**Оглавление.**

[ВВЕДЕНИЕ. 2](#_Toc74684991)-3

[1.](#_Toc74684992)[Веб-сервисы интернет-магазинов. 3](#_Toc74684992)

[1.1. История создания Django. 3](#_Toc74684993)

[1.2. Характеристика веб-сервисов интернет-магазинов. 3](#_Toc74684993)

[1.3. Требования к инструментам, оборудованию и инфраструктуре. 4](#_Toc74684994)

[1.4. Охрана труда и организация рабочего места разработчика. 5](#_Toc74684995)

[2. РАЗРАБОТКА ВЕБ-СЕРВИСА. 11](#_Toc74684998)

[2.1.Подготовка рабочей среды . 11](#_Toc74684999)

[2.2. Установка веб фреймворка Django и создание проекта. 12](#_Toc74685000)

[2.3. Структура проекта на Django. 13](#_Toc74685001)

[2.4. Архитектура Django и создание приложения для интернет-магазина автозапчастей. 16](#_Toc74685002)

[2.5. Добавление товарных позиций. 17](#_Toc74685003)

[2.6. Структура сайта. 22](#_Toc74685004)

[ЗАКЛЮЧЕНИЕ. 27](#_Toc74685005)

[СПИСОК ЛИТЕРАТУРЫ. 28](#_Toc74685006)

**ВВЕДЕНИЕ.**

**Актуальность темы**

В современном мире электронная коммерция стала неотъемлемой частью глобальной экономики. По данным статистики, объем мирового рынка интернет-торговли в 2023 году превысил 6,3 триллиона долларов США, а к 2027 году ожидается рост до 8,1 триллиона. В условиях цифровой трансформации и пандемийных ограничений разработка эффективных веб-сервисов для интернет-магазинов приобретает особую значимость.

Современные потребители ожидают от электронной коммерции не только удобства совершения покупок, но и персонализированного подхода, высокой скорости работы, безопасности транзакций и мобильной адаптации. Эти требования обуславливают необходимость создания технологически сложных, но при этом удобных для пользователей решений.

**Цель и задачи исследования**

Целью данной дипломной работы является разработка полнофункционального веб-сервиса интернет-магазина с использованием современных технологий веб-разработки.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести анализ существующих решений в области электронной коммерции

2. Определить оптимальный стек технологий для реализации проекта

3. Разработать архитектуру веб-сервиса

4. Реализовать основные функциональные модули:

- Систему управления товарами

- Пользовательскую корзину

- Механизм оформления заказов

- Систему аутентификации и авторизации

- Панель администратора

5. Обеспечить безопасность и производительность системы

6. Протестировать работоспособность всех компонентов

**Объект и предмет исследования**

Объектом исследования выступают веб-сервисы электронной коммерции и технологии их разработки.

Предметом исследования являются методы и подходы к созданию интернет-магазина с использованием современных веб-технологий.

**Методы исследования**

В работе применялись следующие методы исследования:

- Анализ существующих решений и литературных источников

- Сравнительный анализ технологических решений

- Проектирование архитектуры системы

- Программная реализация с использованием современных фреймворков

- Тестирование производительности и безопасности

**Практическая значимость**

Разработанный веб-сервис может быть использован:

- Как готовое решение для малого и среднего бизнеса

- В качестве основы для более сложных систем электронной коммерции

- Как учебный пример реализации полнофункционального интернет-магазина

- Для дальнейшего исследования и оптимизации технологий веб-разработки

1. **Веб-сервис интернет-магазинов**

**1.1 История создания Django**

Django — высокоуровневый веб-фреймворк на Python, известный своей простотой, безопасностью и скоростью разработки. Его история тесно связана с практическими задачами журналистики и эволюцией веб-технологий в начале 2000-х.

1. Предпосылки создания\*\*

В 2003 году команда разработчиков газеты \*\*Lawrence Journal-World\*\* (Канзас, США) столкнулась с необходимостью быстро создавать и обновлять веб-приложения для публикации новостей. Основные проблемы, которые они решали:

- Сжатые сроки: Журналистам требовалось оперативно добавлять контент, не завися от программистов.

- Повторяющиеся задачи: Ручная настройка админ-панелей, форм и баз данных замедляла работу.

- Масштабируемость: Существующие инструменты не подходили для растущего трафика и сложных проектов.

Команда, включая Адриана Головатого и Саймона Уиллисона, начала разрабатывать внутренний фреймворк, который автоматизировал рутину и ускорял разработку.

2. Рождение Django

- 2003–2005: Фреймворк использовался внутри газеты для создания таких проектов, как LJWorld.com и KUsports.com .

