МИНИСТЕРСТВО ОБРАЗОВАНИЯ МОСКОВСКОЙ ОБЛАСТИ

Государственное бюджетное профессиональное образовательное учреждение Московской области «Ногинский колледж»

Курсовой проект

по МДК.09.01 Проектирование и разработка веб-приложений

ПМ.09 Проектирование, разработка и оптимизация веб-приложений

Тема:

ИСПОЛЬЗОВАНИЕ ФРЕЙМВОРКА DJANGO В РАЗРАБОТКЕ ВЕБ-ПРИЛОЖЕНИЯ «ИНТЕРНЕТ-МАГАЗИН ДОМСТРОЙ»

Разработчик: студент группы 3ИСПР2

Макарычев Г.А.

(подпись)

Оценка защиты курсового проекта

« »

Дата защиты

« » 2024

Руководитель проекта:

преподаватель Степанов С.О.

(подпись)

Ногинск, 2024 г

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc169652449)

[ВВЕДЕНИЕ 3](#_Toc169652450)

[ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 4](#_Toc169652451)

[1.1.Определения программ, фреймворков, редакторов и др. 4](#_Toc169652452)

[1.2.Расширения необходимые в Visual Studio Code 5](#_Toc169652453)

[ОСНОВНАЯ ЧАСТЬ КУРСОВОЙ РАБОТЫ 6](#_Toc169652454)

[1.Описание рабочего процесса с GitHub Desktop 6](#_Toc169652455)

[2. работа с фреймворком Django 6](#_Toc169652456)

[2.1. Model-View-Template (MVT) 6](#_Toc169652457)

[2.2. Установка и активация Django 7](#_Toc169652458)

[2.3. Создание первого проекта Django 7](#_Toc169652459)

[2.4. Работа с Django приложениями 9](#_Toc169652460)

[2.4.1. Приложение main2 11](#_Toc169652461)

[2.4.2. Приложение goods 12](#_Toc169652462)

[2.4.3. Приложение users 15](#_Toc169652463)

[2.4.3. Приложение carts 17](#_Toc169652464)

[2.4.5. Приложение orders 19](#_Toc169652465)

[2.5. Установка PostgresSql 22](#_Toc169652466)

[ЗАКЛЮЧЕНИЕ 24](#_Toc169652467)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 25](#_Toc169652468)

[ПРИЛОЖЕНИЕ 1 26](#_Toc169652469)

[ПРИЛОЖЕНИЕ 2 34](#_Toc169652470)

# ВВЕДЕНИЕ

Многие современные компании занимаются продажей товаров и услуг через интернет-магазины, так как это дает неплохой прирост покупателей за счет ряда факторов:

* В современном мире интернетом пользуется большинство людей, многие могут стать потенциальными покупателями.
* В отличие от торговых точек, привязанных к конкретному месту, доступ к сайту открыт в любое время суток.
* Уменьшение затрат на производственные издержки. Затраты на продажу товаров в Интернет-магазине ниже, чем в торговой точке.

В ходе курсового проекта я подробно распишу создание своего веб-приложения по продаже мебели «интернет-магазин ДомСтрой». Приложение будет написано в редакторе кода Visual Studio Code на языке программирования python с помощью популярного фреймворка Django. В рамках данной работы были поставлены следующие задачи:

1. Узнать, как в Visual Studio Code с помощью фреймворка Django создать интернет-магазин «ДомСтрой»;
2. Как и какие компоненты (расширения) понадобится установить для работы с фреймворком в редакторе;
3. Использование GitHub Desktop для работы над проектом, контроля версий и управления изменениями.
4. Поиск подходящего для нашего приложения шаблона из сторонних источников, его реорганизация и внедрение в него основной части редактора (Backend-а);
5. Создание и настройка приложений Django и его компонентов;
6. Скачивание PostgresSql и подключение БД к редактору и его приложениям.
7. Создание панели администратора для получения возможности добавления, изменения и удаления товаров;
8. Создание возможности авторизации и регистрации пользователя;
9. Создание корзины и возможность добавления товара в корзину;
10. Создание личного кабинета со списком заказов, корзиной товаров и информации о пользователе;
11. Создание каталога товаров с различными категориями и возможностью сортировки товаров по акции, от дешевых к дорогим и наоборот, а также реализовать возможность просмотра характеристик товаров;
12. Создание системы поиска товаров по их названию или описанию;

По возможности также будет реализована загрузка готового сайта на хостинг.

# ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

1.1.Определения программ, фреймворков, редакторов и др.

GitHub Desktop (GitD) - это графический пользовательский интерфейс (GUI) для управления репозиториями Git и взаимодействия с ними. Он предоставляет разработчикам простой в использовании способ совместной работы, отслеживания изменений и управления ветвями в своих проектах. С помощью GitHub Desktop пользователи могут легко клонировать репозитории, создавать ветки, фиксировать изменения и отправлять свою работу на GitHub. Настройки для этого редактора будут находится в файле settings.json.

<https://beget.com/ru> - международный хостинг-провайдер, аккредитованный национальный и международный регистратор доменных имен[1]. Предоставляет услуги виртуального хостинга, аренды виртуальных серверов, аренды выделенных серверов и регистрации доменных имен. Бегет является аккредитованным регистратором в русскоязычных доменных зонах .РФ, .RU и .SU, национальных доменных зонах .WS, .TV, .ME, .KZ, .TW и др.

Visual Studio Code (vsсode) - это мощный и популярный редактор кода, разработанный компанией Microsoft. Он предназначен для работы с различными языками программирования и обеспечивает широкий спектр функций, облегчающих разработку и редактирование кода.

Django - это фреймворк для создания веб-приложений с помощью языка программирования Python. На Django можно создавать широкий диапазон веб-приложений: от небольших персональных сайтов до высоконагруженных сложных веб-сервисов. Пумолчанию он предлагает готовую функциональность для ряда распространенных задач, например, систему аутентификации, генерацию карт сайта и т.д., благодаря чему нам достаточно взять уже готовые компоненты сайта и добавить часть фреймворка. В Django большое внимание уделяется безопасности, благодаря чему фреймворк помогает разработчикам избежать многих распространенных проблем в системе безопасности, например, sql-инъекций.

Model-View-Template(MVT) - это набор из трех важных компонентов: Model, View и Template (модель, шаблон и представление). Модель помогает обрабатывать базу данных. Это уровень доступа к данным, который обрабатывает данные.

PostgreSQL - это мощная объектно-реляционная система баз данных с открытым исходным кодом, активная разработка которой ведется более 35 лет и заслужила прочную репутацию благодаря надежности, функциональной устойчивости и производительности в ней мы будем хранить продукты, пользователей, карзины и др. База Данных – БД(бд).

1.2.Расширения необходимые в Visual Studio Code

Python- Расширение с широкой поддержкой языка Python (для всех активно поддерживаемых версий Python), предоставляющее точки доступа для расширений для простой интеграции и поддержку IntelliSense (Pylance), отладки (Python Debugger), форматирования, проверки и навигации по коду, рефакторинг, обозреватель переменных, обозреватель тестов и многое другое!

Pylance — это расширение, которое работает вместе с Python в vsсode и обеспечивает эффективную языковую поддержку. Под капотом Pylance работает Pyright, инструмент проверки статических типов Microsoft. Используя Pyright, Pylance имеет возможность расширить возможности Python IntelliSense за счет расширенной информации о типах, помогая вам быстрее писать более качественный код.

Auto Close Tag - Автоматически добавляйте закрывающий тег HTML/XML.

Auto Rename Tag - Автоматически переименовывайте парные теги HTML/XML.

Black Formatter - обеспечивает поддержку форматирования ваших файлов Python.

Code Runner – нужен для выделение и запуска в консоли части кода.

Django, Django Template Support - подсветка синтаксиса html, css, JavaScript, формат и красивый код, автодополнение для html-тегов, атрибутов, css, JavaScript, некоторые советы по ошибкам/проблемам при форматировании.

IntelliCode –предоставляет функции разработки с помощью искусственного интеллекта для разработчиков Python, JavaScript и Java в vsсode, а также аналитические сведения, основанные на понимании контекста вашего кода в сочетании с машинным обучением. Может прогнозировать целую строку кода на основе вашего текущего контекста для Python, JavaScript.

API IntelliCode — это функция, которая позволяет вам увидеть реальные примеры того, как другие разработчики использовали данную функцию. Показанные примеры взяты из общедоступных репозиториев с открытым исходным кодом на GitHub.

# ОСНОВНАЯ ЧАСТЬ КУРСОВОЙ РАБОТЫ

1.Описание рабочего процесса с GitHub Desktop

Рабочий процесс с GitD начинается следующим образом:

1. Необходимо установить GitD на свой компьютер, а также помимо расширение описанных выше для VS Code нужно установить расширение GitHub Classroom для взаимодействия с GitHub.

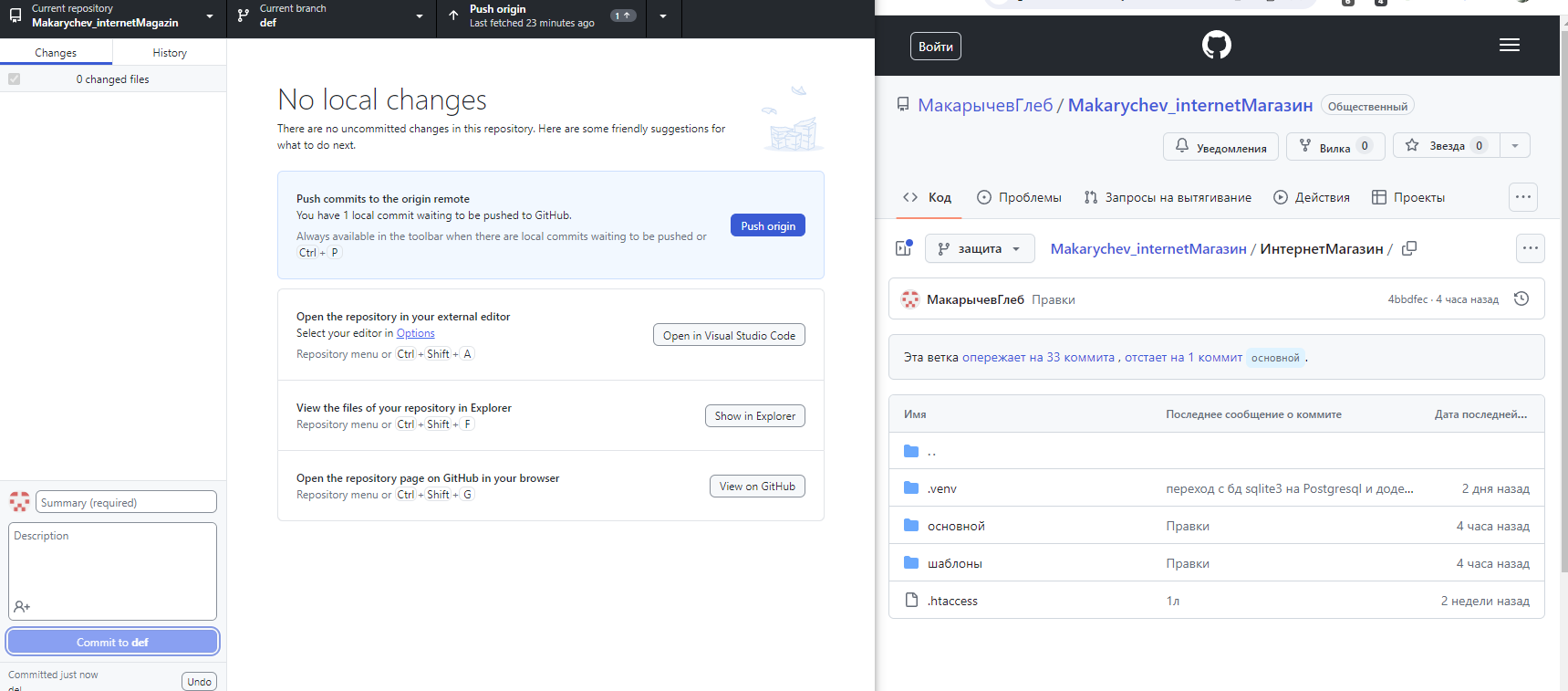


Рис.1.1- GitD

2. Нужно создать или войти в свой аккаунт на GitHub.

3. Открываем или создаем репозиторий:

4. Добавляем или изменяем файлы своего проекта.

5. Чтобы зафиксировать изменения добавляем текст в описание, выбираем файлы, которые нужно зафиксировать, и кнопку нажимаем “commit”.

6. После коммита необходимо отправить изменения на удаленный репозиторий находящийся на аккаунте в GitHub. Для этого нажимаю кнопку "Push origin" и подтверждаю отправку изменений.

7. При необходимости синхронизирую локальные изменения с удаленным репозиторием, нажав кнопку "Fetch origin" или "Sync" в GitHub Desktop.

Таким образом, GitHub Desktop облегчает работу с репозиториями и позволяет быстро и удобно вести разработку на GitHub.

2. работа с фреймворком Django

2.1. Model-View-Template (MVT)

Фреймворк Django реализует архитектурный паттерн Model-View-Template который является модификацией распространенного в веб-программировании паттерна MVC (Model-View-Controller). Основные элементы паттерна:

* URL dispatcher: при получение запроса на основании запрошенного адреса URL определяет, какой ресурс должен обрабатывать данный запрос.
* View: получает запрос, обрабатывает его и отправляет в ответ пользователю некоторый ответ. Если для обработки запроса необходимо обращение к модели и базе данных, то View взаимодействует с ними. Для создания ответа может применять Template или шаблоны. В архитектуре MVC этому компоненту соответствуют контроллеры (но не представления).
* Model: описывает данные, используемые в приложении. Отдельные классы, как правило, соответствуют таблицам в базе данных.
* Template: представляет логику представления в виде сгенерированной разметки html. В MVC этому компоненту соответствует View, то есть представления.

