срока в 90 дней (может быть изменено).

- Также определены метаданные для отображения в админке.

Эти модели (классы) позволяют удобно управлять и отслеживать закупочные процедуры в системе. Процедуры могут быть добавлены, обновлены и архивированы, что помогает поддерживать порядок в данных.

### Файл admin.py

Файл «admin.py» в Django используется для настройки административного интерфейса приложения. В этом файле можно зарегистрировать модели, настроить их отображение и добавить дополнительные функции, такие как фильтры и поиск.

**from** django**.***contrib* **import** admin

**from** **.***models* **import** NewProcedures**,** Archive

**from** import\_export **import** resources

**from** import\_export**.***admin* **import** ImportExportModelAdmin

# класс обработки данных

**class** **ProceduresResource(**resources**.***ModelResource***):**

**class** **Meta:**

model **=** NewProcedures

skip\_unchanged **=** **True**

report\_skipped **=** **True**

# вывод данных на странице

**class** **ProceduresAdmin(**ImportExportModelAdmin**):**

resource\_classes **=** **[**ProceduresResource**]**

admin**.***site***.***register***(**NewProcedures**,** ProceduresAdmin**)**

admin**.***site***.***register***(**Archive**)**

Импорт необходимых модулей:

- admin из django.contrib: предоставляет интерфейс для работы с моделями.

- .models: импорт моделей NewProcedures и Archive из текущего приложения.

- resources и ImportExportModelAdmin из import\_export: используются для импорта и экспорта данных.

Использование классов ProceduresResource и ProceduresAdmin представляет собой пример использования библиотеки django-import-export в приложении Django для обеспечения функционала импорта и экспорта данных из модели Django через административный интерфейс.

Регистрация моделей в админ-панели:

- NewProcedures регистрируется с использованием настроек из ProceduresAdmin.

- Модель Archive регистрируется без дополнительных настроек

Таким образом, код настраивает админку Django, позволяя удобно управлять данными процедур и архива с функциями импорта и экспорта.

### Файл views.py

Файл views.py в Django отвечает за обработку запросов и возвращение ответов.

Вот основные функции views.py:

\* Обработка запросов. Определяет, что делать с входящими HTTP-запросами (GET, POST и т.д.).

\* Логика приложения. Содержит бизнес-логику, связанную с обработкой данных.

\* Взаимодействие с моделями. Получает данные из базы данных через модели и передает их в шаблоны.

\* Отправка ответов. Формирует и возвращает HTTP-ответы (как страницы HTML, JSON и т.д.).

Таким образом, views.py является связующим звеном между моделями и интерфейсом пользователя

**import** datetime **as** dt

**import** requests **as** req

**from** bs4 **import** BeautifulSoup **as** bs

**from** django**.***shortcuts* **import** redirect**,** render

**from** **.***models* **import** Archive**,** NewProcedures

**from** user\_app**.***models* **import** User

**from** user\_procedures\_app**.***models* **import** Contract**,** ManagerProcedure

# Стартовая странциа

**def** index**(**request**):**

managers **=** User**.***objects***.filter(**is\_superuser**=False)**

user\_procedures **=** ManagerProcedure**.***objects***.all()**

contracts **=** Contract**.***objects***.all().***count***()**

context **=** **{**

"title"**:** "CRM 44 FZ"**,**

"content"**:** "Главная страница CRM"**,**

"managers"**:** managers**,**

"user\_procedures"**:** user\_procedures**,**

"contracts"**:** contracts**,**

**}**

**return** render**(**request**,** "procedures\_app/index.html"**,** context**)**

# страница с новыми закупками

**def** new**(**request**):**

context **=** **{**

"title"**:** "Новые закупки"**,**

"procedures"**:** NewProcedures**.***objects***.***order\_by***(**

"end"

**),** # Сортировка по возрастанию даты

**}**

**return** render**(**request**,** "procedures\_app/new.html"**,** context**)**

# страница со всеми закупками

**def** archive**(**request**):**

arch **=** Archive**.***objects***.all()**

**for** item **in** arch**:**

**if** **not** item**.***end\_submission***():**

item**.***delete***()**

context **=** **{**

"title"**:** "Архив процедур"**,**

"procedures"**:** Archive**.***objects***.***order\_by***(**"-end"**),** # Сортировка по убыванию даты

**}**

**return** render**(**request**,** "procedures\_app/archive.html"**,** context**)**

# страница инфо

**def** about**(**request**):**

**return** render**(**request**,** "procedures\_app/about.html"**)**

# функция собирает данные для обработки с сайта закупок

**def** get\_data**():**

HEADERS **=** **{**

"User-Agent"**:** "Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_10\_1) AppleWebKit/537.36 (KHTML, like Gecko) "

"Chrome/39.0.2171.95 Safari/537.36"

**}**

# сайт с настроенными фильтрами

URL **=** "https://zakupki.gov.ru/epz/order/extendedsearch/results.html?morphology=on&search-filter=%D0%94%D0%B0%' \

'D1%82%D0%B5+%D1%80%D0%B0%D0%B7%D0%BC%D0%B5%D1%89%D0%B5%D0%BD%D0%B8%D1%8F&pageNumber=1&sortDirection=false' \

'&recordsPerPage=\_10&showLotsInfoHidden=false&sortBy=UPDATE\_DATE&fz44=on&af=on&currencyIdGeneral=' \

'-1&okpd2IdsWithNested=on&okpd2Ids=8874055&okpd2IdsCodes=26.2&gws=' \

%D0%92%D1%8B%D0%B1%D0%B5%D1%80%D0%B8%D1%82%D0%B5+%D1%82%D0%B8%D0%BF+' \

%D0%B7%D0%B0%D0%BA%D1%83%D0%BF%D0%BA%D0%B8&OrderPlacementSmallBusinessSubject=on&OrderPlacementRnpData=' \

on&OrderPlacementExecutionRequirement=on&orderPlacement94\_0=0&orderPlacement94\_1=0&orderPlacement94\_2=0"

**try:**

page **=** req**.***get***(**URL**,** headers**=**HEADERS**)**

**except** **Exception** **as** e**:**

**return** **None**

**if** page**:** # возвращает True если page.status\_code == 200

soup **=** bs**(**

page**.***text***,** "lxml"