- Июль 2005 : Код был открыт под лицензией BSD на сайте djangoproject.com . Название Django — дань уважения джазовому гитаристу Джанго Рейнхардту , чья музыка вдохновляла разработчиков.

- 2008 : Вышла первая стабильная версия Django 1.0 , закрепившая ключевые принципы: ORM, автоматическая админка, система шаблонов.

3. Ключевые этапы развития



4. Философия Django

- «Batteries included» : Готовые модули для аутентификации, сессий, кэширования и админки.

- DRY (Don’t Repeat Yourself) : Минимизация дублирования кода.

- Convention over Configuration : Соглашения упрощают настройку (например, имена моделей и таблиц в БД).

- Безопасность : Защита от SQL-инъекций, XSS, CSRF «из коробки».

5. Влияние на индустрию

- Крупные проекты :

- Instagram : Обрабатывает миллионы запросов в день.

- Pinterest : Использует Django для API и управления контентом.

- Mozilla , NASA , The Washington Post .

- Сообщество :

- Более 2500 пакетов на PyPI (например, Django REST Framework, Django CMS).

- Ежегодные конференции ( DjangoCon , Django Girls ).

- Образование : Django часто выбирают для обучения из-за понятной структуры и документации.

6. Интересные факты

- Админка Django изначально создавалась для журналистов, чтобы они могли редактировать контент без знания программирования.

- В 2008 году основан Django Software Foundation для поддержки и развития проекта.

- Джанго Рейнхардт , в честь которого назван фреймворк, играл на гитаре двумя пальцами из-за травмы руки — это стало метафорой «элегантности вопреки ограничениям».

7. Почему Django актуален сегодня?

- Скорость разработки : Готовые решения для типовых задач (регистрация, корзина покупок, блоги).

- Масштабируемость : Подходит как для стартапов, так и для высоконагруженных систем.

- Безопасность : Регулярные обновления и встроенные механизмы защиты.

- Гибкость : Интеграция с React, Vue.js, микросервисной архитектурой.

Django прошел путь от внутреннего инструмента газеты до одного из ведущих фреймворков в мире. Его история — пример того, как практические задачи могут породить технологию, меняющую подход к веб-разработке. Для вашей дипломной работы выбор Django обоснован его надежностью, богатой экосистемой и поддержкой сообщества, что делает его идеальным для создания интернет-магазина.

### **1.2.** **Характеристика** **веб-сервисов интернет-магазинов**

Современные веб-сервисы интернет-магазинов представляют собой сложные программные комплексы, обеспечивающие полный цикл электронной коммерции: от каталога товаров до оформления заказа, оплаты и доставки.

#### **Классификация интернет-магазинов**

1. **По масштабу:**
   * Локальные (малый бизнес, узкая специализация)
   * Региональные и национальные (средний и крупный бизнес)
   * Международные маркетплейсы (Amazon, AliExpress)
2. **По типу товаров:**
   * Физические товары (электроника, одежда, продукты)
   * Цифровые товары (программы, книги, курсы)
   * Услуги (онлайн-запись, подписки)
3. **По технологии реализации:**
   * Готовые CMS (OpenCart, WooCommerce, Shopify)
   * Самописные решения (индивидуальная разработка)
   * Гибридные системы (Headless Commerce + API)

#### **Основные модули интернет-магазина**

* **Фронтенд:** пользовательский интерфейс (каталог, корзина, личный кабинет)
* **Бэкенд:** управление товарами, заказами, пользователями
* **База данных:** хранение информации о товарах, клиентах, транзакциях
* **Платежные системы:** интеграция с банками, электронными кошельками
* **Системы логистики:** расчет доставки, трекинг заказов

#### **Используемые технологии**

* **Frontend:** HTML/CSS, JavaScript, React/Vue/Angular, Bootstrap/Tailwind
* **Backend:**  Python (Django), Node.js (Express), Java (Spring)
* **Базы данных:**  PostgreSQL
* **Дополнительные сервисы:** Redis (кеширование), Elasticsearch (поиск), RabbitMQ (очереди)

### **1.3.** **Требования к инструментам, оборудованию и инфраструктуре**

Для разработки и эксплуатации веб-сервиса интернет-магазина необходимо обеспечить:

#### **1. Программное обеспечение и инструменты**

* **IDE / Редакторы кода:** Visual Studio Code
* **Системы контроля версий:** Git (GitHub, GitLab, Bitbucket)
* **Серверное ПО:**
  + Веб-сервер (Nginx, Apache)
  + СУБД (MySQL, PostgreSQL)
  + PHP/Python/Node.js (в зависимости от стека)
* **Дополнительные инструменты:**
  + Docker (контейнеризация)
  + Composer/npm/pip (менеджеры пакетов)
  + Postman (тестирование API)