Когда к приложению приходит запрос, то URL dispatcher определяет, с каким ресурсом сопоставляется данный запрос и передает этот запрос выбранному ресурсу. Ресурс фактически представляет функцию или View, который получает запрос и определенным образом обрабатывает его. В процессе обработки View может обращаться к моделям и базе данных, получать из нее данные, или, наоборот, сохранять в нее данные. Результат обработки запроса отправляется обратно, и этот результат пользователь видит в своем браузере. Как правило, результат обработки запроса представляет сгенерированный html-код, для генерации которого применяются шаблоны (Template).

2.2. Установка и активация Django

Создаем папку «InternetMagazin» и открываем в редакторе vscode. Открываем терминал и создаем изолированную виртуальную среду: python |-m venv .venv

Для использования виртуальную среду надо активировать. В терминале vscode (в PowerSell) вводим команды по порядку:

1| Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser

2| .venv\Scripts\Activate.ps1

После активации виртуальной среды для установки Django выполним в консоли следующую команду: python |-m pip install Django

Она устанавливает последнюю версию Django.

После окончания работы с виртуальной средой мы можем ее деактивировать с помощью команды: |deactivate

2.3. Создание первого проекта Django

При установке Django в папке виртуальной среды устанавливается утилита «django-admin». А на Windows также исполняемый файл «django-admin.exe.» Их можно найти в папке виртуальной среды, в которую производилась установка Django. «django-admin» предоставляет ряд команд для управления проектом Django. В частности, для создания проекта применяется команда «startproject»:

|django-admin startproject main

Этой команде в качестве аргумента передается название проекта.

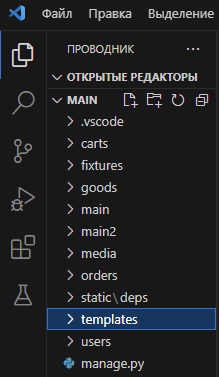


Рис.2.3.1-main каталог

Рассмотрим вкратце его структуру:

папка migrations: предназначена для хранения миграций - скриптов, которые позволяют синхронизировать структуру базы данных с определением моделей

* \_\_init\_\_.py: указывает интерпретатору python, что текущий каталог будет рассматриваться в качестве пакета;
* admin.py: предназначен для административных функций, в частности, здесь производится регистрация моделей, которые используются в интерфейсе администратора;
* apps.py: определяет конфигурацию приложения;
* models.py: хранит определение моделей, которые описывают используемые в приложении данные;
* tests.py: хранит тесты приложения;
* views.py: определяет функции, которые получают запросы пользователей, обрабатывают их и возвращают ответ;

Чтобы виртуальное окружение работало постоянно перезаходим из vscode в проект «main» выбираем manage.py и в правом нижнем углу меняем системный интерпретатор на интерпретатор виртуальной среды: venv\python.exe

Для запуска сервера набираем команду |python manage.py runserverhev

У нас появляется ссылка на сервер, на котором мы будем строить сервер:

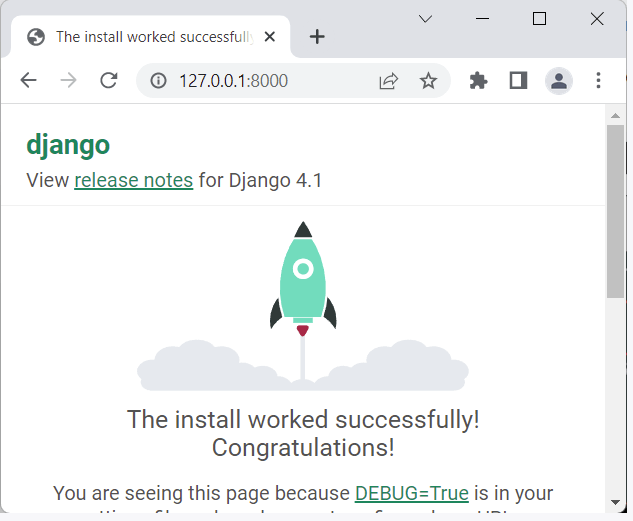


Рис.2.3.2-startserver

Также у нас появляедся db.sqlite3 которое в будущем мы заменим на Postresql т.к. функция поиска товара неработает в db.sqlite3.

2.4. Работа с Django приложениями

Создаем в корне main папку .vscode и в нее добавляем файл settings.json с настройками vscode (для корректной работы с расширениями). Создаем папки templates где будут хранится наши переделанные страницы шаблона:

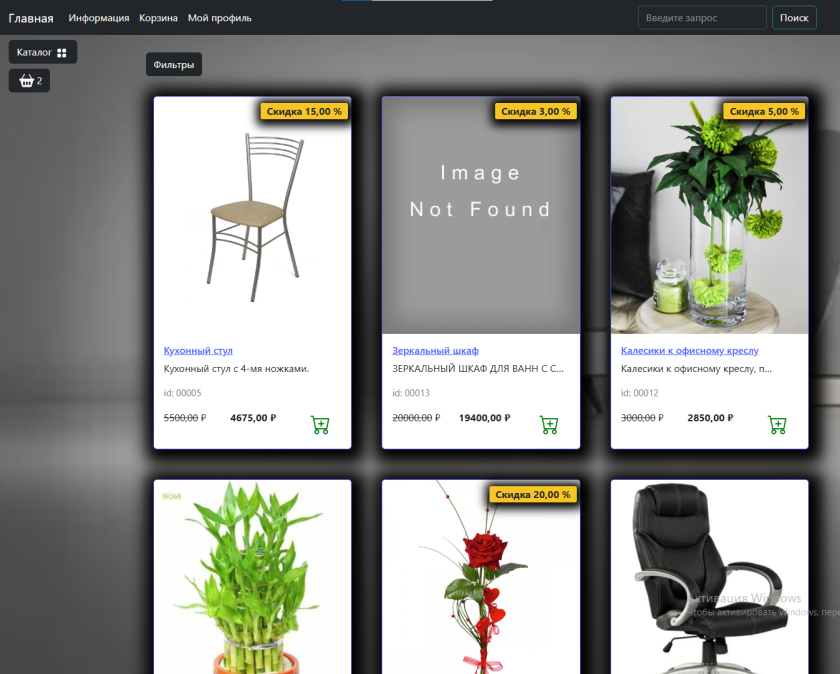


Рис.2.4.1-base.html

и static в которм находятся все елементы связанные с bootstrap, css и js.

В первом приложении «main» есть файл settings.py - Этот файл играет основную роль в настройке и функционировании веб-приложений на основе Django.

from pathlib import Path

from django.http import HttpResponseNotFound

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

SECRET\_KEY = 'django-insecure-gw\_-v+upl=+=o0\_3uuergtca$)\*#d+4s%qgy%^c9d\*4!vqn^r%'

DEBUG = True

ALLOWED\_HOSTS = ['\*']

INSTALLED\_APPS = ['django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'django.contrib.postgres',

    'main2', 'goods', 'users', 'carts', 'orders']

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',]

ROOT\_URLCONF = 'main.urls'

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 = 'main.wsgi.application'

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.postgresql',

        'NAME': 'home',

        'USER': 'home',

        'PASSWORD': 'home',

        'HOST': 'localhost',

        'PORT': '5432', }}

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', },

]

LANGUAGE\_CODE = 'ru'

TIME\_ZONE = 'UTC'

USE\_I18N = True

USE\_TZ = True

STATIC\_URL = 'static/'

STATICFILES\_DIRS = [

    BASE\_DIR / 'static'

    ]

MEDIA\_URL = 'media/'

MEDIA\_ROOT = BASE\_DIR / 'media'

DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'

AUTH\_USER\_MODEL = 'users.User'

LOGIN\_URL = '/user/login/'

2.4.2- main/Settings.py

Некоторые ключевые моменты работы данного кода:

* Определение структуры директории проекта и ее использование для задания путей к шаблонам и статическим файлам.
* Установленные приложения, например, которые будут использоваться в проекте, приложения для работы с товарами, пользователями, корзинами и заказами.
* Настройки базы данных PostgreSQL для проекта.
* Настройки безопасности, такие как проверка паролей пользователей.
* Настройки международной локации и времени.
* Настройки для обработки статических файлов и медиафайлов.
* Определение модели пользователя для авторизации.
* Настройка URL-адреса для страницы входа пользователя(main2).

В main/urls.py записана конфигурация URL-адресов помогающая направлять различные URL-адреса ко всем приложениям, а также обслуживать мультимедийные файлы в режиме админ-панели:

from django.contrib import admin

from django.urls import path, include

from django.conf.urls.static import static

from main import settings

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', include('main2.urls', namespace='main2')),

    path('catalog/', include('goods.urls', namespace='catalog')),

    path('user/', include('users.urls', namespace='user')),

    path('cart/', include('carts.urls', namespace='cart')),

    path('orders/', include('orders.urls', namespace='orders')),

]

if settings.DEBUG:

    urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

2.4.3- main/urls.py

urls.py работает как маршрутизатор вызывающий функции или классы описанные во views-е, views запрашивает данные из БД и она возвращается к vies далее информация полученная из БД заполняет пустой шаблон информацией и после формирования шаблона функция возвращает ответ пользователю. Структура моего веб-приложения будет состоять из 5 основных приложений

2.4.1. Приложение main2

Приложение main2 необходимо для регистрации таблиц, которые будут создаваться в админ-панели:

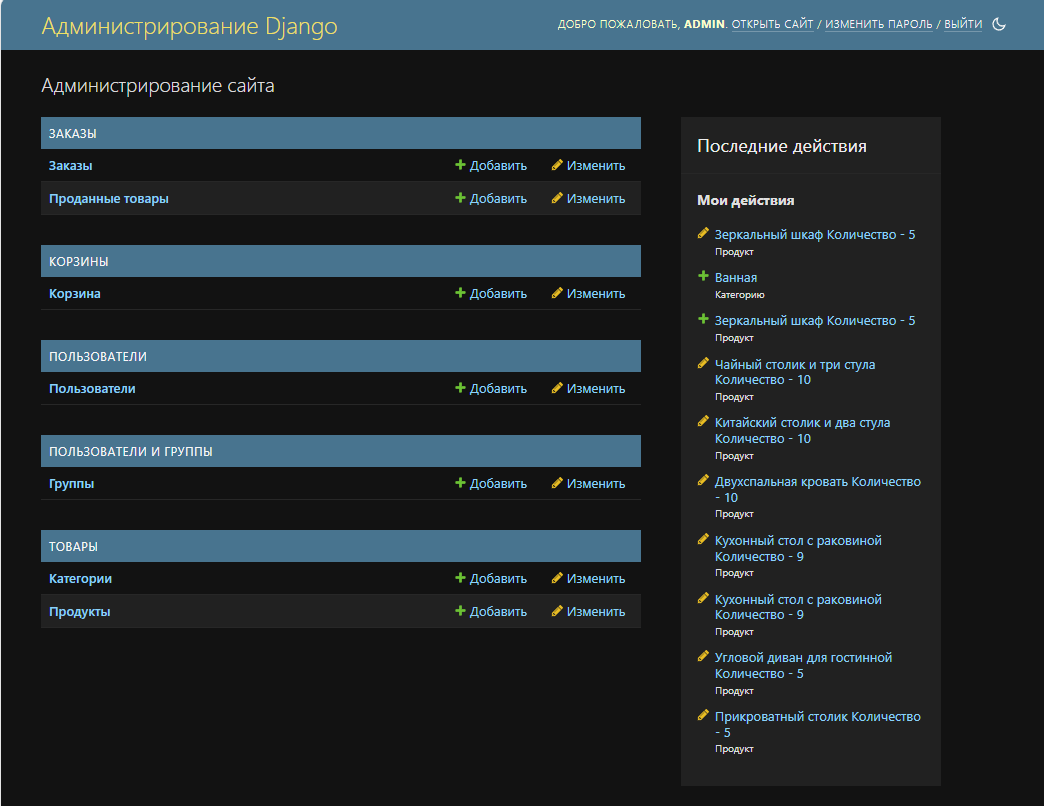


Рис.2.4.1.1-admin

без подключения базы данных мы не сможем зайти в админ-панель, поэтому до момента создания функции поиска будем использовать встроенную БД db.sqlite3. Функция данного приложения будет вести на главную страницу сайта, а также на страницу с описанием сайта.