**)** # "lxml" установлен вместо "html.parser" для улучшения обработки

data **=** soup**.***find\_all***(**"div"**,** class\_**=**"row no-gutters registry-entry\_\_form mr-0"**)**

**return** data

**return** **None**

# Обрабатка данных и формирует словарь

**def** add\_data\_dict**(**data**)** **->** **dict:**

bd **=** **{}**

**id** **=** 1

**for** item **in** data**:**

# Способ определния поставщика

method **=** " "**.***join***(**

item**.***find***(**

"div"**,**

class\_**=**"col-9 p-0 registry-entry\_\_header-top\_\_title text-truncate"**,**

**)**

**.***text***.***replace***(**"\n"**,** ""**)**

**.***split***()**

**)**

# номер процедуры

number **=** **(**

item**.***find***(**"div"**,** class\_**=**"registry-entry\_\_header-mid\_\_number"**)**

**.***text***.***replace***(**"\n"**,** ""**)**

**.***replace***(**"№"**,** ""**)**

**.***strip***()**

**)**

# предмет закупки

purchase\_name **=** **(**

item**.***find***(**"div"**,** class\_**=**"registry-entry\_\_body-value"**)**

**.***text***.***replace***(**"\n"**,** ""**)**

**.***strip***()**

**)**

# заказчик

customer **=** **(**

item**.***find***(**"div"**,** class\_**=**"registry-entry\_\_body-href"**)**

**.***text***.***replace***(**"\n"**,** ""**)**

**.***strip***()**

**)**

# начальная цена

price **=** **(**

item**.***find***(**"div"**,** class\_**=**"price-block\_\_value"**).***text***.***replace***(**"\n"**,** ""**).***strip***()**

**)**

# Даты

start**,** update**,** end **=** item**.***find\_all***(**"div"**,** class\_**=**"data-block\_\_value"**)**

# Размещено

start **=** dt**.***datetime***.***strptime***(**start**.***text***,** "%d.%m.%Y"**).***date***()**

# Обновлено

update **=** dt**.***datetime***.***strptime***(**update**.***text***,** "%d.%m.%Y"**).***date***()**

# Окончание подачи заявок

end **=** dt**.***datetime***.***strptime***(**end**.***text***,** "%d.%m.%Y"**).***date***()**

# получаем ссылку на закупку по номеру

url\_data **=** item**.***find***(**"div"**,** class\_**=**"registry-entry\_\_header-mid\_\_number"**)**

purchase\_link **=** "https://zakupki.gov.ru" **+** url\_data**.***find***(**

"a"**,** target**=**"\_blank"

**).***get***(**"href"**)**

bd**[id]** **=** **{**

"method"**:** method**,**

"number"**:** number**,**

"purchase\_name"**:** purchase\_name**,**

"customer"**:** customer**,**

"price"**:** price**,**

"start"**:** start**,**

"update"**:** update**,**

"end"**:** end**,**

"purchase\_link"**:** purchase\_link**,**

**}**

**id** **+=** 1

**return** bd

# добавление новых закупок на страницу

**def** update\_procedures**(**request**):**

bd **=** add\_data\_dict**(**get\_data**())** # формируем новый словарь с закупками

NewProcedures**.***objects***.all().***delete***()** # убираем закупки из БД

**if** request**.***method* **==** "POST"**:**

**for** k**,** v **in** bd**.***items***():**

\_**,** created **=** NewProcedures**.***objects***.***get\_or\_create***(**

**id=**k**,**

method**=**v**[**"method"**],**

number**=**v**[**"number"**],**

purchase\_name**=**v**[**"purchase\_name"**],**

customer**=**v**[**"customer"**],**

price**=**v**[**"price"**],**

start**=**v**[**"start"**],**

update**=**v**[**"update"**],**

end**=**v**[**"end"**],**

purchase\_link**=**v**[**"purchase\_link"**],**

**)**

update\_archive**()** # добавляем все новые закупки в общий архив

**return** redirect**(**"procedures\_app:new"**)**

# для добавления закупок в архив

**def** update\_archive**():**

procedures **=** NewProcedures**.***objects***.all()**

**for** item **in** procedures**:**

**if** item**.***number* **not** **in** Archive**.***objects***.all():**

Archive**.***objects***.***get\_or\_create***(**

method**=**item**.***method***,**

number**=**item**.***number***,**

purchase\_name**=**item**.***purchase\_name***,**

customer**=**item**.***customer***,**

price**=**item**.***price***,**

start**=**item**.***start***,**

update**=**item**.***update***,**

end**=**item**.***end***,**

purchase\_link**=**item**.***purchase\_link***,**

**)**

Рассмотрим логику кода подробнее:

В первую очередь выполняется **импорт библиотек и моделей** для выполнения веб-запросов и парсинга данных, а также для работы с фреймворком Django.

- datetime as dt: Импортирует модуль для работы с датой и временем.

- requests as req: Импортирует библиотеку для выполнения HTTP-запросов.

- BeautifulSoup as bs: Импортирует класс для парсинга HTML и XML документов.

- redirect, render из django.shortcuts: Импортирует функции для перенаправления и

Импорт моделей:

- Archive, NewProcedures: Модели приложения, которые хранят архивные и новые процедуры.

- User: Модель пользователя из приложения user\_app.

- Contract, ManagerProcedure: Модели, связанные с пользователями и процедурами, из приложения user\_procedures\_app.

**Функция «def index(request)»** формирует главную страницу CRM-системы, собирая необходимые данные и передавая их в шаблон для отображения.

- managers = User.objects.filter(is\_superuser=False): Получает всех пользователей, которые не являются суперпользователями (то есть обычные менеджеры).

- user\_procedures = ManagerProcedure.objects.all(): Извлекает все объекты процедур, связанные с менеджерами.

- contracts = Contract.objects.all().count(): Получает общее количество контрактов в базе данных.

context: Здесь создается словарь с информацией, которая будет передана в шаблон:

- "title": Название страницы.

- "content": Основное содержание для главной страницы.

- "managers": Список менеджеров.

- "user\_procedures": Список процедур.

- "contracts": Общее количество контрактов.

- return render(request, "procedures\_app/index.html", context): Возвращает отрендеренный (отрисовыванный) HTML-шаблон index.html с переданным контекстом.

**Функция def new(request)** отвечает за обработку HTTP-запроса для отображения страницы с новыми закупками. Генерирует список процедур, полученный из модели NewProcedures, отсортированный по возрастанию значения поля и возвращает рендеринг (отрисовку) HTML-шаблона new.html, передавая в него созданный контекст.