#### **2. Аппаратное обеспечение**

* **Для разработки:**
  + Компьютер с ОС Windows/macOS/Linux (8+ ГБ ОЗУ, SSD)
  + Доступ в интернет
* **Для хостинга:**
  + Виртуальный сервер (VPS) или облачные решения (AWS, Google Cloud)
  + Минимальные требования: 2+ ядра CPU, 4+ ГБ ОЗУ, 50+ ГБ SSD

#### **3. Требования к помещению (для офисной разработки)**

* **Эргономика рабочего места:**
  + Удобное кресло и стол
  + Монитор с антибликовым покрытием
  + Освещение, соответствующее нормам (не менее 300 люкс)
* **Безопасность и комфорт:**
  + Нормальная температура (20–24°C)
  + Вентиляция/кондиционирование
  + Уровень шума не выше 50 дБ

### 1.3. Охрана труда и организация рабочего места разработчика

#### **Основные понятия**

* **Охрана труда** – система сохранения жизни и здоровья работников в процессе трудовой деятельности.
* **Техника безопасности** – комплекс мер для предотвращения травм и профессиональных заболеваний.

#### **Требования к рабочей зоне**

1. **Освещение:**
   * Естественное + искусственное (LED-лампы 4000–5000K)
   * Отсутствие бликов на экране
2. **Микроклимат:**
   * Температура: 20–24°C
   * Влажность: 40–60%
   * Регулярное проветривание
3. **Эргономика:**
   * Кресло с поддержкой поясницы
   * Монитор на уровне глаз (расстояние 50–70 см)
   * Клавиатура и мышь в удобном положении

#### **Техника безопасности при разработке**

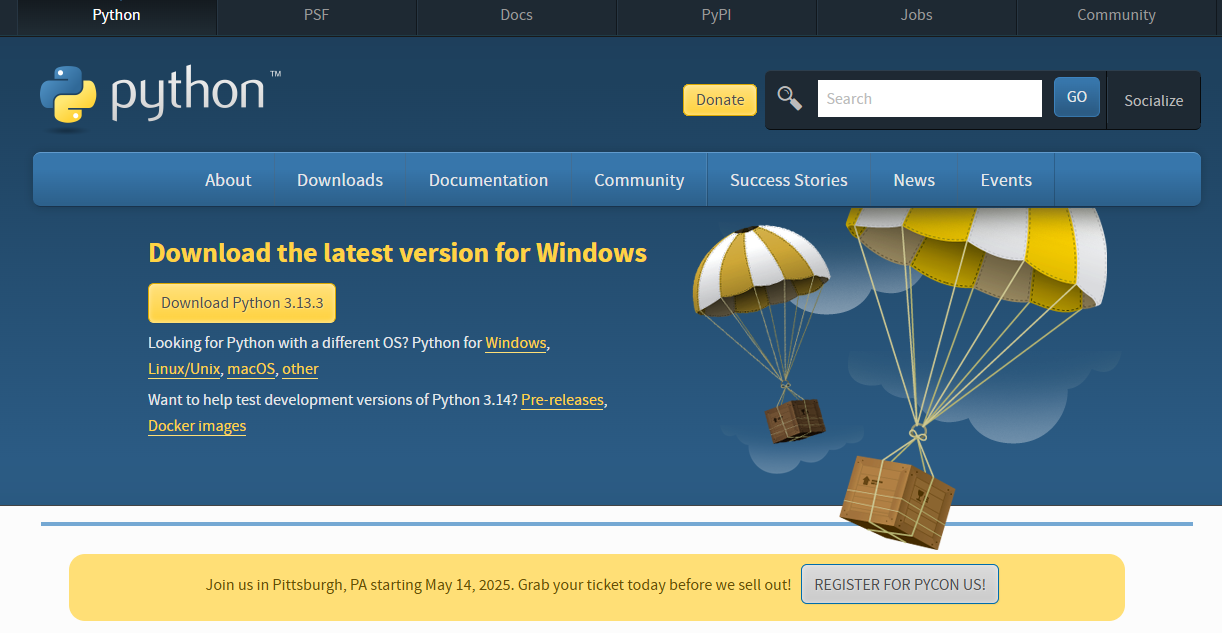
* Делать перерывы каждые 45–60 минут (правило 20-20-20: каждые 20 минут смотреть на 20 футов вдаль 20 секунд)
* Использовать антистатический браслет при работе с электроникой
* Не допускать перегрузки электросети (ИБП для серверов)