From jango.shortcuts import render

def index(request):

    context = {

        'title': 'ДомСтрой – Главная',

        'content': 'Интернет-магазин мебели «ДомСтрой»',

    }

    return render(request, ‘main2/index.html’, context)

def about(request):

    context = {

        ‘title’: ‘ДомСтрой – О нас’,

        'content': 'О нас',

        'text\_on\_page': ' ДомСтрой – это поистине мебельный рай. '

    }

    return render(request, 'main2/about.html', context)

2.4.1.2-main2/views.py

2.4.2. Приложение goods

Приложение goods предназначено для работы с товарами и их категориями. Таблицы категорий будут состоять из имени и slug(id), а таблицы продуктов будут состоять из имени, slug, описания, картинки, цены, скидки, кол-ва и по внешнему ключу они будут связаны с таблицей категорий, их структуры показана в goods/models.py :

class Categories(models.Model):

    name = models.CharField(max\_length=100, unique=True, verbose\_name='Название')

    slug = models.SlugField(max\_length=200, unique=True, blank=True, null=True, verbose\_name='URL')

class Products(models.Model):

    name = models.CharField(max\_length=100, unique=True, verbose\_name='Название')

    slug = models.SlugField(max\_length=200, unique=True, blank=True, null=True, verbose\_name='URL')

    description = models.TextField(blank=True, null=True, verbose\_name='Описание')

    image = models.ImageField(upload\_to='goods\_images', blank=True, null=True, verbose\_name='Изображение')

    price = models.DecimalField(default=0.00, max\_digits=8, decimal\_places=2, verbose\_name='Цена')

    discount = models.DecimalField(default=0.00, max\_digits=4, decimal\_places=2, verbose\_name='Скидка(%)')

    quantity = models.PositiveIntegerField(default=0, verbose\_name='Кол-во')

    category = models.ForeignKey(to=Categories, on\_delete=models.CASCADE, verbose\_name='Категория')

2.4.2.1-goods/models.py

Также к таблице товаров будут подсоединяться таблицы корзин и заказов из приложений carts и orders соответственно. Также основными функциями приложения goods будет отображение определенного количества товара на странице:

from django.core.paginator import Paginator

paginator = Paginator(goods, 6)

    current\_page = paginator.page(int(page))

    context = {

        'title': 'ДомСтрой - Каталог',

        'goods': current\_page,

        "slug\_url": category\_slug}

 return render(request, 'goods/catalog.html', context)

2.4.2.2-goods/views.py

возможность перелистывания каталога на странице:

from django.shortcuts import get\_list\_or\_404, render

from .models import Products

from .utils import q\_search

def catalog(request, category\_slug=None):

    page = request.GET.get('page', 1)

    on\_sale = request.GET.get('on\_sale', None)

    order\_by = request.GET.get('order\_by', None)

    query = request.GET.get('q', None)

    if category\_slug == 'all':

        goods = Products.objects.all()

    elif query:

        goods = q\_search(query)

    else:

        goods = get\_list\_or\_404(

            Products.objects.filter(category\_\_slug=category\_slug)

            )

    if on\_sale:

        goods = goods.filter(discount\_\_gt=0)

    if order\_by and order\_by != "default":

        goods = goods.order\_by(order\_by)

2.4.2.3-goods/views.py

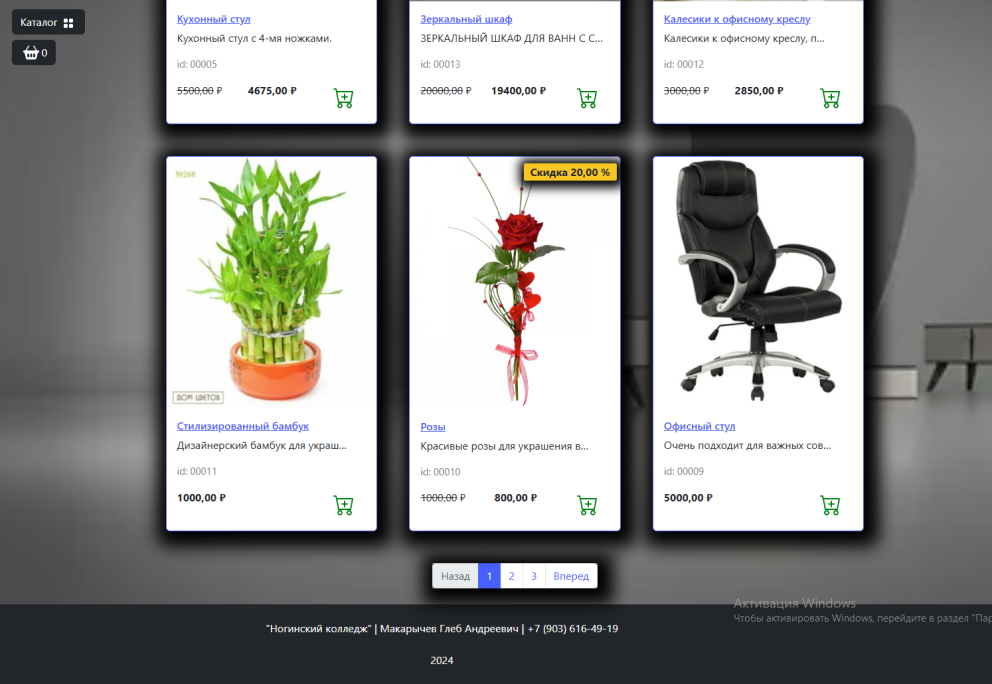


Рис.2.4.2.4-page

Также в приложении goods создана утилита позволяющая пользователю при написании слов в строку поиска в случае если слово совпадает со словом из названия или описания товара в каталоге появляются эти товары, а те слова подсвечиваются желтым:

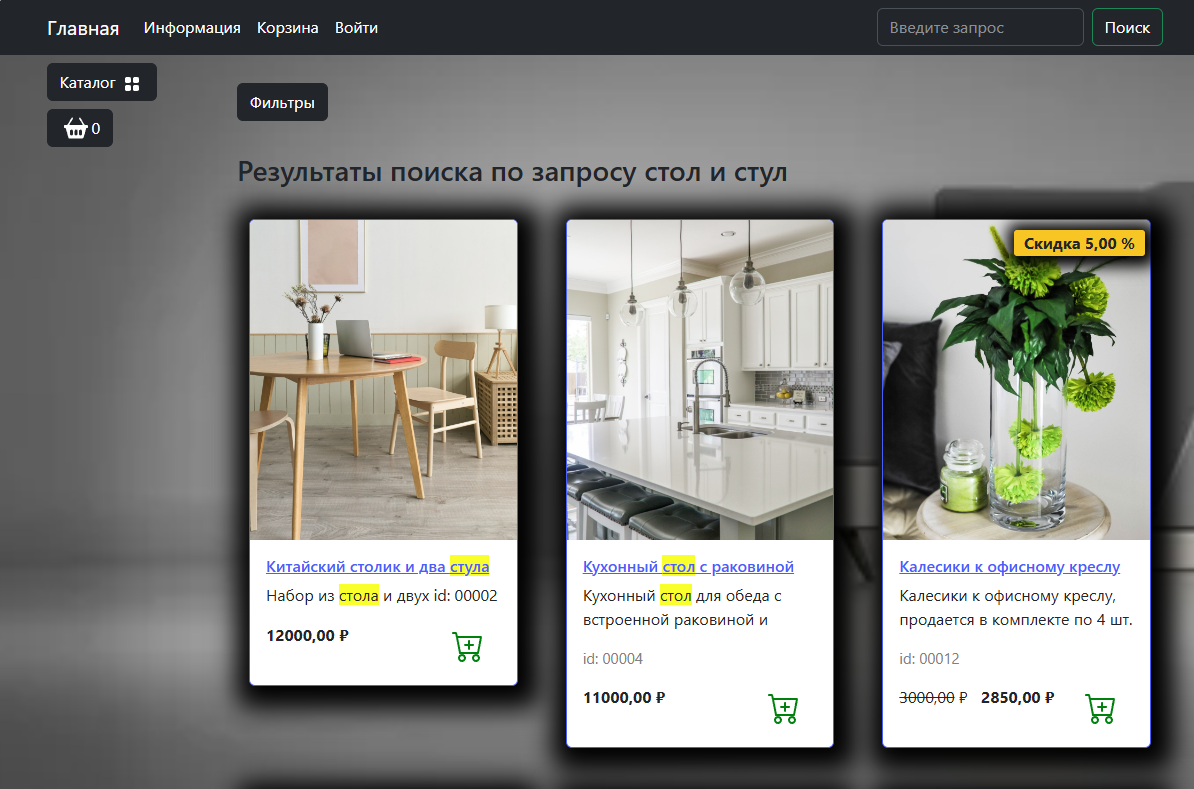


Рис.2.4.2.5- Search-yellow

В случае если товар не совпадает с вбитым запросом в строку поиска то по данному запросу ничего найдено не будет:

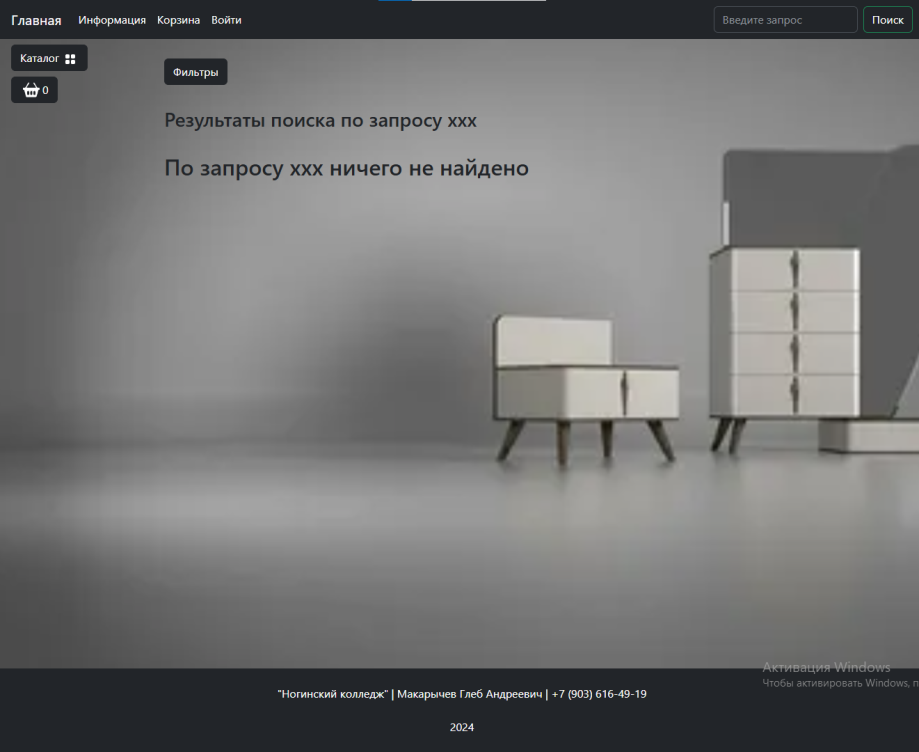


Рис.2.4.2.6- Search-none

В панели администратора в категории товаров был выстроен особый порядок заполнения и отображения информации о товарах:

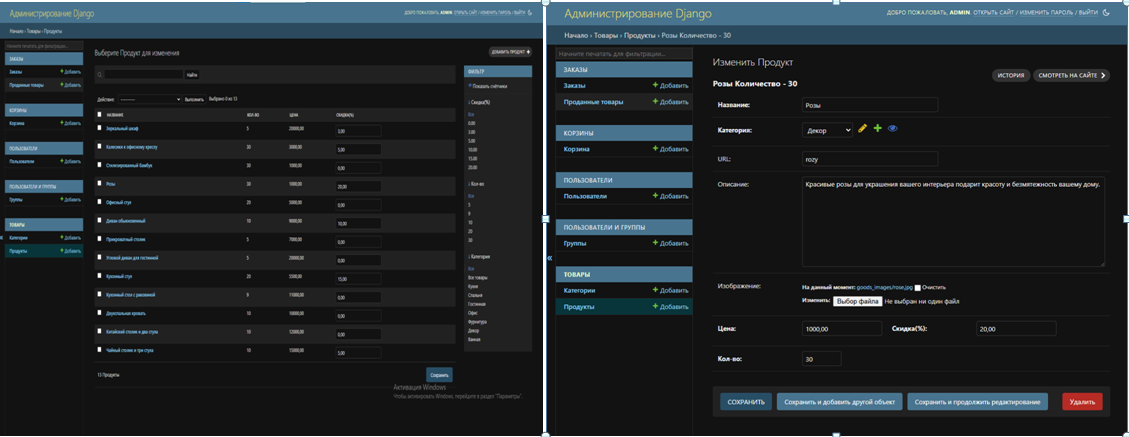


Рис.2.4.2.6-admin/goods

2.4.3. Приложение users

Приложение users предназначена для регистрации новых и авторизации старых пользователей:

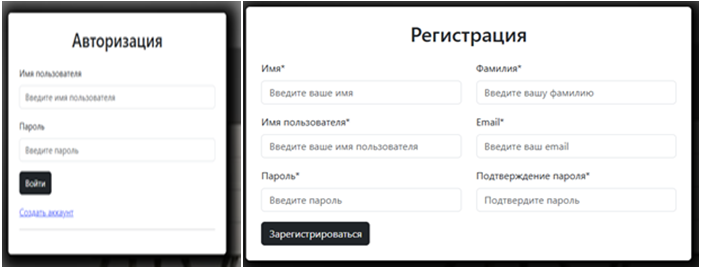


Рис.2.4.3.1-login/word

В формах данного приложения есть возможность изменения имени, маила и пароля в профиле пользователя Модель users состоит из аватарки и номера телефона пользователя который находится в его профиле:

from django.db import models

from django.contrib.auth.models import AbstractUser

class User (AbstractUser):

    image = models.ImageField(upload\_to='users\_images', blank=True, null=True, verbose\_name='Аватар')

    phone\_number = models.CharField(max\_length=10, blank=True, null=True)

    class Meta:

        db\_table = "user"

        verbose\_name = "Пользователя"

        verbose\_name\_plural = "Пользователи"

    def \_\_str\_\_(self):

        return self.username

2.4.3.2-users/models.py

Также профиль пользователей связан с таблицей корзины и таблицей заказов:

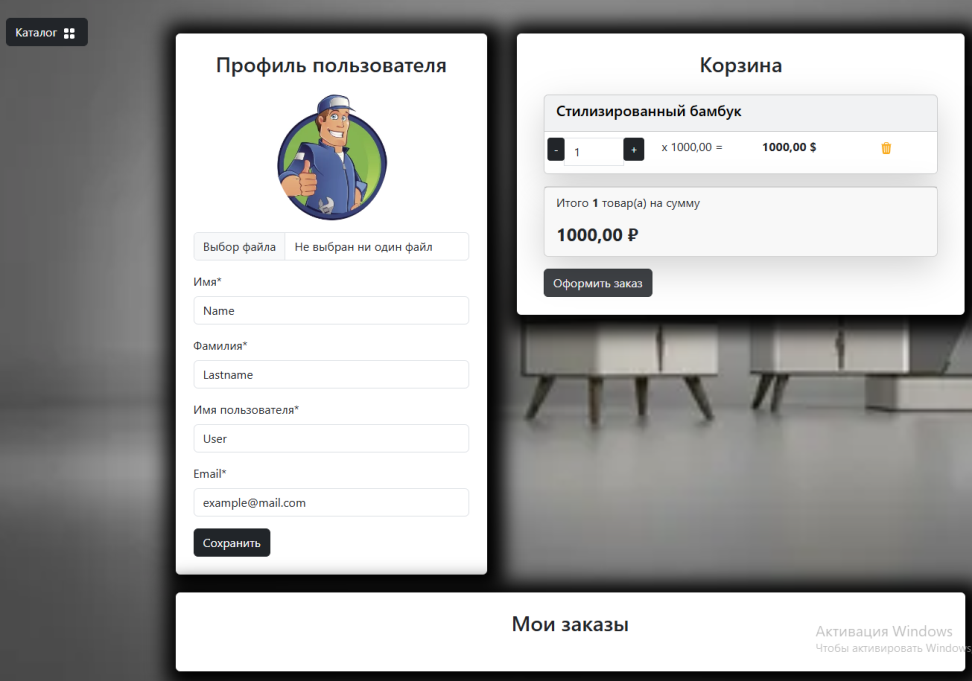


Рис.2.4.3.2-profile

В админ-панели категории users помимо возможности изменить данные пользователя, также возможность наделить правами администратора обычного пользователя:

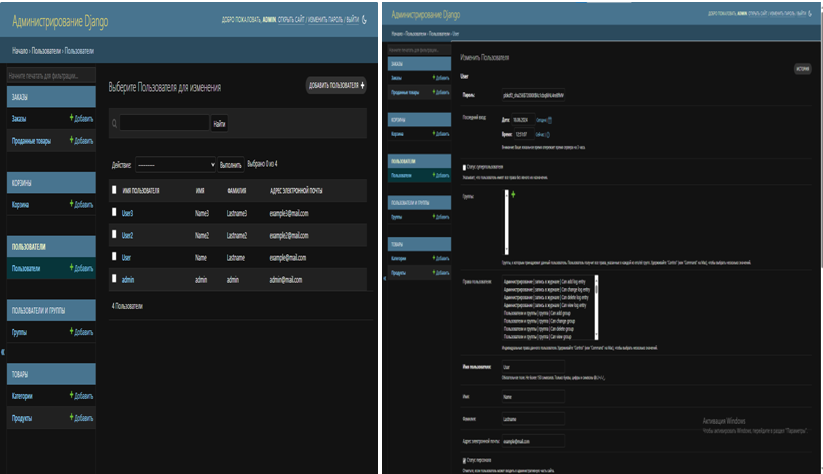


Рис.2.4.3.3-admin/users

2.4.3. Приложение carts

Основной функционал приложения carts завязан на добавлении товара в таблицу, возможность из каталога товаров или со страницы определенного товара добавить его в корзину:

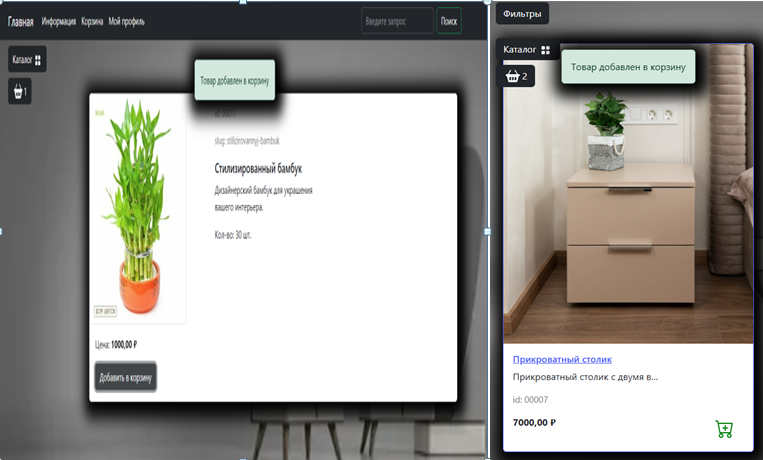


Рис.2.4.4.1-carts/product

На странице корзины есть возможность, увеличения и уменьшения кол-ва товара, его конечная цена возрастает или падает в зависимости от товара (его количества):

from django.db import models

from goods.models import Products

from users.models import User

class CartQueryset(models.QuerySet):

    def total\_price(self):

        return sum(cart.products\_price() for cart in self)

    def total\_quantity(self):

        if self:

            return sum(cart.quantity for cart in self)

        return 0

class Cart(models.Model):

    user = models.ForeignKey(User, on\_delete=models.CASCADE, blank=True, null=True, verbose\_name='Пользователь')

    product = models.ForeignKey(Products, on\_delete=models.CASCADE, verbose\_name='Товар')

    quantity = models.IntegerField(default=0, verbose\_name='Количество')

    session\_key = models.CharField(max\_length=32, blank=True, null=True)

    created\_timestamp = models.DateTimeField(auto\_now\_add=True, verbose\_name='Дата добавления')

    class Meta:

        db\_table = "cart"

        verbose\_name = "Корзина"

        verbose\_name\_plural = "Корзина"

        ordering = ("id",)

    objects = CartQueryset().as\_manager()

    def products\_price(self):

        return round(self.product.sell\_price() \* self.quantity, 2)

    def \_\_str\_\_(self):

        if self.user:

            return f'Корзина {self.user.username} | Товар {self.product.name} | Количество {self.quantity}'

        return f'Анонимная корзина | Товар {self.product.name} | Количество {self.quantity}'

2.4.4.2-carts/models.py

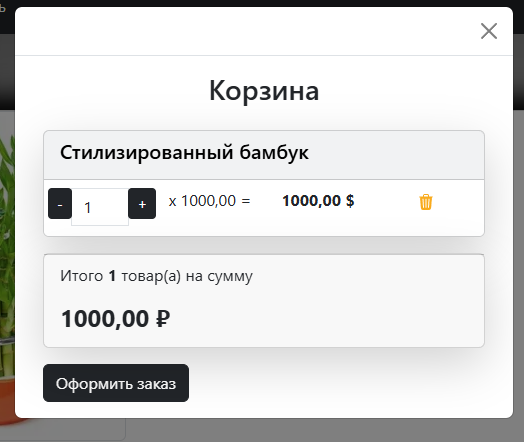


Рис.2.4.4.3-carts/product

В приложении carts модельная таблица состоит из имя пользователя и продукта, которые являются внешними ключами, а также из ключа сессий, кол-ва и даты добавления товара:

from django.db import models

from goods.models import Products

from users.models import User

class CartQueryset(models.QuerySet):

    def total\_price(self):

        return sum(cart.products\_price() for cart in self)

    def total\_quantity(self):

        if self:

            return sum(cart.quantity for cart in self)

        return 0

class Cart(models.Model):

    user = models.ForeignKey(User, on\_delete=models.CASCADE, blank=True, null=True, verbose\_name='Пользователь')

    product = models.ForeignKey(Products, on\_delete=models.CASCADE, verbose\_name='Товар')

    quantity = models.IntegerField(default=0, verbose\_name='Количество')

    session\_key = models.CharField(max\_length=32, blank=True, null=True)

    created\_timestamp = models.DateTimeField(auto\_now\_add=True, verbose\_name='Дата добавления')

    class Meta:

        db\_table = "cart"

        verbose\_name = "Корзина"

        verbose\_name\_plural = "Корзина"

        ordering = ("id",)

    objects = CartQueryset().as\_manager()

    def products\_price(self):

        return round(self.product.sell\_price() \* self.quantity, 2)

    def \_\_str\_\_(self):

        if self.user:

            return f'Корзина {self.user.username} | Товар {self.product.name} | Количество {self.quantity}'

        return f'Анонимная корзина | Товар {self.product.name} | Количество {self.quantity}'

2.4.4.4-carts/models.py

2.4.5. Приложение orders

Приложение orders состоит из двух таблиц. Первая таблица заказа это форма заполнения состоящая из зарегистрированного пользователя, даты создания заказа, номера телефона пользователя, требуется ли доставка, адреса доставки, оплата при получении, оплачен ли заказ и статуса заказа:

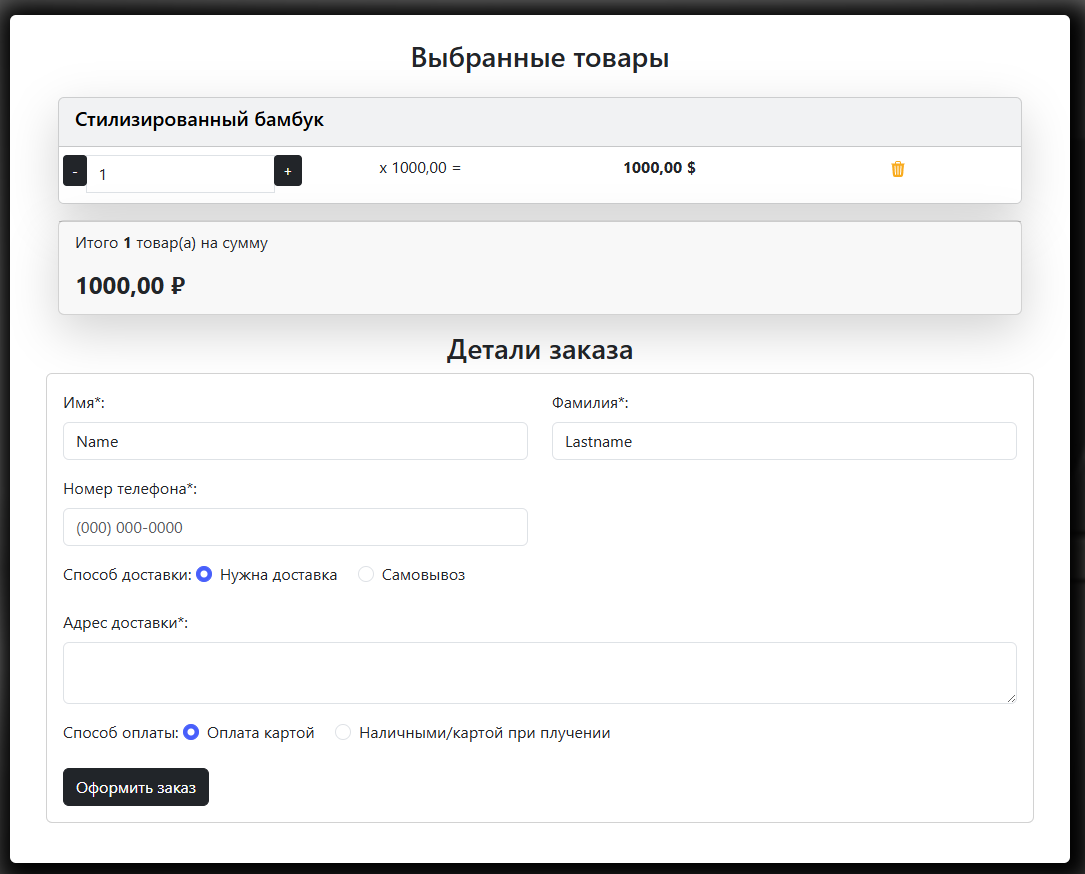


Рис.2.4.5.1-oder/forms

К функциям приложения orders можно отнести create\_order: Это основная функция, которая обрабатывает создание заказа. Декоратор @login\_required гарантирует, что только аутентифицированные пользователи могут получить доступ к функции create\_order Если HTTP-метод запроса — POST (что означает, что форма была отправлена), форма CreateOrderForm заполняется данными из запроса. Если форма валидна, начинается транзакция, чтобы гарантировать целостность данных. Проверяется наличие товаров в корзине пользователя. Если они есть, создается заказ и для каждого товара в корзине создается OrderItem. Если количество заказанного товара превышает количество на складе, вызывается исключение ValidationError.

После успешного создания всех OrderItem корзина пользователя очищается, и пользователь перенаправляется на свой профиль с сообщением об успешном оформлении заказа.

Обработка исключений: Если возникает исключение ValidationError, пользователю показывается соответствующее сообщение, и он перенаправляется обратно на страницу создания заказа.

GET-запрос: Если метод запроса не POST, то есть страница запрашивается впервые, форма инициализируется данными пользователя, и пользователю отображается страница для создания заказа.

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

from django.contrib import messages

from django.db import transaction

from django.forms import ValidationError

from django.shortcuts import redirect, render

from carts.models import Cart

from orders.forms import CreateOrderForm

from orders.models import Order, OrderItem

@login\_required

def create\_order(request):

    if request.method == "POST":

        form = CreateOrderForm(data=request.POST)

        if form.is\_valid():

            try:

                with transaction.atomic():

                    user = request.user

                    cart\_items = Cart.objects.filter(user=user)

                    if cart\_items.exists():

                        order = Order.objects.create(

                            user=user,

                            phone\_number=form.cleaned\_data["phone\_number"],

                            requires\_delivery=form.cleaned\_data["requires\_delivery"],

                            delivery\_address=form.cleaned\_data["delivery\_address"],

                            payment\_on\_get=form.cleaned\_data["payment\_on\_get"],

                        )

                        for cart\_item in cart\_items:

                            product = cart\_item.product

                            name = cart\_item.product.name

                            price = cart\_item.product.sell\_price()

                            quantity = cart\_item.quantity

                            if product.quantity < quantity:

                                raise ValidationError(

                                    f"Недостаточное количество товара {name} на складе\

                                    В наличии - {product.quantity}"

                                )

                            OrderItem.objects.create(

                                order=order,

                                product=product,

                                name=name,

                                price=price,

                                quantity=quantity,

                            )

                            product.quantity -= quantity

                            product.save()

                        cart\_items.delete()

                        messages.success(request, "Заказ оформлен!")