**Функция def archive(request)** отвечает за обработку HTTP-запроса для отображения страницы с архивными закупками. Запрашивает все объекты модели Archive с помощью Archive.objects.all() и сохраняет их в переменной arch. В цикле проходит по каждому элементу в arch. Если метод end\_submission() возвращает False, этот элемент удаляется из базы данных с помощью item.delete(). Создает словарь context, в который добавляет заголовок для шаблона "Архив процедур" и список procedures — все объекты Archive, отсортированные по убыванию даты end (подачи заявок). Возвращает ответ с рендерингом шаблона archive.html, передавая контекст.

**Функция def about(request)** отвечает за обработку HTTP-запроса для отображения страницы с контактной информацией (шаблон about.html).

**Функция def get\_data()** предназначена для извлечения данных с сайта zakupki.gov.ru с использованием библиотеки requests для HTTP-запросов и BeautifulSoup для парсинга HTML. Заголовок User-Agent, имитирует браузер при отправке запроса на сайт. В переменную URL передаем адрес сайта с заранее настроенными фильтрами для поиска. Далее осуществляется проверка GET-запроса на указанный URL. При успешном запросе создается объект soup для парсинга HTML-кода страницы, используя библиотеку lxml. Далее в объект data попадают нужные нам данные из soup. Таким образом, функция предназначена для сбора и возврата определенной информации с указанного веб-сайта.

**Функция add\_data\_dict(data)** принимает объект data, Создает пустой словарь bd и переменную id, начиная с 1, для уникальной идентификации каждой записи. Затем в цикле для каждого элемента в переданных данных (data). Извлекает информацию о поставщике, номере процедуры, предмете закупки, заказчике и начальной цене из HTML-структуры. Преобразует даты (начальная, обновленная и окончательная) в формат даты, используя datetime. Формирует ссылку на закупку. Добавляет извлеченные данные в словарь bd, где ключ - уникальный идентификатор id, а значение - другой словарь с информацией о закупке. После обработки всех элементов возвращает словарь bd со всеми записями.

**Функция update\_procedures(request)** осуществляет добавление новых закупок в модель NewProcedures. С помощью функции add\_data\_dict(get\_data()) формируется новый словарь с данными о закупках (bd). Все записи из модели NewProcedures удаляются с помощью NewProcedures.objects.all().delete(). Это необходимо для того, чтобы обновить информацию о закупках. Если запрос является POST-запросом, функция перебирает элементы словаря bd. Для каждого элемента используется метод get\_or\_create, который пытается найти или создать новую запись в базе данных с указанным идентификатором (id=k) и другими атрибутами (метод, номер, название закупки и т. д.). После добавления данных о новых закупках вызывается функция update\_archive(), которая, обновляет архив с закупками. В конце происходит перенаправление на страницу с новыми закупками. Таким образом, функция обновляет информацию о закупках в базе данных новых закупок и архиве.

**Функция def update\_archive()** осуществляет добавление новых закупок в модель Archive

### Файл urls.py

Этот код определяет маршруты URL для Django-приложения с именем procedures\_app.

**from** django**.***urls* **import** path

**from** **.** **import** views

app\_name **=** "procedures\_app"

urlpatterns **=** **[**

path**(**""**,** views**.***index***,** name**=**"index"**),** # домашняя страница

path**(**"about/"**,** views**.***about***,** name**=**"about"**),** # о нас

path**(**"new/"**,** views**.***new***,** name**=**"new"**),** # новые закупки

path**(**"update\_procedures/"**,** views**.***update\_procedures***,** name**=**"update\_procedure"**),** # обновление закупок

path**(**"archive/"**,** views**.***archive***,** name**=**"archive"**),** # архив

**]**

Сначала мы импортируем функцию path для создания маршрутов и views для подключения представлений, которые будут обрабатывать запросы. Указываем имя приложения, чтобы разделять пространства имен для URL-адресов. Это полезно при использовании reverse или include.

Определяем маршруты:

- "": Корневой маршрут, который связывается с представлением index и имеет имя index.

- "about/": Маршрут для страницы "О нас", связывается с представлением about.

- "new/": Маршрут для отображения новых закупок, связывается с new.

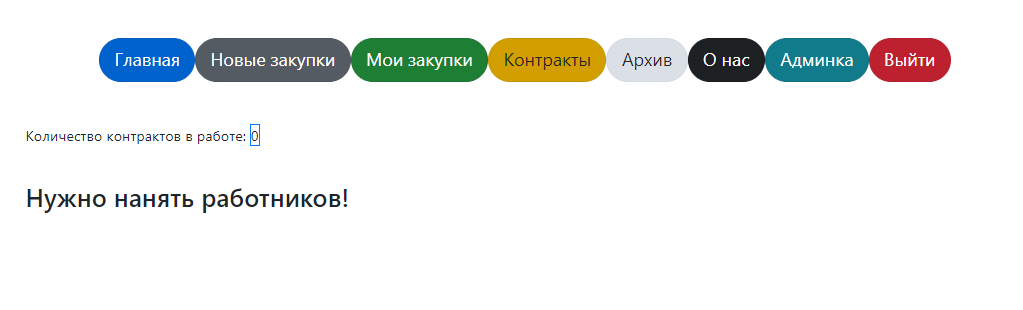
- "update\_procedures/": Маршрут для обновления закупок, связывается с update\_procedures.

- "archive/": Маршрут для доступа к архиву, связывается с archive.

Таким образом, каждый маршрут связан с соответствующим представлением, которое будет обрабатывать HTTP-запросы и возвращать необходмые страницы или данные.

### Директория с шаблонами html страниц (templates\procedures\_app)

#### Стартовая страница (index.html)



**{% extends "base.html" %}**

**{% block title %}{{title}}{% endblock %}**

**{% block content %}**

<div>

<p>**Количество контрактов в работе:** <span class=**"border border-primary"**>**{{ contracts }}**</span></p>

</div>

<br>

<div>

**{%if managers %}**

<table class=**"table table-striped table-hover text-center"**>

<thead>

<tr class=**"table-secondary"**>

<th style=**"width: 34%"**>**Сотрудник**</th>

<th style=**"width: 22%"**>**Процедур в работе**</th>

</tr>

</thead>

<tbody>

**{% for el in managers %}**

<tr>

<th scope=**"row"** class=**"text-start"**>**{{ el }}**</th>

<td>

**{{ el.managerprocedure\_set.all|length }}**

</td>

</tr>

**{% endfor %}**

**{% else %}**

<h3>**Нужно нанять работников!**</h3>

**{% endif %}**

</tbody>

</table>

</div>

**{% endblock %}**

Расширение базового шаблона - {% extends "base.html" %} указывает, что данный шаблон наследует содержимое из base.html о котором было упомянуто ранее.