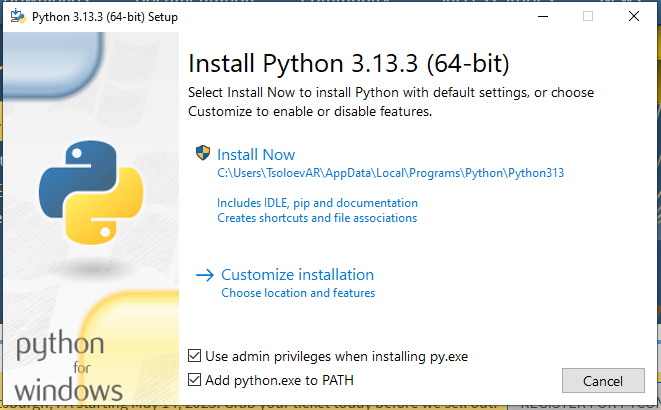
1. **Разработка веб сервиса**

2.1.Подготовка рабочей среды

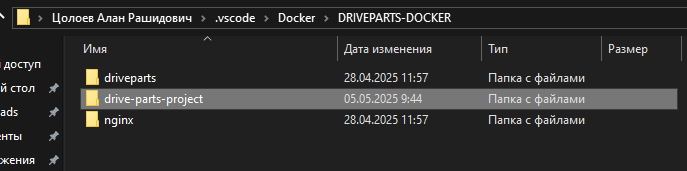
После того как мы установили и настроили нашу IDE, нам потребуется  
установить язык программирования Python и настроить виртуальное окружение для изолированной работы проекта.

Скачиваем и устанавливаем последнюю версию Python.





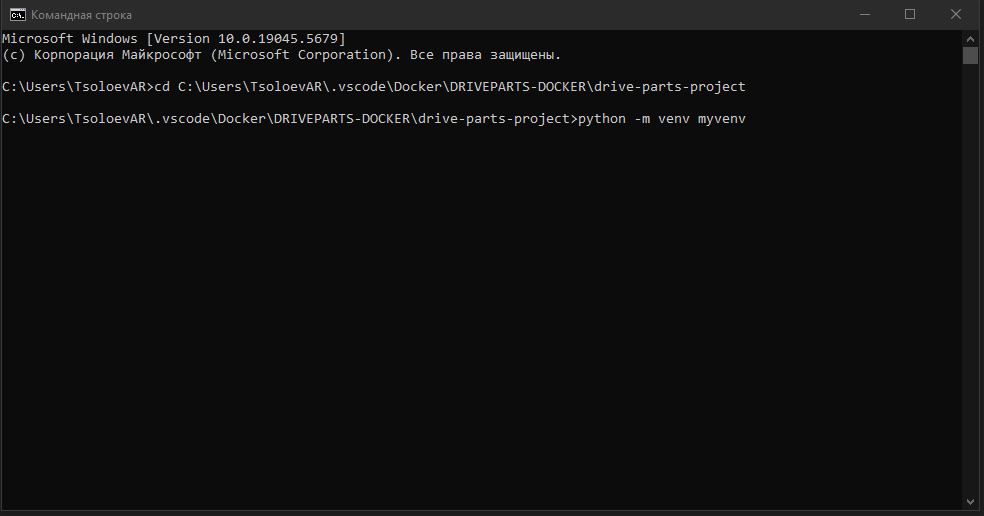
Дальше приступаем к созданию корневой директории проекта.



Назовём её drive-parts-project и приступим к созданию виртуального окружения внутри неё.

***Виртуальное окружение Python*** (Python virtual environment) – это инструмент, позволяющий создавать изолированные среды для выполнения и разработки приложений на языке Python.

Запускаем командную строку Windows.