                        return redirect("user:profile")

            except ValidationError as e:

                messages.success(request, str(e))

                return redirect("orders:create\_order")

    else:

        initial = {

            "first\_name": request.user.first\_name,

            "last\_name": request.user.last\_name,

        }

        form = CreateOrderForm(initial=initial)

    context = {

        "title": "Home - Оформление заказа",

        "form": form,

        "order": True,

    }

    return render(request, "orders/create\_order.html", context=context)

2.4.5.2-orders/views.py

В приложения orders форма предназначена для использования в интерфейсе, где пользователь может ввести свои данные для оформления заказа, и включает в себя валидацию данных для обеспечения их корректности перед созданием заказа в системе:

import re

from django import forms

class CreateOrderForm(forms.Form):

    first\_name = forms.CharField()

    last\_name = forms.CharField()

    phone\_number = forms.CharField()

    requires\_delivery = forms.ChoiceField(

        choices=[("0", False), ("1", True), ],

        )

    delivery\_address = forms.CharField(required=False)

    payment\_on\_get = forms.ChoiceField(

        choices=[ ("0", 'False'), ("1", 'True'), ],

        )

    def clean\_phone\_number(self):

        data = self.cleaned\_data['phone\_number']

        if not data.isdigit():

            raise forms.ValidationError("Номер телефона должен содержать только цифры")

        pattern = re.compile(r'^\d{10}$')

        if not pattern.match(data):

            raise forms.ValidationError("Неверный формат номера")

        return data

2.4.5.3-orders/forms.py

2.5. Установка PostgresSql

Установка PostgreSQL и его интеграция с проектом Django состоит из нескольких шагов:

Для начала качаем установщик с официального сайта Настройка PostgreSQL:

Создайте базу данных и пользователя в админке PostgreSQL:

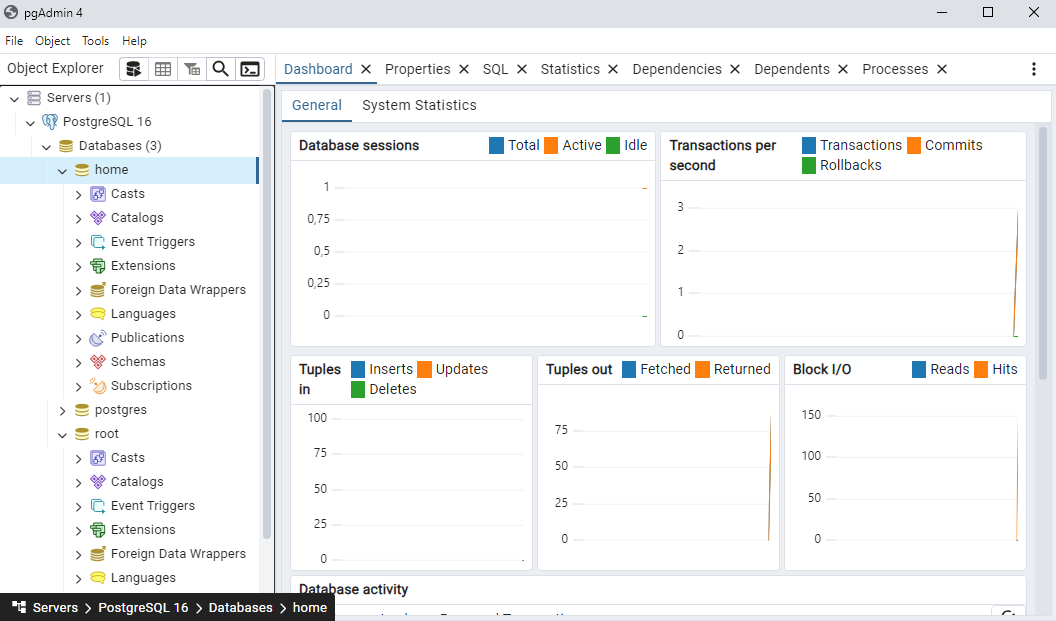


Рис.2.5.1-ДБ/ PostgreSQL

sudo -u postgres psql

CREATE DATABASE myprojectdb;

CREATE USER myuser WITH PASSWORD 'mypassword';

ALTER ROLE myuser SET client\_encoding TO 'utf8';

ALTER ROLE myuser SET default\_transaction\_isolation TO 'read committed';

ALTER ROLE myuser SET timezone TO 'UTC';

GRANT ALL PRIVILEGES ON DATABASE myprojectdb TO myuser;

\q

Вписываем в INSTALLED\_APPS =[ 'django.contrib.postgres'],

Устанавливаем пакет psycopg2, который является адаптером PostgreSQL для Python:

pip install psycopg2

В файле settings.py проекта Django меняем конфигурацию бд:

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.postgresql',

        'NAME': 'home',

        'USER': 'home',

        'PASSWORD': 'home',

        'HOST': 'localhost',

        'PORT': '5432',

    }

}

2.5.2-дб/url

Применим миграции для создания таблиц в новой базе данных:

python manage.py makemigrations

python manage.py migrate

Запускаем сервер разработки Django, чтобы убедиться, что все настроено правильно:

python manage.py runserver

После выполнения этих шагов проект Django будет использовать PostgreSQL в качестве системы управления бд. Убедитесь, что PostgreSQL запущен и работает, прежде чем пытаться подключиться к нему из Django.

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы с помощью фреймворка Django по созданию интернет-магазина «ДомСтрой» были выполнены следующие задачи:

1. Узнали, какие компоненты (расширения) больше всего подходят для работы с Django в vscode;
2. Была проведена работа с GitHub Desktop для управления версиями кода, создания коммитов, обеспечивая надежную разработку и контроль версий кода.
3. Нашли подходящий шаблон, реорганизовали и внедрили в него основную часть кода (Backend-а);
4. Скачали PostgresSql и подключили БД к редактору и его приложениям.
5. Создали панель администратора для получения возможности добавления, изменения и удаления товаров;
6. Сделали авторизации и регистрации пользователя;
7. Создание корзины и возможность добавления товара в корзину;
8. Сделали личный кабинет со списком заказов, корзиной товаров, информации о пользователе и возможность ее изменять;
9. Создали каталог товаров с различными категориями и возможностью сортировать товары по акции, от дешевых к дорогим и дорогих к дешевым или по умолчанию, а также реализована возможность просмотра характеристик товаров;
10. Создали корзины и возможность добавления товара в корзину как авторизированным, так и не авторизированным пользователям;
11. Создали систему поиска товаров при введении slug, id, названия или слова из описания товара;

В ходе работы над интернет-магазином было выявлено, что работа в Django ускоряет процесс разработки веб-приложений, позволяет эффективно управлять БД и выстраивать интуитивно-адаптивную систему, однако важно помнить об особенности проектирования БД, оптимизации производительности и защите от уязвимостей.

Хоть из-за несовместимости баз данных интернет-магазина и Beget-а приложение не удалось выложить на хостинг в будущем эта ошибка будет исправлена, а пока сайт будет находиться в открытом доступе на сайте Github.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Мой сайт: <https://github.com/MakarychevGleb/Makarychev_internetMagazin>