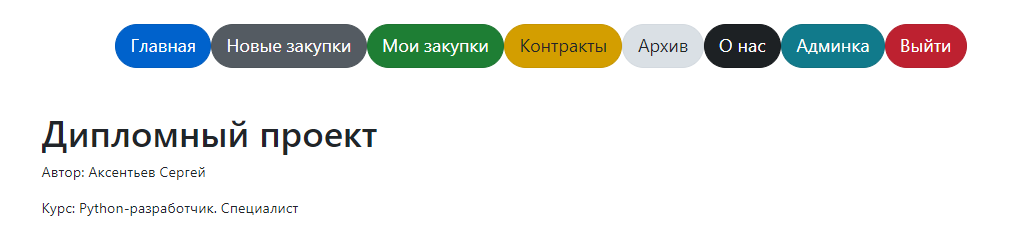
{% block title %}{{title}}{% endblock %} – определяем блок title, который будет заменен на значение переменной title, переданное из представления.

Далее в блоке с контентом страницы выводится количество контрактов, которое передается из представления.

Если есть информация о менеджерах, то для каждого менеджера выводится его имя и количество связанных процедур.

По итогу код создает страницу с показом информации о процедурах менеджеров и количестве действий по контрактам. Если менеджеры есть, выводится таблица с их данными; если нет — сообщение с предложением нанять работников.

#### Страница контактов (about.html)



Страница предназначена для контактной информации о компании и т.д. В данном проекте оставлена информация обо мне.

**{% extends "base.html" %}**

**{% block title %} Информация {% endblock %}**

**{% block content %}**

<h1>**Дипломный проект**</h1>

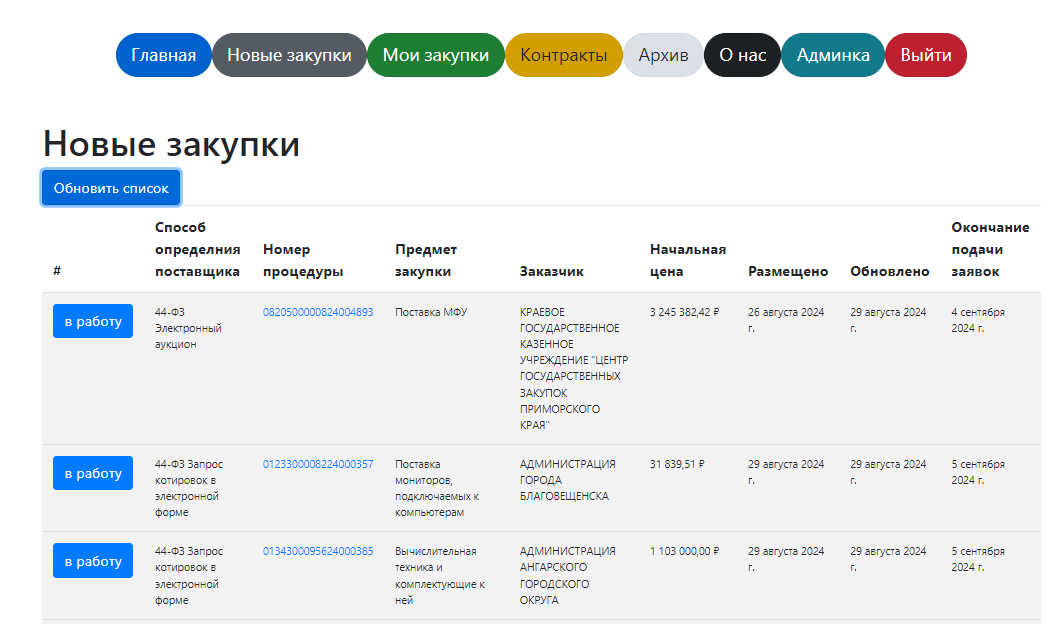
<p>**Автор: Аксентьев Сергей**</p>

<p>**Курс: Python-разработчик. Специалист**</p>

**{% endblock %}**

Шаблон так же наследуется от базового с изменением блока title и блока content.

#### Страница с новыми закупками (new.html)



Шаблон позволяет динамически отображать и обновлять список закупок с соответствующей формой для управления.

**{% extends "base.html" %}**

**{% block title %} {{ title }} {% endblock %}**

**{% block content %}**

<h1>**{{ title }}**</h1>

<form action=**"{% url 'procedures\_app:update\_procedure' %}"** method=**"post"**>

**{% csrf\_token %}**

<button class=**"btn btn-primary"** type=**"submit"**>**Обновить список**</button>

</form>

**{% if procedures %}**

<div class=**"table-responsive"**>

<table class=**"table table-striped"**>

<thead class=**"thead--light"**>

<tr>

<th scope=**"col"**>**#**</th>

<th scope=**"col"**>**Способ определния поставщика**</th>

<th scope=**"col"**>**Номер процедуры**</th>

<th scope=**"col"**>**Предмет закупки**</th>

<th scope=**"col"**>**Заказчик**</th>

<th scope=**"col"**>**Начальная цена**</th>

<th scope=**"col"**>**Размещено**</th>

<th scope=**"col"**>**Обновлено**</th>

<th scope=**"col"**>**Окончание подачи заявок**</th>

</tr>

</thead>

**{% for el in procedures %}**

<tbody style=**"font-size: 75%"** ;>

<tr>

<td>

<form action=**"{% url 'user\_procedures\_app:add\_procedure\_to\_user' el.id %}"** method=**"post"**>