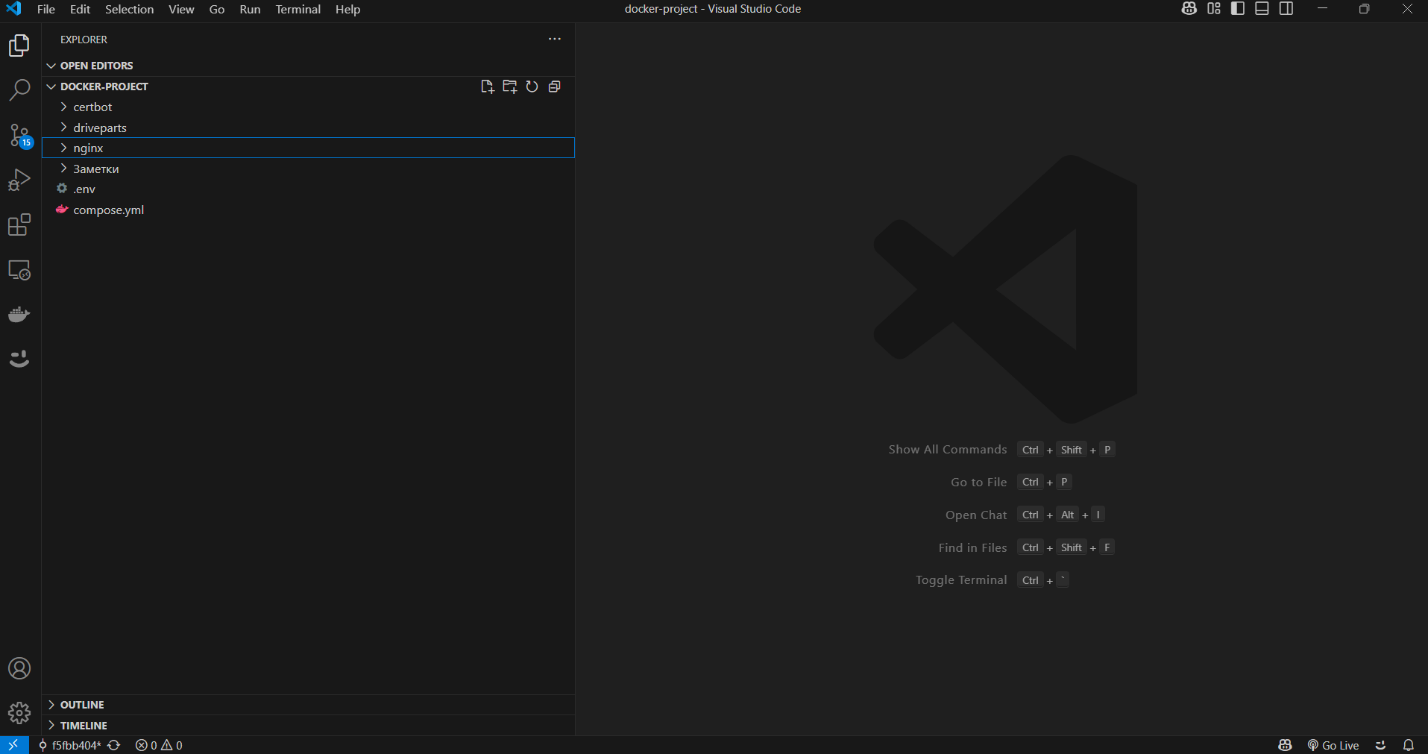


С помощью команды cd переходим в нашу рабочую директорию. Дальше запускаем скрипт **python -m venv myvenv**, который создаёт виртуальное окружение **myvenv**.

Мы завершили установку и настройку нашей рабочей директории. В следующем модули мы будем устанавливать модули веб фреймворка Django

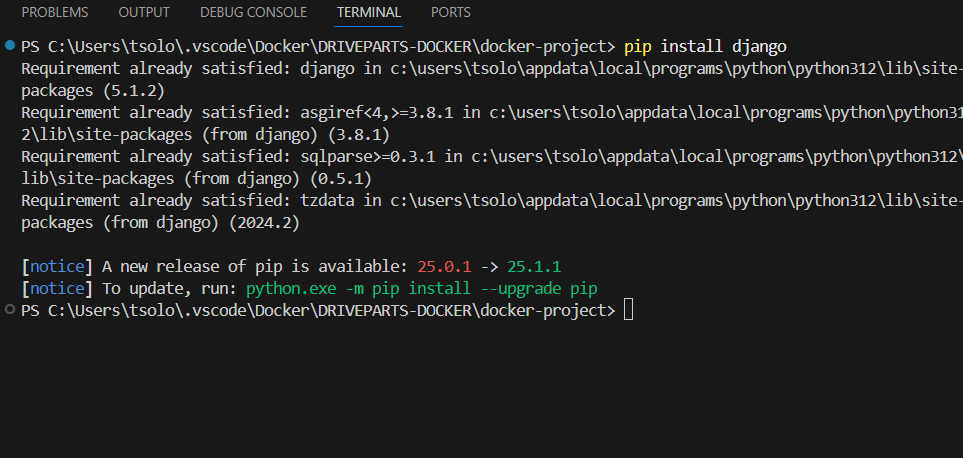
2.2 Установка веб фреймворка Django и создание проекта

Переходим в IDE и заходим в рабочую директорию.