Шаблон: <https://github.com/Trash-Fresh/templates-shop-home>

Установщик PostgreSQL: <https://www.postgresql.org/download/>

Метанит активация и 1-ые шаги Django: <https://metanit.com/python/django/>

Django документация: <https://docs.djangoproject.com/en/5.0/>

Django документация: <https://django.fun/docs/django/5.0/>

Гайды по Django: <https://youtu.be/L-FyeHQwo4U?si=1dq2S__K0jtcOow8>

Сайт-хостинг: <https://beget.com/ru>

Источник(MVT): <https://pythonpip.ru/django/django-mvt>

# ПРИЛОЖЕНИЕ 1

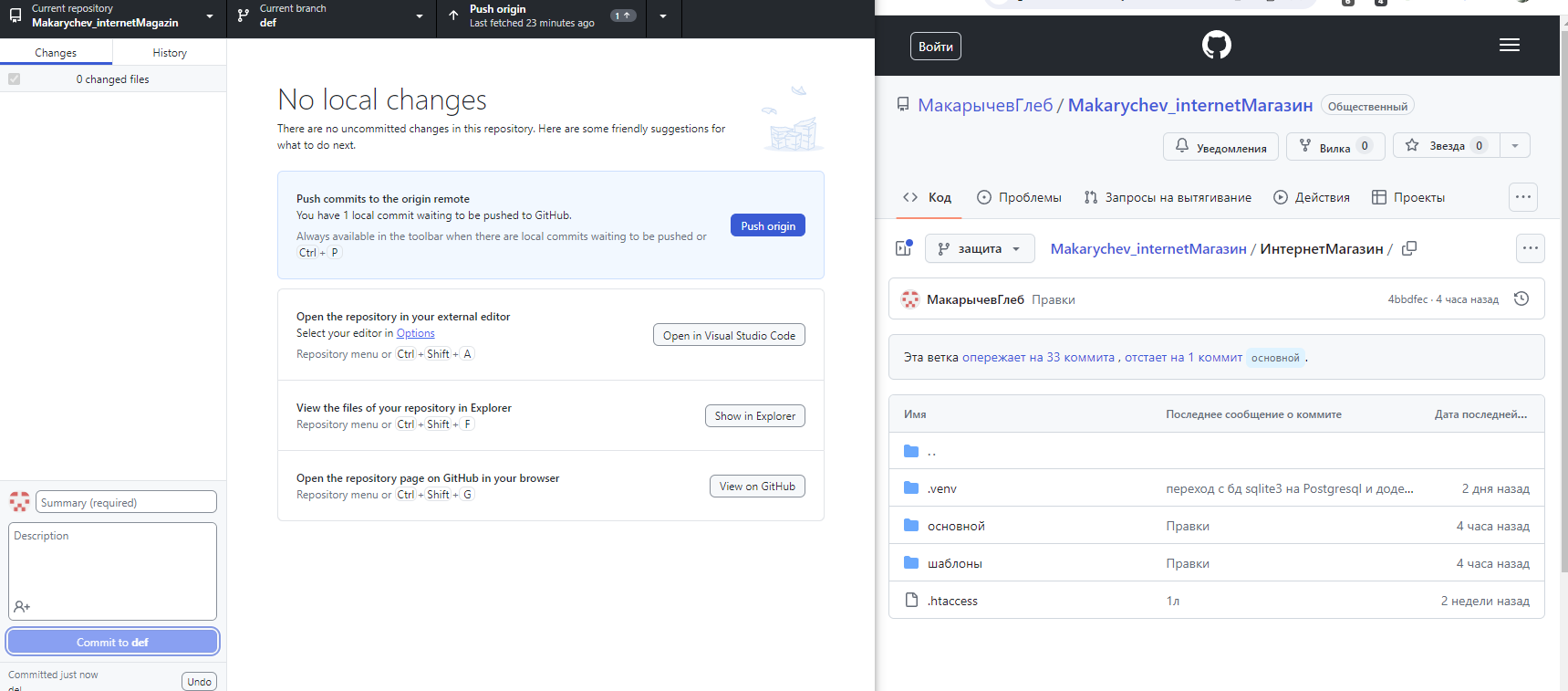


Рис.1.1.1- GitD

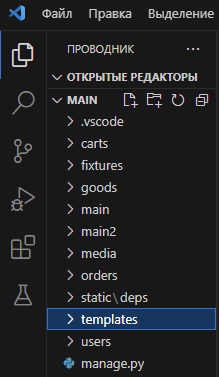


Рис.1.2.3.1-main каталог

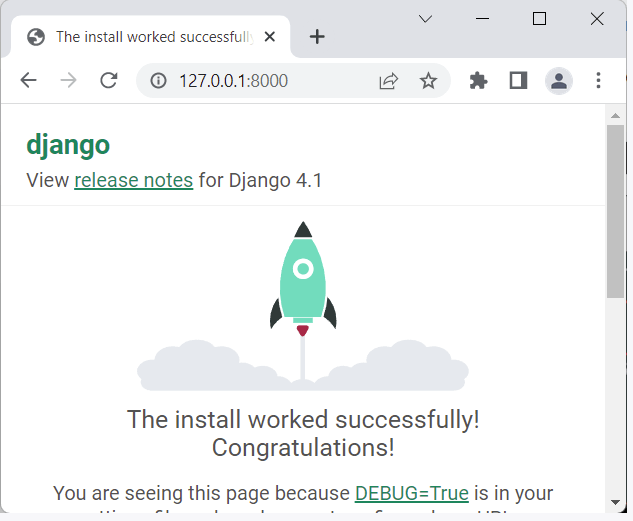


Рис.1.2.3.2-startserver

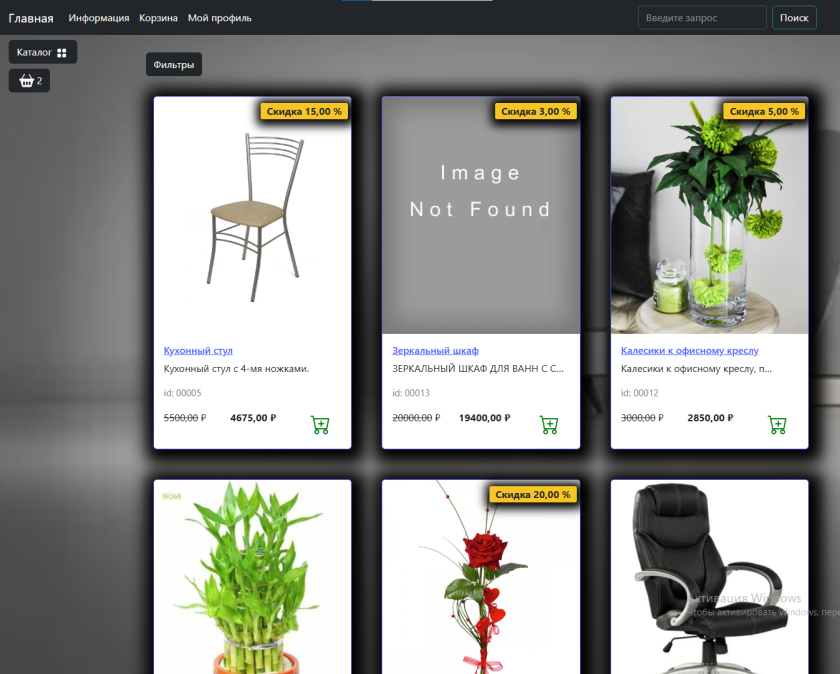


Рис.2.4.1-base.html

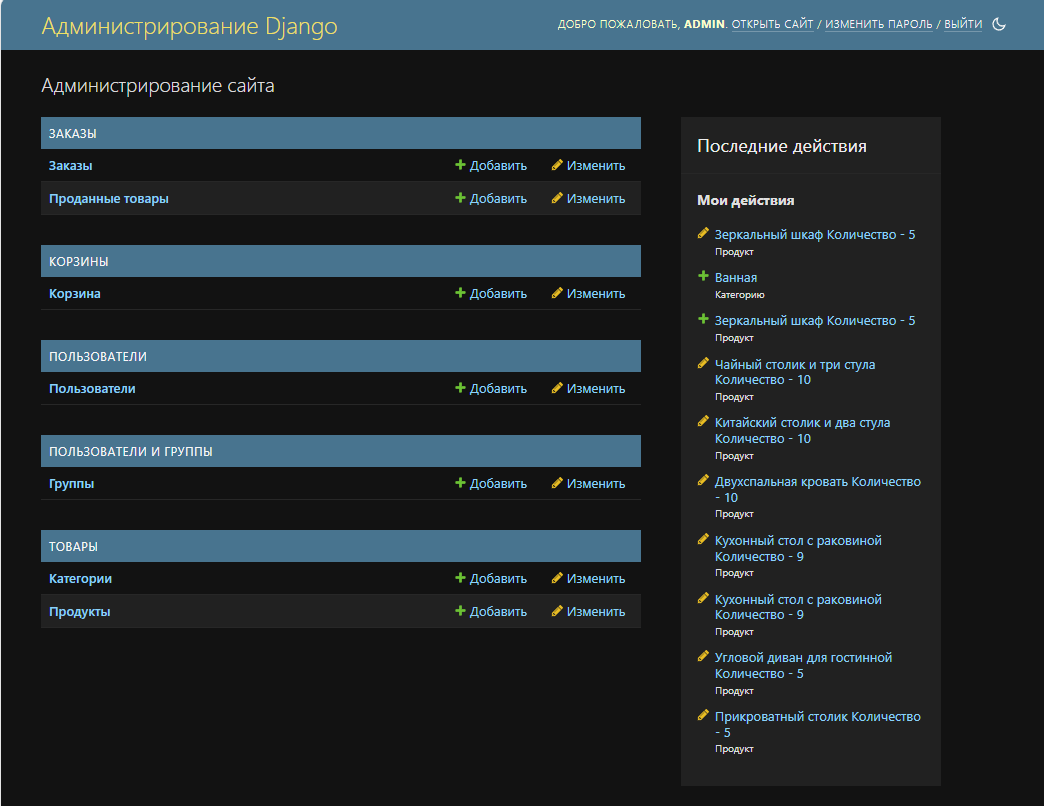