**{% csrf\_token %}**

<input class=**"btn btn-primary"** type=**"submit"** value=**"в работу"** />

</form>

</td>

<td>**{{el.method}}**</td>

<td><a href=**"{{el.purchase\_link}}"**>**{{el.number}}**</a></td>

<td>**{{el.purchase\_name}}**</td>

<td>**{{el.customer}}**</td>

<td>**{{el.price}}**</td>

<td>**{{el.start}}**</td>

<td>**{{el.update}}**</td>

<td>**{{el.end}}**</td>

</tr>

**{% endfor %}**

</tbody>

</table>

</div>

**{% else %}**

<h3>**Нет новых закупок**</h3>

**{% endif %}**

**{% endblock %}**

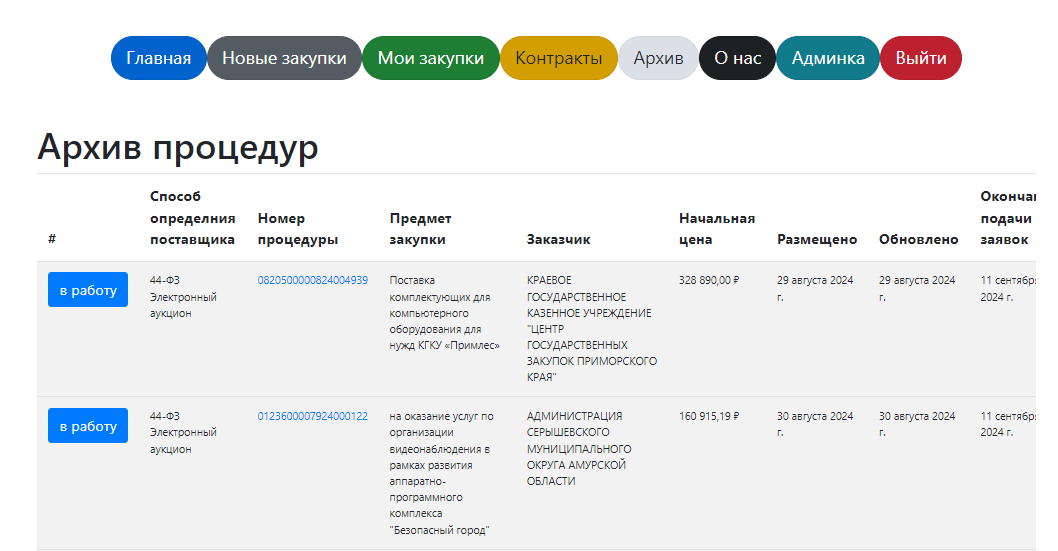
Шаблон так же наследуется от базового с изменением блока title и блока content.

На странице расположена кнопка «Обновить список», которая вызывает функцию update\_procedure, для обновления списка новых опубликованных процедур (форма, отправляющая POST-запрос для обновления списка процедур, защищенная токеном CSRF).

Далее выводится таблица с найденными процедурами. Заголовки колонок предоставляют информацию о различных аспектах каждой процедуры. Строки же формируются путем перебора всех элементов в procedures (передана из предстваления). Для каждой процедуры создается строка таблицы, содержащая информацию о методе определения поставщика, номере процедуры и другой важной информации. Также есть форма для добавления процедуры в работу, то есть нажимая данную кнопку менеджер берет данную процедуру себе в работу.

Если список процедур пуст, выводится сообщение о том, что новых закупок нет.

#### Страница с архивом процедур archive.html



**{% extends "base.html" %}**

**{% block title %}{{ title }} {% endblock %}**

**{% block content %}**

<h1>**{{ title }}**</h1>

**{% if procedures %}**

<div class=**"table-responsive"**>

<table class=**"table table-striped"**>

<thead class=**"thead--light"**>

<tr>

<th scope=**"col"**>**#**</th>

<th scope=**"col"**>**Способ определения поставщика**</th>

<th scope=**"col"**>**Номер процедуры**</th>

<th scope=**"col"**>**Предмет закупки**</th>

<th scope=**"col"**>**Заказчик**</th>

<th scope=**"col"**>**Начальная цена**</th>

<th scope=**"col"**>**Размещено**</th>

<th scope=**"col"**>**Обновлено**</th>

<th scope=**"col"**>**Окончание подачи заявок**</th>

</tr>

</thead>

**{% for el in procedures %}**

<tbody style=**"font-size: 75%"** ;>

<tr>

<td>

<form action=**"{% url 'user\_procedures\_app:add\_procedure\_to\_user' el.id %}"** method=**"post"**>

**{% csrf\_token %}**

<input class=**"btn btn-primary"** type=**"submit"** value=**"в работу"** />

</form>

</td>

<td>**{{el.method}}**</td>

<td><a href=**"{{el.purchase\_link}}"**>**{{el.number}}**</a></td>

<td>**{{el.purchase\_name}}**</td>

<td>**{{el.customer}}**</td>

<td>**{{el.price}}**</td>

<td>**{{el.start}}**</td>

<td>**{{el.update}}**</td>

<td>**{{el.end}}**</td>

</tr>

**{% endfor %}**

</tbody>

</table>

</div>

**{% else %}**

<h3>**Нет закупок**</h3>

**{% endif %} {% endblock %}**

Страница аналогична предыдущей, за исключением кнопки добавления новых процедур, т.к. они добавляются автоматически при обновлении новых процедур и удаляются из базы данных по истечении 90 дней. В остальном так же отображается таблица с данными о процедурах и кнопкой взять в работу.

### Директория с миграциями (migrations)

Директория migrations в приложении Django используется для управления изменениями в базе данных, связанными с моделями. Она содержит файлы миграций, которые отслеживают изменения в структуре базы данных (например, добавление или изменение полей в моделях).

Данная директория создается автоматически при миграции моделей (создании таблиц в базе данных) с параметрами, заданными при разработке приложения.

## Приложение user\_app

Приложение user\_app отвечает за регистрацию новых пользователей, отображение информации о существующих пользователях, а также их настройку

### Файл apps.py

**from** django**.***apps* **import** AppConfig

**class** **UserAppConfig(**AppConfig**):**

default\_auto\_field **=** 'django.db.models.BigAutoField'

name **=** 'user\_app'

verbose\_name **=** 'Сотрудники'

Описание полей:

- default\_auto\_field: устанавливает тип автоматически создаваемого идентификатора для моделей. В данном случае используется BigAutoField, что позволяет использовать более широкий диапазон значений для ID.

- name: указывает имя приложения в Django. Здесь оно обозначено как "user\_app".