Далее подключаем терминал и вбиваем команду **pip install Django**

**pip** — система управления пакетами, которая используется для установки и управления программными пакетами, написанными на Python.



Все нужные модули установлены и мы можем переходить к созданию каталога нашего проекта.

Для создания рабочего каталога нам потребуется команда **python manage.py startproject driveparts.**  Также вбиваем её в терминале.

**manage.py —** это автоматически создаваемый скрипт в каждом Django-проекте, который служит для взаимодействия с проектом через командную строку. Он упрощает выполнение административных задач, таких как запуск сервера, создание миграций, управление приложениями и многое другое.

Команда **startproject** отвечает за создание нового проекта, а дальше прописывается его название.

2.3. Структура проекта на Django.

Рассмотрим стандартную структуру веб приложения на Django.

**Django** использует **модульную структуру**, где проект состоит из одного или нескольких приложений. Каждое приложение отвечает за конкретную функциональность (например, аутентификацию, блог, API).

Стандартная структура проекта:

drive-parts/ # Корневая директория

├── manage.py # Основной скрипт управления

├── drive-parts/ # Пакет проекта (настройки)

│ ├── \_\_init\_\_.py

│ ├── settings.py # Все настройки проекта

│ ├── urls.py # Главные URL-маршруты

│ ├── asgi.py # Для ASGI-серверов (Django 3.0+)

│ └── wsgi.py # Для развертывания (WSGI)

├── goods/ # Приложение "Товары"

│ ├── models.py # Модель Goods, Category

│ ├── views.py # Логика магазина

│ ├── urls.py # URL-маршруты приложения

│ └── templates/blog/ # Шаблоны блога

├── users/ # Приложение "Пользователи"

│ ├── models.py # Кастомная модель User

│ ├── views.py # Регистрация, вход

│ └── ...

├── static/ # Общие статические файлы

│ └── css/main.css

└── templates/ # Глобальные шаблоны (базовый layout)

└── base.html

└── requirements.txt # Зависимости (создается вручную)

Разберём каждый пункт по отдельности.

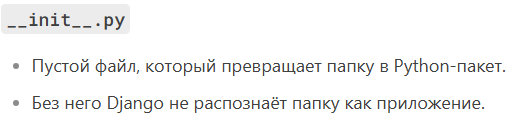
**Django-приложение** — это модуль, выполняющий конкретную задачу (например, блог, аутентификацию, API). Главная особенность приложений в Django это их изолированность. Приложение можно переносить между проектами, если оно не зависит от специфичных настроек.

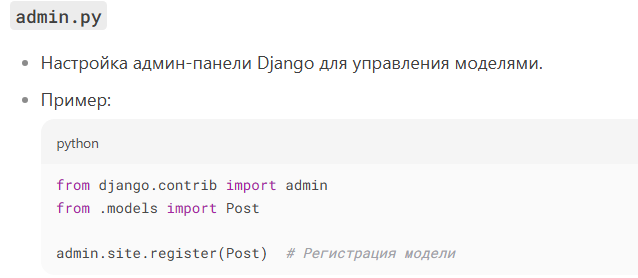
Стандартная структура Django-приложения\*\*

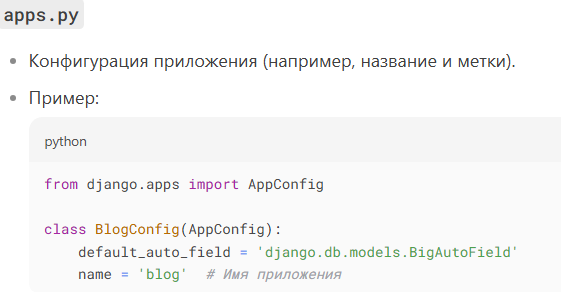
Django-приложение — это модуль, выполняющий конкретную задачу (например, блог, аутентификацию, API).