Рис.2.4.1.1-admin

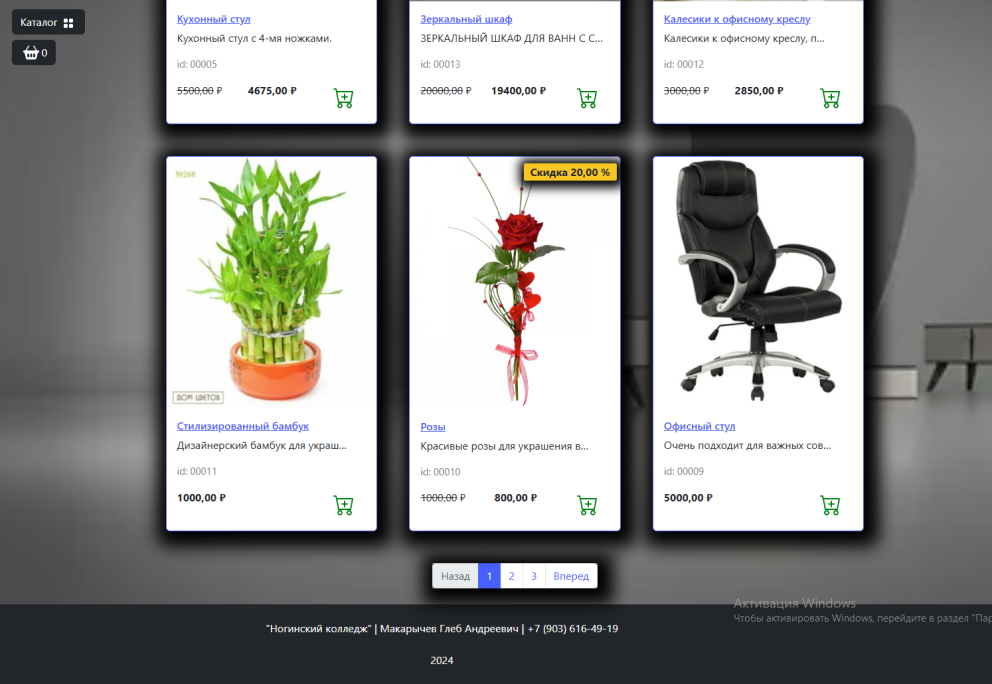


Рис.2.4.2.4-page

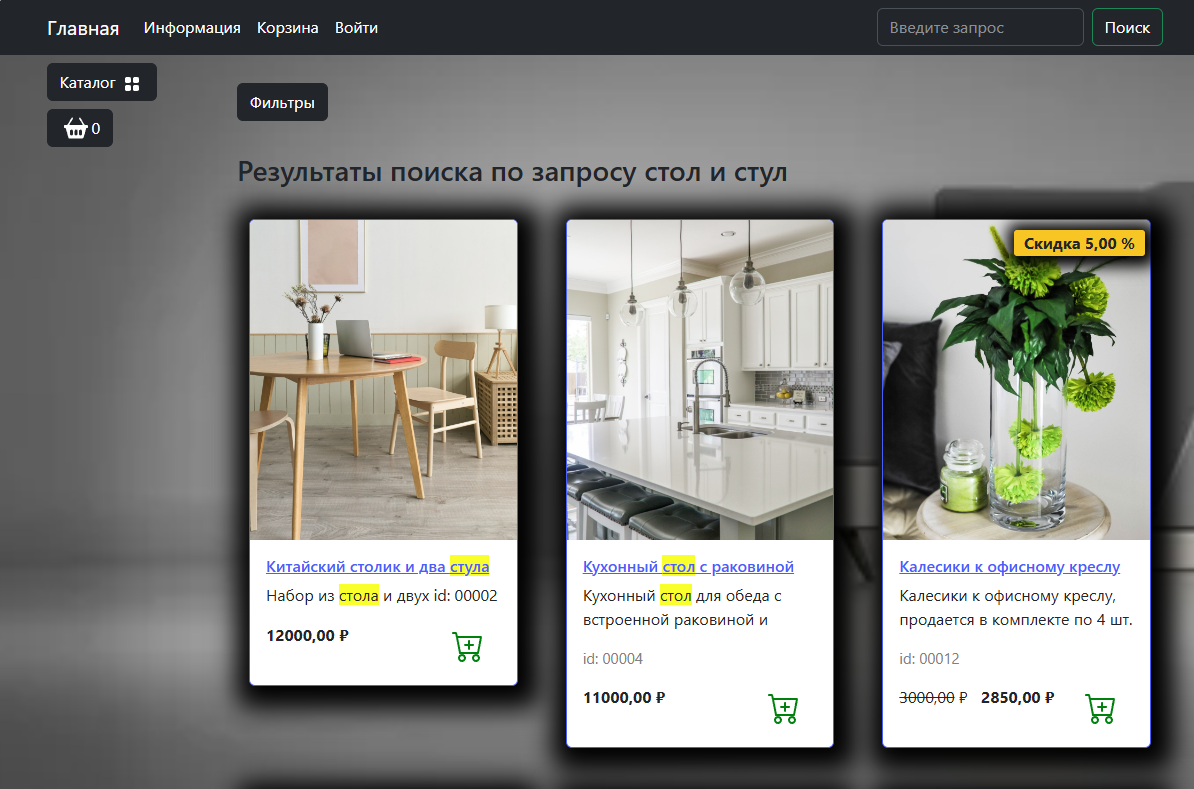


Рис.2.4.2.5- Search-yellow

# 

Рис.2.4.2.6- Search-none

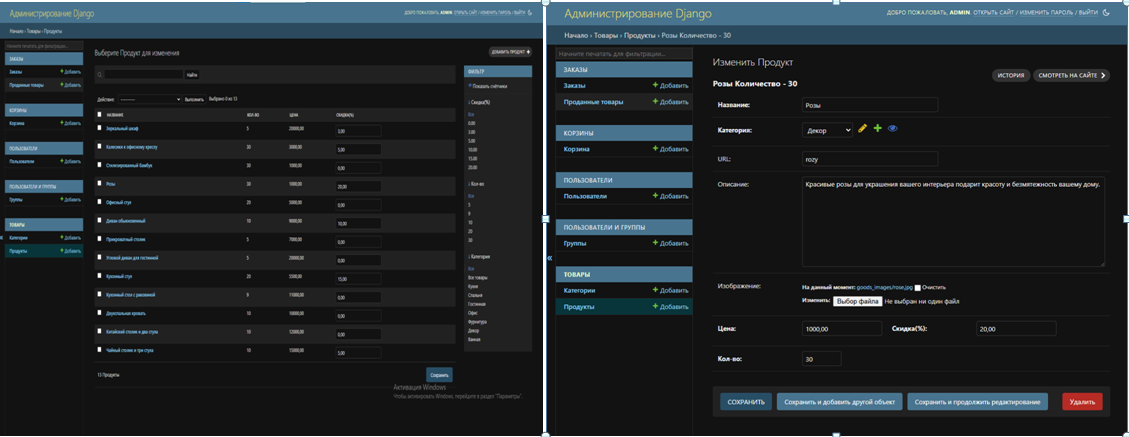


Рис.2.4.2.7-admin/goods

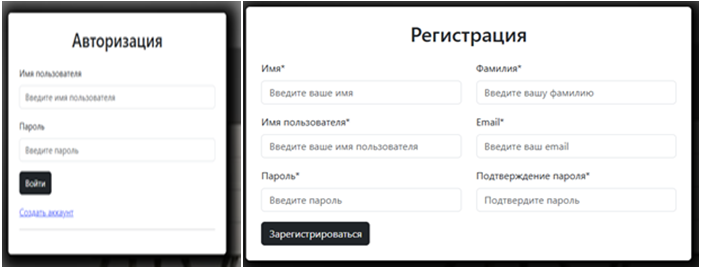


Рис.2.4.3.1-login/word

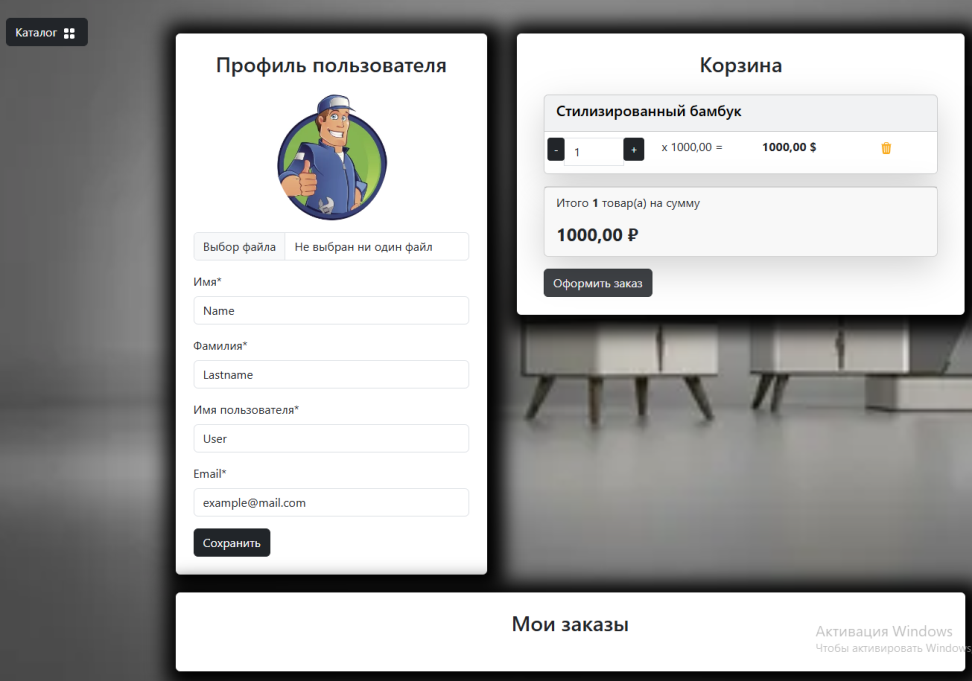


Рис.2.4.3.2-profile

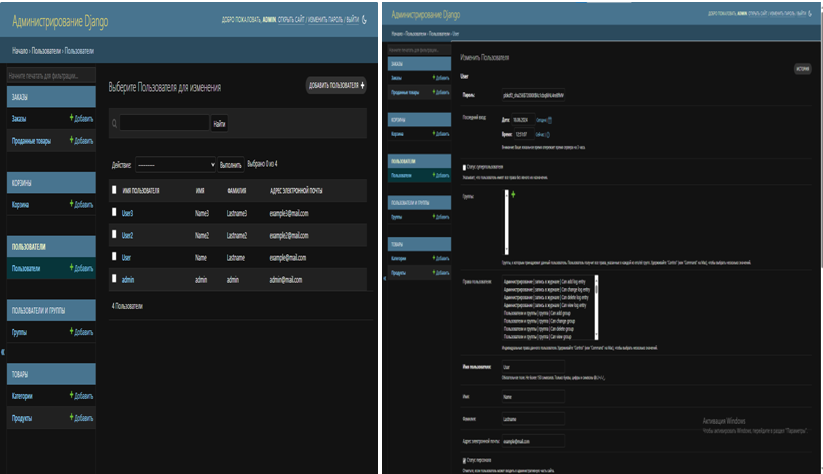


Рис.2.4.3.3-admin/users

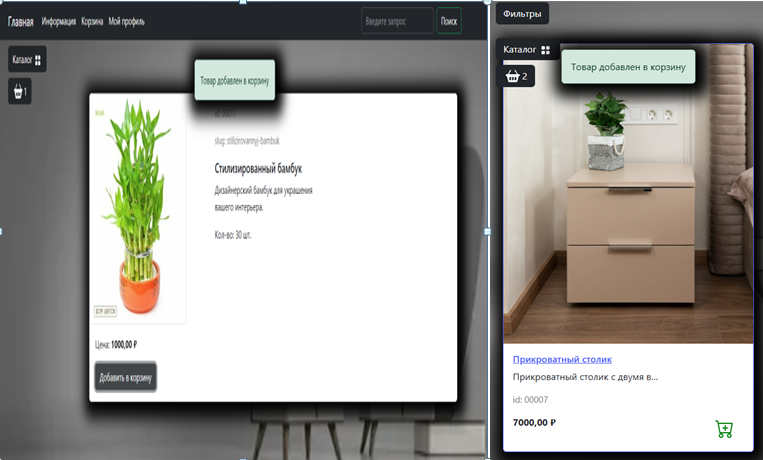


Рис.2.4.4.1-carts/product

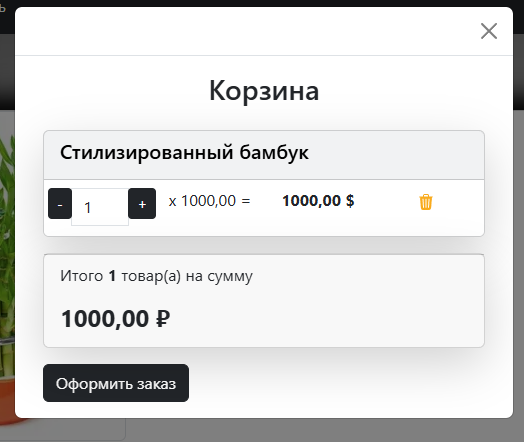


Рис.2.4.4.3-carts/product

# 

Рис.2.4.5.1-oder/forms

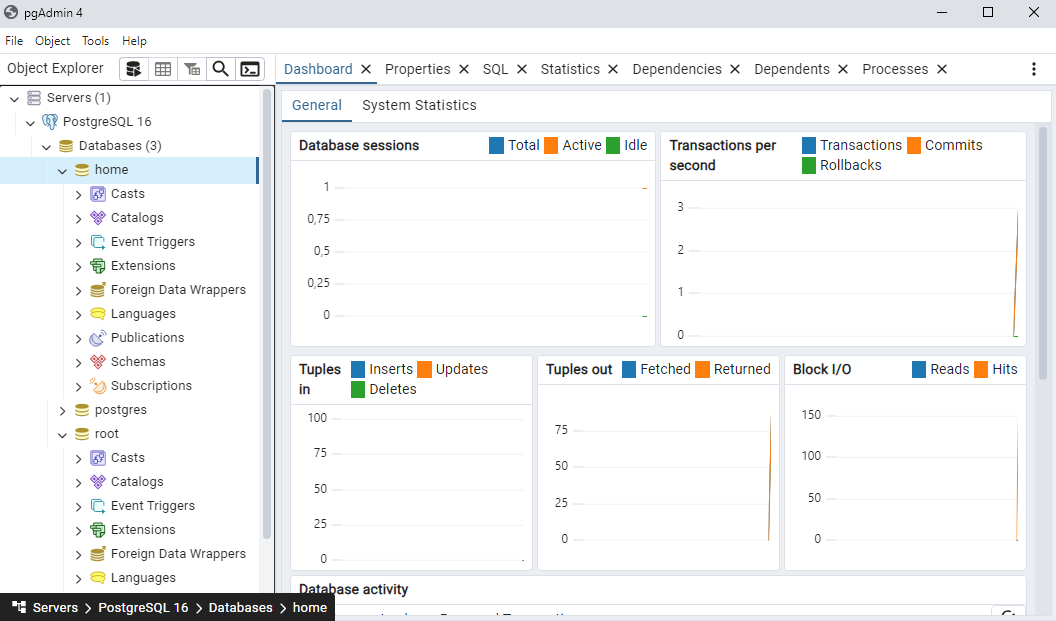
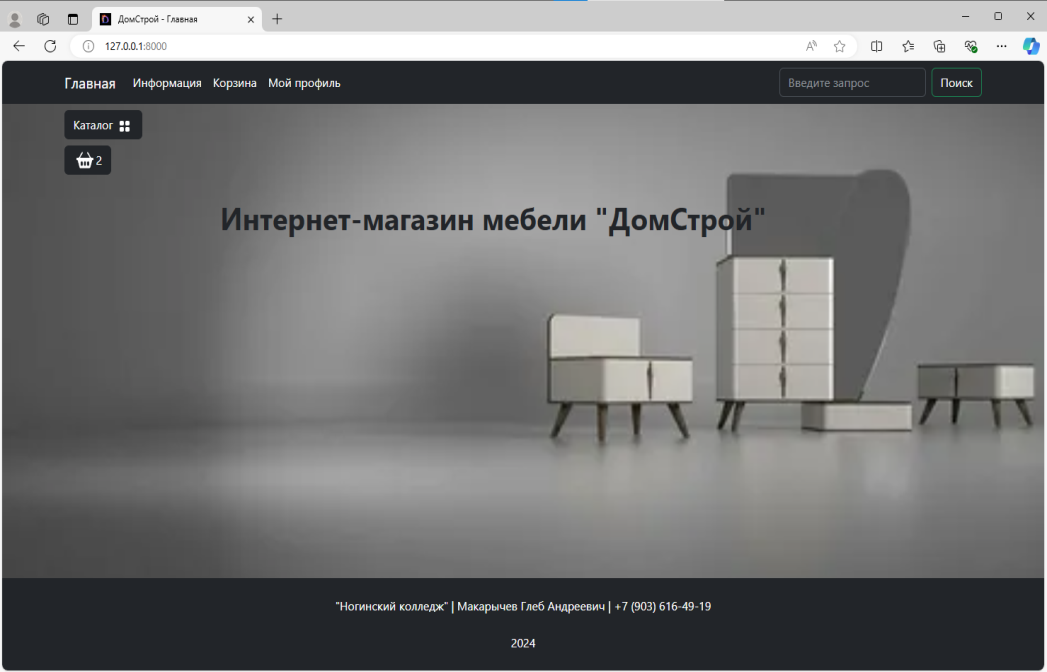
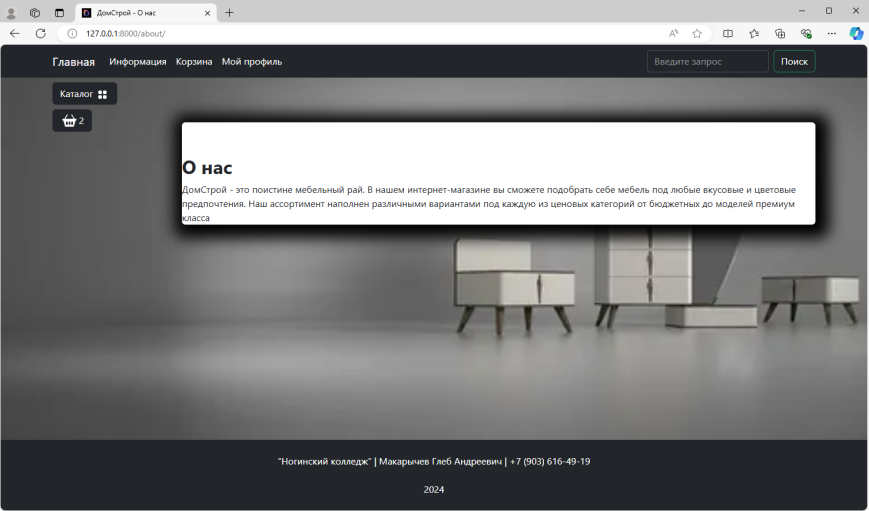
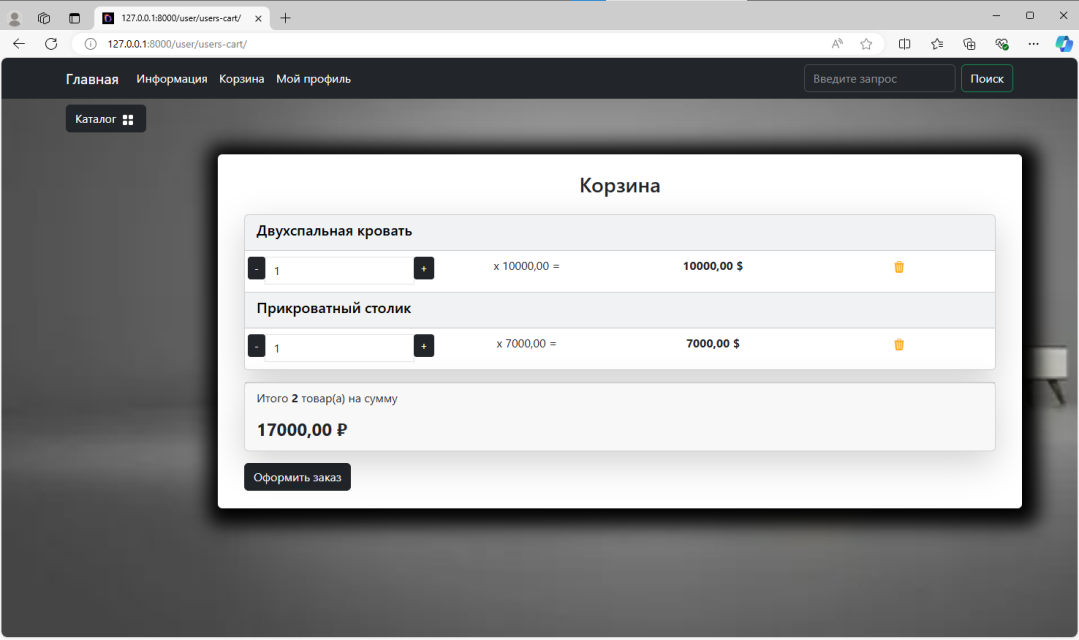
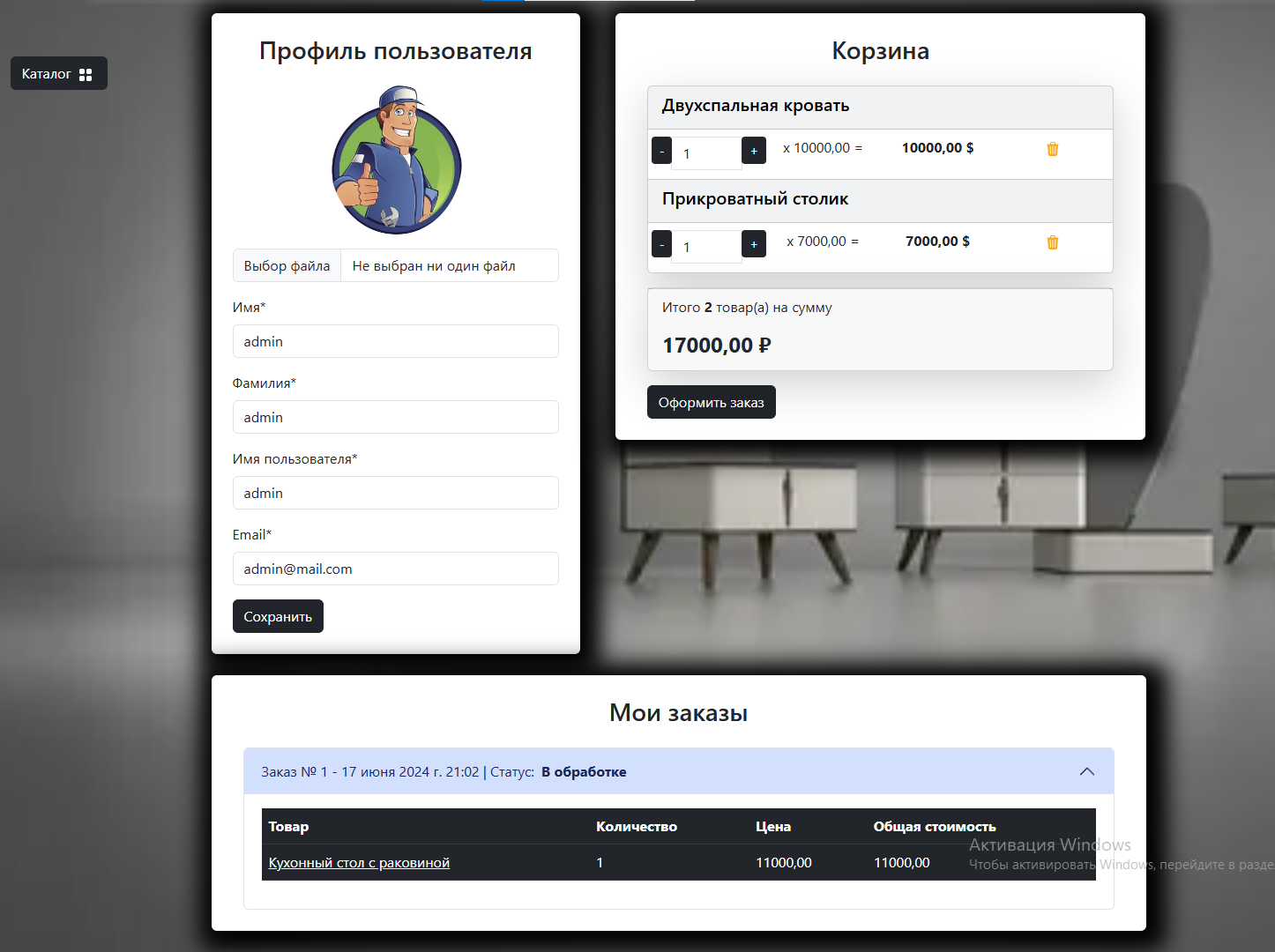