- verbose\_name: определяет понятное и человеко-читаемое имя приложения, которое будет отображаться в админ-панели и других местах. В данном случае это " Сотрудники ".

### Файл models.py

**from** django**.***db* **import** models

**from** django**.***contrib***.***auth***.***models* **import** AbstractUser

**class** **User(**AbstractUser**):**

# Добавление поля для номера телефона в модель User

phone\_number **=** models**.***CharField***(**max\_length**=**16**,** blank**=True,** null**=True)**

**class** **Meta:**

db\_table **=** "user"

verbose\_name **=** "Сотрудника"

verbose\_name\_plural **=** "Сотрудники"

**def** \_\_str\_\_**(**self**):**

**return** f"{self**.***first\_name*} {self**.***last\_name*}"

В данном случае создаем настраиваемую модель пользователя в Django, наследуя от стандартной модели AbstractUser.

Импорты:

- models используется для создания моделей в Django.

- AbstractUser — это базовая модель пользователя, которая предоставляет стандартные поля и функциональность.

- **Класс User** наследует от AbstractUser, что позволяет нам использовать все стандартные поля (такие как имя, фамилия, пароль и т. д.) с добавлением собственных..

- phone\_number — это новое поле, добавленное для хранения номера телефона пользователя.

- CharField используется, чтобы сохранить строку фиксированной длины (максимум 16 символов).

- blank=True и null=True означают, что поле может быть пустым.

В классе Meta - db\_table — устанавливает имя таблицы в базе данных, - verbose\_name и verbose\_name\_plural определяют пользовательские имена для модели в админке Django.

Это расширение модели пользователя полезно для хранения дополнительных данных, таких как номер телефона, при этом сохраняя все стандартные функции управления пользователями из Django.

### Файл admin.py

В этом файле прописаны настройки для настройки административной панели Django для модели User.

**from** django**.***contrib* **import** admin

**from** **.***models* **import** User

**from** user\_procedures\_app**.***admin* **import** UserProcedureTabAdmin

# Таблица отображения сотрудников

*@admin***.***register***(**User**)**

**class** **UserAdmin(**admin**.***ModelAdmin***):**

# список отображения полей

list\_display **=** **[**

"username"**,**

"first\_name"**,**

"last\_name"**,**

"email"**,**

**]**

# поле поиска

search\_fields **=** **[**

"username"**,**

"first\_name"**,**

"last\_name"**,**

"email"**,**

**]**

# настрока отображения полей конкретного сотруднка

fieldsets **=** **[**

**(**

**None,**

**{**

"classes"**:** **[**"wide"**],** # будет занимать все доступное место на странице

"fields"**:** **[**

"first\_name"**,**

"last\_name"**,**

**],**

**},**

**),**

**(**

"Контакты"**,**

**{**

"classes"**:** **[**"collapse"**],** # будет скрыто в раскрывающемся меню

"fields"**:** **[**"username"**,** "email"**,** "phone\_number"**],**

**},**

**),**

**(**

"Группы и разрешения"**,**

**{**

"fields"**:** **[**"groups"**,** "user\_permissions"**],**

**},**

**),**

**(**

"Статусы"**,**

**{**

"fields"**:** **[**"is\_active"**,** "is\_staff"**,** "is\_superuser"**],**

**},**

**),**

**(**

**None,**

**{**

"fields"**:** **[**"last\_login"**,** "date\_joined"**],**

**},**

**),**

**]**

filter\_horizontal **=** **(**"user\_permissions"**,)**

# встраиваем таблицу отображения взятых в работу закупок, с возможностью добавления новой закупки

inlines **=** **[**UserProcedureTabAdmin**]**

Сначала импортируем необходимые модули. admin из Django, модель User из текущего приложения и UserProcedureTabAdmin для встраивания внутри админки.

Далее прописаны настройки для админской панели. Регистрируется модель User в админской панели и создается для нее класс UserAdmin, унаследованный от admin.ModelAdmin.

Настройка отображения полей:

- list\_display определяет, какие поля будут видны в списке пользователей. Здесь отображаются имя пользователя, имя, фамилия и адрес электронной почты.

- search\_fields позволяет пользователю искать записи по указанным полям.

- fieldsets используется для организации полей на странице редактирования пользователя.

- filter\_horizontal добавляет удобный интерфейс для настройки разрешений.

- В inlines мы включаем возможность редактирования связанных объектов (закупок) — в данном случае, структур данных из UserProcedureTabAdmin.

В итоге этот код делает админскую панель более удобной и функциональной, позволяя легко управлять пользователями и их данными.

### Файл views.py

Отвечает за аутентификацию пользователей в приложении и включает в себя представления для входа, регистрации и выхода из системы.

**from** django**.***contrib***.***auth***.***decorators* **import** login\_required

**from** django**.***shortcuts* **import** redirect

**from** django**.***contrib***.***auth***.***views* **import** LoginView

**from** django**.***views***.***generic* **import** CreateView

**from** **.***forms* **import** UserLoginForm**,** UserRegistrationForm

**from** django**.***contrib* **import** auth**,** messages

# Обработка Login

**class** **UserLoginView(**LoginView**):**

template\_name **=** "user\_app/login.html"

form\_class **=** UserLoginForm

**def** form\_valid**(**self**,** form**):**

user **=** form**.***get\_user***()**

**if** user**:**

auth**.***login***(**self**.***request***,** user**)**

**return** redirect**(**"procedures\_app:index"**)**

**def** get\_context\_data**(**self**,** **\*\***kwargs**):**

context **=** **super().***get\_context\_data***(\*\***kwargs**)**

context**[**"title"**]** **=** "Авторизация"

**return** context

**class** **UserRegistrationView(**CreateView**):**

template\_name **=** "user\_app/registration.html"

form\_class **=** UserRegistrationForm

# Если форма заполнена верно, сразу сохраняем в БД и выполняем вход

**def** form\_valid**(**self**,** form**):**

user **=** form**.***instance*

**if** user**:**

form**.***save***()**

auth**.***login***(**self**.***request***,** user**)**

**return** redirect**(**"procedures\_app:index"**)**

# добавляем в контекст название страницы "Регистрация"

**def** get\_context\_data**(**self**,** **\*\***kwargs**):**

context **=** **super().***get\_context\_data***(\*\***kwargs**)**

context**[**"title"**]** **=** "Регистрация"

**return** context

*@login\_required*

**def** logout**(**request**):**

messages**.***success***(**request**,** f"{request**.***user***.***username*}, Вы вышли из аккаунта"**)**

auth**.***logout***(**request**)**

**return** redirect**(**"procedures\_app:index"**)**

Импорт необходимых модулей. Импортируем декоратор для проверки авторизации, функции для перенаправления, представления для входа и создания пользователей, а также формы и модули для работы с аутентификацией и сообщениями.