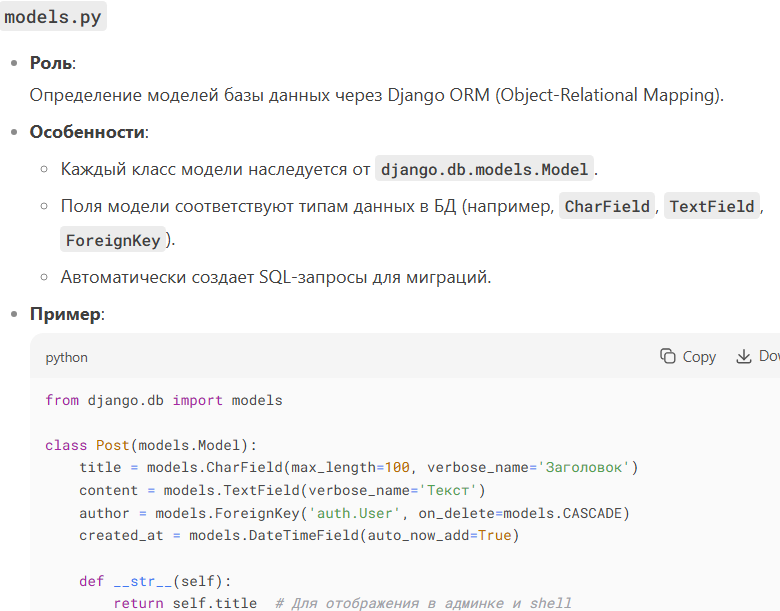
Каждый модуль в Django отвечает за свой фунционал:

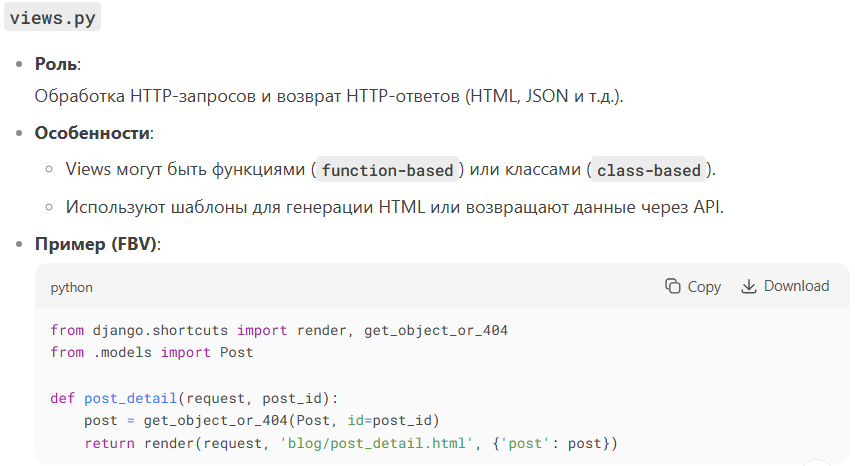
1. Автоматически создаваемые файлы (при `startapp`)\*\*