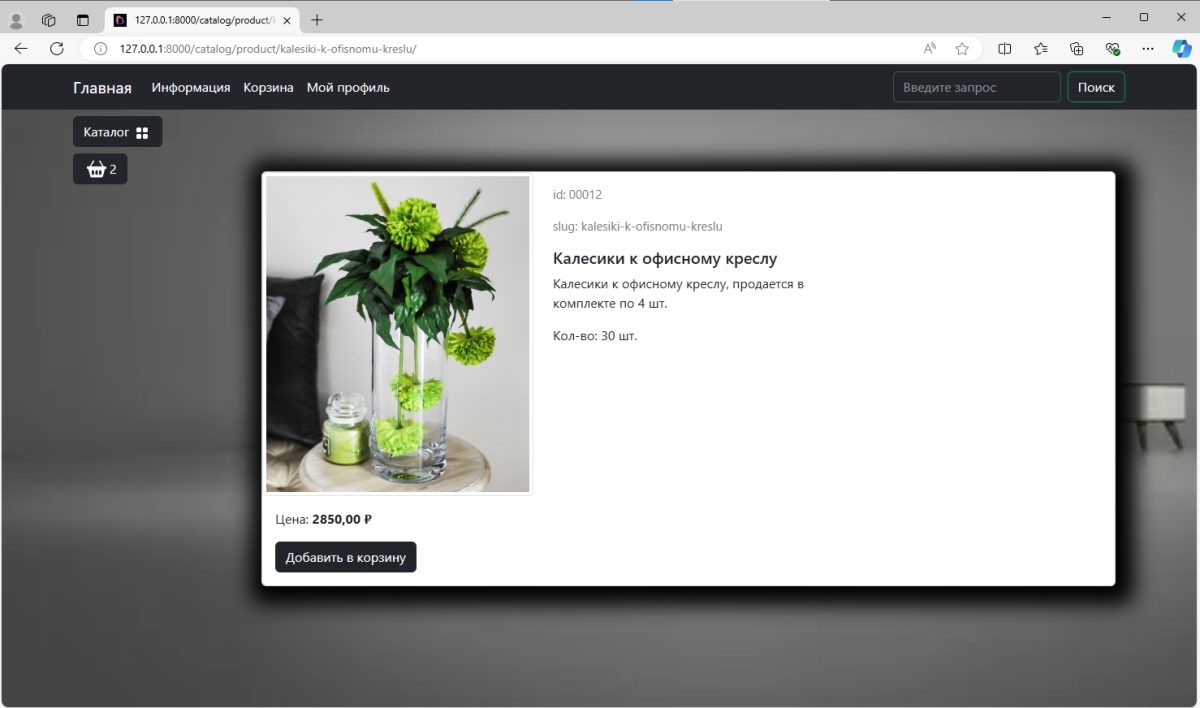
Рис.2.5.1-ДБ/ PostgreSQL

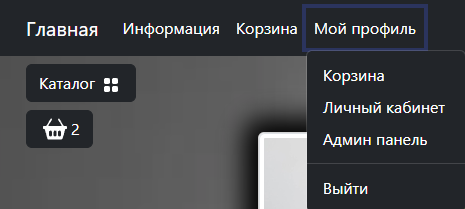












# 

# 

# 

# 

# 

# 

# ПРИЛОЖЕНИЕ 2

from pathlib import Path

from django.http import HttpResponseNotFound

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

SECRET\_KEY = 'django-insecure-gw\_-v+upl=+=o0\_3uuergtca$)\*#d+4s%qgy%^c9d\*4!vqn^r%'

DEBUG = True

ALLOWED\_HOSTS = ['\*']

INSTALLED\_APPS = ['django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'django.contrib.postgres',

    'main2',

    'goods',

    'users',

    'carts',

    'orders']

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',]

ROOT\_URLCONF = 'main.urls'

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 = 'main.wsgi.application'

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.postgresql',

        'NAME': 'home',

        'USER': 'home',

        'PASSWORD': 'home',

        'HOST': 'localhost',

        'PORT': '5432', }}

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', },

]

LANGUAGE\_CODE = 'ru'

TIME\_ZONE = 'UTC'

USE\_I18N = True

USE\_TZ = True

STATIC\_URL = 'static/'

STATICFILES\_DIRS = [

    BASE\_DIR / 'static'

    ]

MEDIA\_URL = 'media/'

MEDIA\_ROOT = BASE\_DIR / 'media'

DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'

AUTH\_USER\_MODEL = 'users.User'

LOGIN\_URL = '/user/login/'

2.4.2- main/Settings.py

from django.contrib import admin

from django.urls import path, include

from django.conf.urls.static import static

from main import settings

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', include('main2.urls', namespace='main2')),

    path('catalog/', include('goods.urls', namespace='catalog')),

    path('user/', include('users.urls', namespace='user')),

    path('cart/', include('carts.urls', namespace='cart')),

    path('orders/', include('orders.urls', namespace='orders')),

]

if settings.DEBUG:

    urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

2.4.3- main/urls.py

From jango.shortcuts import render

def index(request):

    context = {

        'title': 'ДомСтрой – Главная',

        'content': 'Интернет-магазин мебели «ДомСтрой»',

    }

    return render(request, ‘main2/index.html’, context)

def about(request):

    context = {

        ‘title’: ‘ДомСтрой – О нас’,

        'content': 'О нас',

        'text\_on\_page': ' ДомСтрой – это поистине мебельный рай. '

    }

    return render(request, 'main2/about.html', context)

2.4.1.1-main2/views.py

class Categories(models.Model):

    name = models.CharField(max\_length=100, unique=True, verbose\_name='Название')

    slug = models.SlugField(max\_length=200, unique=True, blank=True, null=True, verbose\_name='URL')

class Products(models.Model):

    name = models.CharField(max\_length=100, unique=True, verbose\_name='Название')

    slug = models.SlugField(max\_length=200, unique=True, blank=True, null=True, verbose\_name='URL')

    description = models.TextField(blank=True, null=True, verbose\_name='Описание')

    image = models.ImageField(upload\_to='goods\_images', blank=True, null=True, verbose\_name='Изображение')

    price = models.DecimalField(default=0.00, max\_digits=8, decimal\_places=2, verbose\_name='Цена')

    discount = models.DecimalField(default=0.00, max\_digits=4, decimal\_places=2, verbose\_name='Скидка(%)')

    quantity = models.PositiveIntegerField(default=0, verbose\_name='Кол-во')

    category = models.ForeignKey(to=Categories, on\_delete=models.CASCADE, verbose\_name='Категория')

2.4.2.1-goods/models.py

from django.core.paginator import Paginator

paginator = Paginator(goods, 6)

    current\_page = paginator.page(int(page))

    context = {

        'title': 'ДомСтрой - Каталог',

        'goods': current\_page,

        "slug\_url": category\_slug}

 return render(request, 'goods/catalog.html', context)

2.4.2.2-goods/views.py

from django.shortcuts import get\_list\_or\_404, render

from .models import Products

from .utils import q\_search

def catalog(request, category\_slug=None):

    page = request.GET.get('page', 1)

    on\_sale = request.GET.get('on\_sale', None)

    order\_by = request.GET.get('order\_by', None)

    query = request.GET.get('q', None)

    if category\_slug == 'all':

        goods = Products.objects.all()

    elif query:

        goods = q\_search(query)

    else:

        goods = get\_list\_or\_404(

            Products.objects.filter(category\_\_slug=category\_slug)

            )

    if on\_sale:

        goods = goods.filter(discount\_\_gt=0)

    if order\_by and order\_by != "default":

        goods = goods.order\_by(order\_by)

2.4.2.3-goods/views.py

from django.db import models

from django.contrib.auth.models import AbstractUser

class User (AbstractUser):

    image = models.ImageField(upload\_to='users\_images', blank=True, null=True, verbose\_name='Аватар')

    phone\_number = models.CharField(max\_length=10, blank=True, null=True)

    class Meta:

        db\_table = "user"

        verbose\_name = "Пользователя"

        verbose\_name\_plural = "Пользователи"

    def \_\_str\_\_(self):

        return self.username

2.4.3.2-users/models.py

    def total\_quantity(self):

        if self:

            return sum(cart.quantity for cart in self)

        return 0

class Cart(models.Model):

    user = models.ForeignKey(User, on\_delete=models.CASCADE, blank=True, null=True, verbose\_name='Пользователь')

    product = models.ForeignKey(Products, on\_delete=models.CASCADE, verbose\_name='Товар')

    quantity = models.IntegerField(default=0, verbose\_name='Количество')

    session\_key = models.CharField(max\_length=32, blank=True, null=True)

    created\_timestamp = models.DateTimeField(auto\_now\_add=True, verbose\_name='Дата добавления')

    class Meta:

        db\_table = "cart"

        verbose\_name = "Корзина"

        verbose\_name\_plural = "Корзина"

        ordering = ("id",)

    objects = CartQueryset().as\_manager()

    def products\_price(self):

        return round(self.product.sell\_price() \* self.quantity, 2)

    def \_\_str\_\_(self):

        if self.user:

            return f'Корзина {self.user.username} | Товар {self.product.name} | Количество {self.quantity}'

        return f'Анонимная корзина | Товар {self.product.name} | Количество {self.quantity}'

2.4.4.2-carts/models.py

from django.db import models

from goods.models import Products

from users.models import User

class CartQueryset(models.QuerySet):

    def total\_price(self):

        return sum(cart.products\_price() for cart in self)

    def total\_quantity(self):

        if self:

            return sum(cart.quantity for cart in self)

        return 0

class Cart(models.Model):

    user = models.ForeignKey(User, on\_delete=models.CASCADE, blank=True, null=True, verbose\_name='Пользователь')

    product = models.ForeignKey(Products, on\_delete=models.CASCADE, verbose\_name='Товар')

    quantity = models.IntegerField(default=0, verbose\_name='Количество')

    session\_key = models.CharField(max\_length=32, blank=True, null=True)

    created\_timestamp = models.DateTimeField(auto\_now\_add=True, verbose\_name='Дата добавления')

    class Meta:

        db\_table = "cart"

        verbose\_name = "Корзина"

        verbose\_name\_plural = "Корзина"

        ordering = ("id",)

    objects = CartQueryset().as\_manager()

    def products\_price(self):

        return round(self.product.sell\_price() \* self.quantity, 2)

    def \_\_str\_\_(self):

        if self.user:

            return f'Корзина {self.user.username} | Товар {self.product.name} | Количество {self.quantity}'

        return f'Анонимная корзина | Товар {self.product.name} | Количество {self.quantity}'

2.4.4.4-carts/models.py

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

from django.contrib import messages

from django.db import transaction

from django.forms import ValidationError

from django.shortcuts import redirect, render

from carts.models import Cart

from orders.forms import CreateOrderForm

from orders.models import Order, OrderItem

@login\_required

def create\_order(request):

    if request.method == "POST":

        form = CreateOrderForm(data=request.POST)

        if form.is\_valid():

            try:

                with transaction.atomic():

                    user = request.user

                    cart\_items = Cart.objects.filter(user=user)

                    if cart\_items.exists():

                        order = Order.objects.create(

                            user=user,

                            phone\_number=form.cleaned\_data["phone\_number"],

                            requires\_delivery=form.cleaned\_data["requires\_delivery"],

                            delivery\_address=form.cleaned\_data["delivery\_address"],

                            payment\_on\_get=form.cleaned\_data["payment\_on\_get"],

                        )

                        for cart\_item in cart\_items:

                            product = cart\_item.product

                            name = cart\_item.product.name

                            price = cart\_item.product.sell\_price()

                            quantity = cart\_item.quantity

                            if product.quantity < quantity:

                                raise ValidationError(

                                    f"Недостаточное количество товара {name} на складе\

                                    В наличии - {product.quantity}"

                                )

                            OrderItem.objects.create(

                                order=order,

                                product=product,

                                name=name,

                                price=price,

                                quantity=quantity,

                            )

                            product.quantity -= quantity

                            product.save()

                        cart\_items.delete()

                        messages.success(request, "Заказ оформлен!")

                        return redirect("user:profile")

            except ValidationError as e:

                messages.success(request, str(e))

                return redirect("orders:create\_order")

    else:

        initial = {

            "first\_name": request.user.first\_name,

            "last\_name": request.user.last\_name,

        }

        form = CreateOrderForm(initial=initial)

    context = {

        "title": "Home - Оформление заказа",

        "form": form,

        "order": True,

    }

    return render(request, "orders/create\_order.html", context=context)

2.4.5.2-orders/views.py

import re

from django import forms

class CreateOrderForm(forms.Form):

    first\_name = forms.CharField()

    last\_name = forms.CharField()

    phone\_number = forms.CharField()

    requires\_delivery = forms.ChoiceField(

        choices=[("0", False), ("1", True), ],

        )

    delivery\_address = forms.CharField(required=False)

    payment\_on\_get = forms.ChoiceField(

        choices=[ ("0", 'False'), ("1", 'True'), ],

        )

    def clean\_phone\_number(self):

        data = self.cleaned\_data['phone\_number']

        if not data.isdigit():

            raise forms.ValidationError("Номер телефона должен содержать только цифры")

        pattern = re.compile(r'^\d{10}$')

        if not pattern.match(data):

            raise forms.ValidationError("Неверный формат номера")

        return data

2.4.5.3-orders/forms.py

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.postgresql',

        'NAME': 'home',

        'USER': 'home',

        'PASSWORD': 'home',

        'HOST': 'localhost',

        'PORT': '5432',

    }

}

2.5.2-дб/url