**class UserLoginView(LoginView)** - Представление для входа:

- UserLoginView: Наследуется от LoginView и обрабатывает вход пользователей.

- template\_name: Указывает, какой шаблон использовать.

- form\_class: Задает форму для входа.

- form\_valid: Если форма валидна, пользователь аутентифицируется и перенаправляется на главную страницу.

- get\_context\_data: Добавляет заголовок "Авторизация" в контекст шаблона.

**class UserRegistrationView(CreateView)** - Представление для регистрации:

- UserRegistrationView: Создает нового пользователя через форму регистрации.

- form\_valid: Если форма заполнена успешно, пользователь сохраняется в БД и аутентифицируется.

- get\_context\_data: Добавляет заголовок "Регистрация" в контекст шаблона.

**def logout(request)** - Выход из системы

- logout: Декорирован @login\_required, гарантирует, что только авторизованные пользователи могут выйти из системы.

- Сообщение: После выхода выводится сообщение об успешном выходе.

- Перенаправление: Пользователь перенаправляется на главную страницу.

### Файл forms.py

Файл forms.py в Django используется для создания форм и обработки данных, которые отправляются пользователю.

**from** django **import** forms

**from** django**.***contrib***.***auth***.***forms* **import** **(**

AuthenticationForm**,**

UserCreationForm**,**

**)**

**from** **.***models* **import** User

**class** **UserLoginForm(**AuthenticationForm**):**

**class** **Meta:**

model **=** User

username **=** forms**.***CharField***(**

label**=**"Логин"**,**

widget**=**forms**.***TextInput***(**

attrs**={**

"autofocus"**:** **True,**

"class"**:** "form-control"**,**

"placeholder"**:** "Введите ваше имя пользователя"**,**

**}**

**),**

**)**

password **=** forms**.***CharField***(**

label**=**"Пароль"**,**

widget**=**forms**.***PasswordInput***(**

attrs**={**

"autocomplete"**:** "current-password"**,**

"class"**:** "form-control"**,**

"placeholder"**:** "Введите ваш пароль"**,**

**}**

**),**

**)**

**class** **UserRegistrationForm(**UserCreationForm**):**

**class** **Meta:**

model **=** User

fields **=** **(**

"username"**,**

"first\_name"**,**

"last\_name"**,**

"email"**,**

"password1"**,**

"password2"**,**

**)**

username **=** forms**.***CharField***(**

label**=**"Логин"**,**

widget**=**forms**.***TextInput***(**

attrs**={**

"class"**:** "form-control"**,**

"placeholder"**:** "Введите ваше имя пользователя"**,**

**}**

**),**

**)**

first\_name **=** forms**.***CharField***(**

label**=**"Имя"**,**

widget**=**forms**.***TextInput***(**

attrs**={**

"class"**:** "form-control"**,**

"placeholder"**:** "Введите ваше имя"**,**

**}**

**),**

**)**

last\_name **=** forms**.***CharField***(**

label**=**"Фамилия"**,**

widget**=**forms**.***TextInput***(**

attrs**={**

"class"**:** "form-control"**,**

"placeholder"**:** "Введите вашу фамилию"**,**

**}**

**),**

**)**

email **=** forms**.***CharField***(**

widget**=**forms**.***EmailInput***(**

attrs**={**

"class"**:** "form-control"**,**

"placeholder"**:** "Введите ваш email \*youremail@example.com"**,**

**}**

**)**

**)**

password1 **=** forms**.***CharField***(**

label**=**"Пароль"**,**

widget**=**forms**.***PasswordInput***(**

attrs**={**

"class"**:** "form-control"**,**

"placeholder"**:** "Введите ваш пароль"**,**

**}**

**),**

**)**

password2 **=** forms**.***CharField***(**

label**=**"Подтверждение пароля"**,**

widget**=**forms**.***PasswordInput***(**

attrs**={**

"class"**:** "form-control"**,**

"placeholder"**:** "Поддтвердите ваш пароль"**,**

**}**

**),**

**)**

Данный код создает формы для аутентификации и регистрации пользователей.

Импорт необходимых модулей: импортируем необходимые классы из Django: формы и модель пользователя, чтобы создать свои собственные формы аутентификации и регистрации.

**UserLoginForm(AuthenticationForm)** - Форма ввода для аутентификации (логин):

- username (логин) и password (пароль) представлены как поля с необходимыми атрибутами, такими как class и placeholder для стилизации.

**class UserRegistrationForm(UserCreationForm) -** Форма регистрации нового пользователя:

- Meta указывает, что данная форма будет связана с моделью User и определяет поля, которые будут использоваться в форме. Каждое поле аналогично полям логина, но с дополнительной информацией. Все поля имеют свои атрибуты, чтобы улучшить пользовательский интерфейс, используя классы Bootstrap.

### Файл urls.py

Определяет маршруты (URLs) для приложения Django, связанного с управлением пользователями.

**from** django**.***urls* **import** path

**from** **.** **import** views

app\_name **=** "user\_app"

urlpatterns **=** **[**

path**(**"login/"**,** views**.***UserLoginView***.***as\_view***(),** name**=**"login"**),**

path**(**"registration/"**,** views**.***UserRegistrationView***.***as\_view***(),** name**=**"registration"**),**

path**(**"logout/"**,** views**.***logout***,** name**=**"logout"**),**

**]**

Импорт необходимых модулей: функция path для создания маршрутов, представления (views) из текущего модуля (.).

Определили пространство имен для приложения, что позволяет ссылаться на маршруты через user\_app:login и т.д.

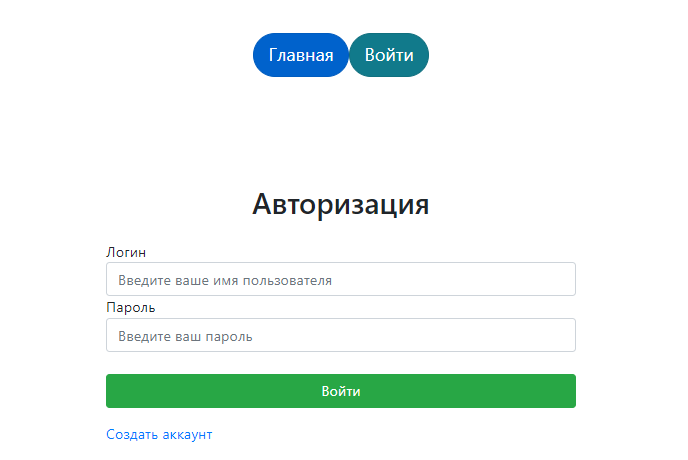
- path("login/", views.UserLoginView.as\_view(), name="login"): маршрут для входа пользователя. Используется класс UserLoginView для обработки запроса.

- path("registration/", views.UserRegistrationView.as\_view(), name="registration"): маршрут для регистрации нового пользователя. Обрабатывается классом UserRegistrationView.

- path("logout/", views.logout, name="logout"): маршрут для выхода пользователя. В этом случае используется функция logout из представлений.

### Директория с шаблонами html страниц (templates\user\_app)

#### login.htm



**{% extends "base.html" %}**

**{% block title %} {{ title }} {% endblock %}**

**{% block content %}**

<div class=**"row"**>

<div class=**"container mt-5"**>

<div class=**"row justify-content-center"**>

<div class=**"col-md-6 bg-white p-4 mb-4 mx-3 rounded custom-shadow"**>

<h2 class=**"text-center mb-4"**>**{{ title }}**</h2>

<form action=**"{% url 'user\_app:login' %}"** method=**"post"**>

**{% csrf\_token %}**

**{% if form.non\_field\_errors %}**

<div class=**"alert alert-danger"** role=**"alert"**>

**{{ form.non\_field\_errors }}**

</div>

**{% endif %}**

**{% for field in form %}**

**{{ field.label }}**

**{{ field }}**

**{% endfor %}**

<br>

<button type=**"submit"** class=**"btn btn-success btn-block"**>**Войти**</button>

</form>

<div class=**"mt-3"**>

<a href=**"{% url "**user\_app:registration" %}**">Создать аккаунт</a>**

**</div>**

**</div>**

**</div>**

**</div>**

**</div>**

**{% endblock %}**

Этот код представляет собой шаблон для страницы входа (авторизации) пользователя.

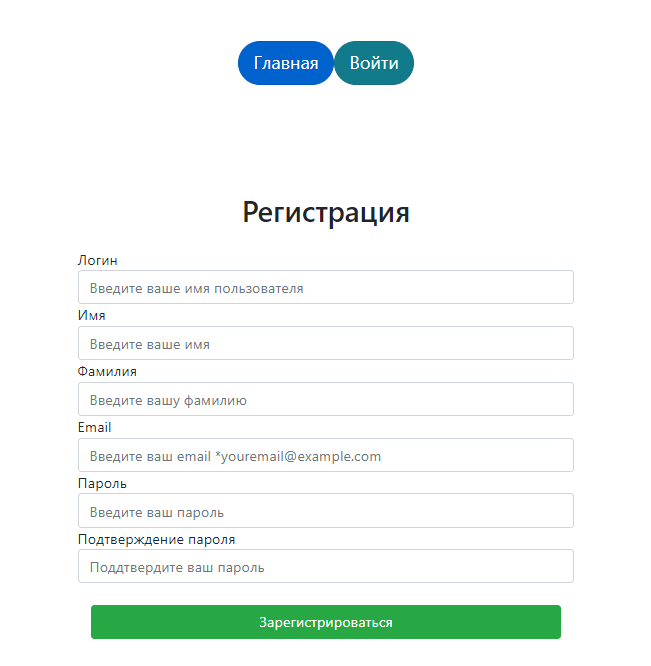
Данный шаблон наследует содержимое из base.html. Это позволяет использовать общие элементы оформления, определенные в базовом шаблоне. Создаем блок для заголовка страницы. Значение {{ title }} будет заменено на конкретное название страницы, переданное из представления.

В блоке контента создаётся форма, которая отправляется на URL, соответствующий представлению для входа (user\_app:login. Здесь также добавлен токен CSRF для защиты от атак. Если есть ошибки ввода, они будут отображаться в виде красного уведомления.

Внизу под формой добавляем ссылку для перехода на страницу регистрации нового аккаунта.

Как итог этот шаблон создает страницу для входа пользователя с обработкой ошибок, использованием форм и с возможностью перехода на страницу регистрации. Он оформлен с помощью Bootstrap и наследует общий стиль из базового шаблона.

#### registration.html



**{% extends "base.html" %}**

**{% block title %} {{ title }} {% endblock %}**

**{% block content %}**

<div class=**"row"**>

<div class=**"container mt-5"**>

<div class=**"row justify-content-center"**>

<div class=**"col-md-6 bg-white p-4 mb-4 mx-3 rounded custom-shadow"**>

<h2 class=**"text-center mb-4"**>**{{ title }}**</h2>

<form action=**"{% url 'user\_app:registration' %}"** method=**"post"**>

**{% csrf\_token %}**

<div class=**"row"**>

**{% for field in form %}**

**{{ field.label }}**

**{{ field }}**

**{% if field.errors %}**

<div class=**"alert alert-warning"** role=**"alert"**>

**{{ field.errors }}**

</div>

**{% endif %}**

**{% endfor %}**

</div>

<br>

<button type=**"submit"** class=**"btn btn-success btn-block"**>**Зарегистрироваться**</button>

</form>

</div>

</div>

</div>

</div>

**{% endblock %}**

Этот шаблон отвечает за отображение формы регистрации пользователя на веб-странице.

Он также наследуется от базового шаблона base.html с применением стилей и передачей заголовка title.

Форма отправляется по URL, который соответствует представлению registration в приложении user\_app. Метод отправки — POST. Включает защиту от подделки межсайтовых запросов.

Далее отображаются поля формы. Каждое поле формы отображается с соответствующей меткой. Если есть ошибки валидации, они будут выведены в блоке предупреждения.

Данный шаблон предоставляет пользователю удобный интерфейс для регистрации, обрабатывает ввод данных и проверяет наличие ошибок, чтобы обеспечить корректное заполнение формы для сохранения данных в БД.