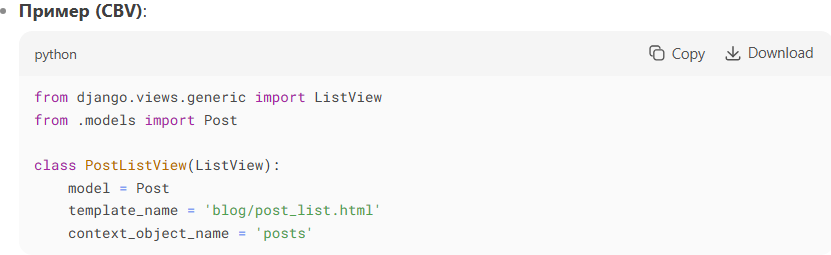


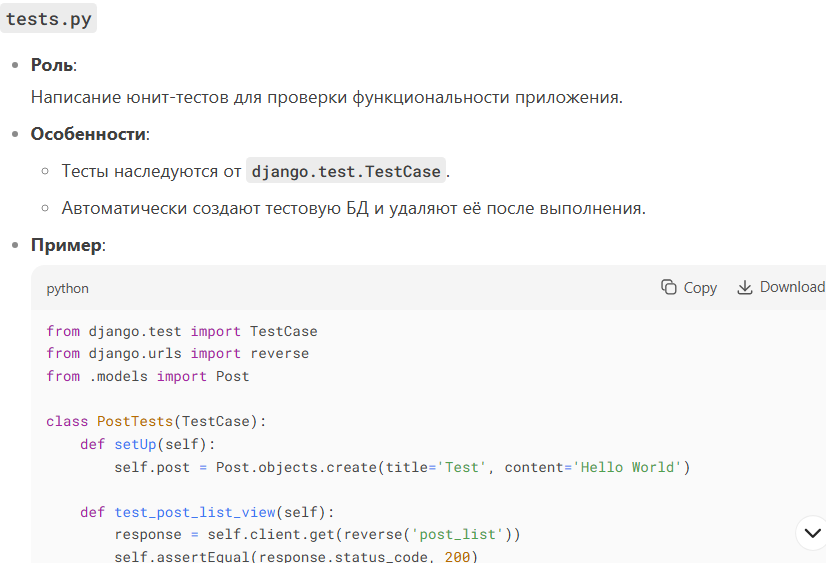


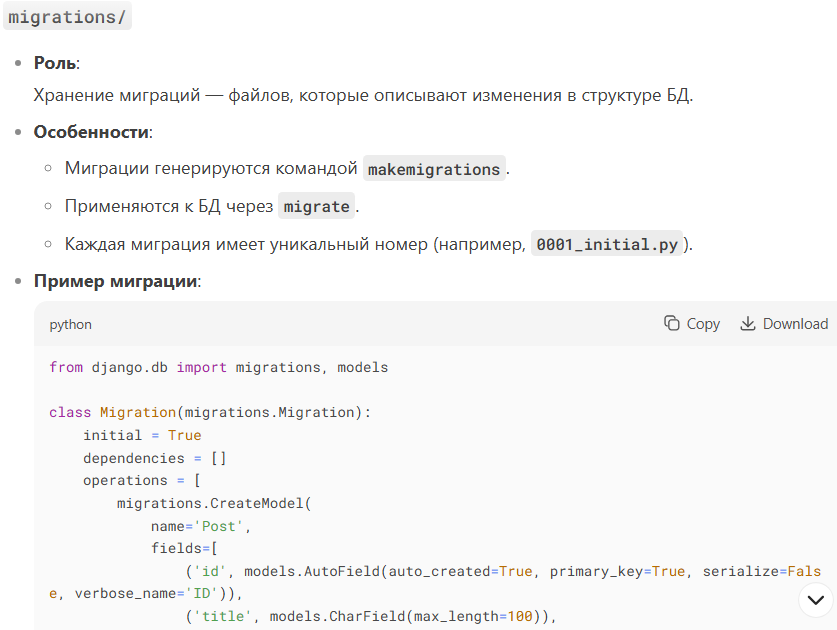




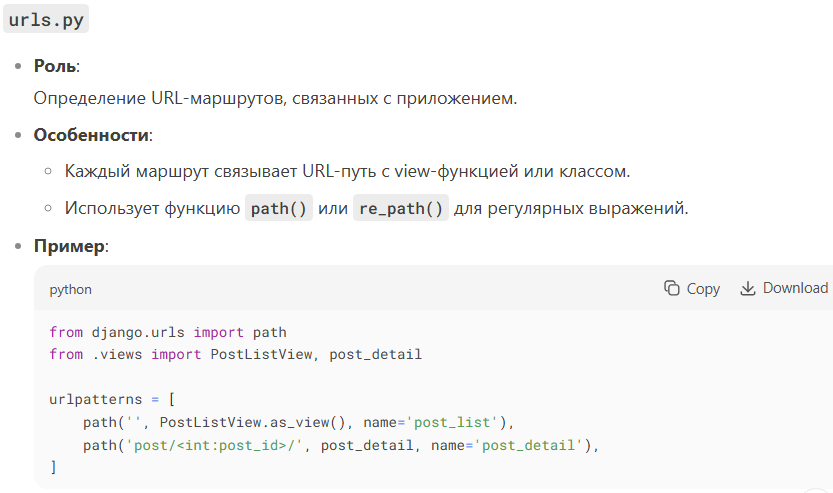


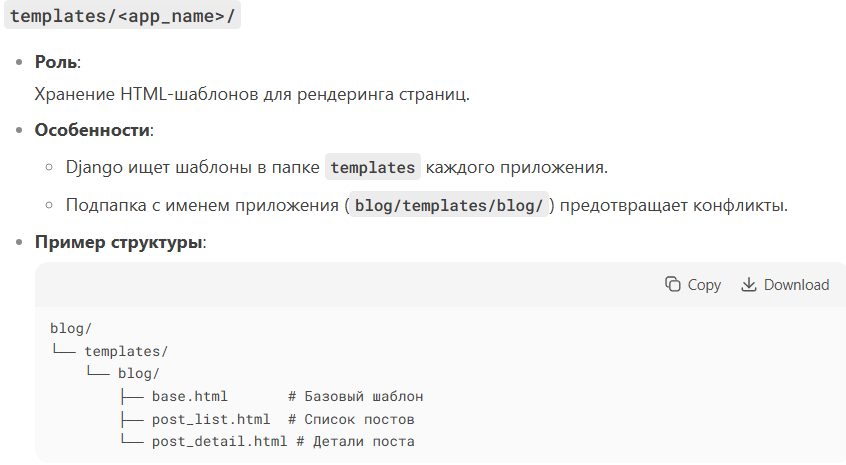


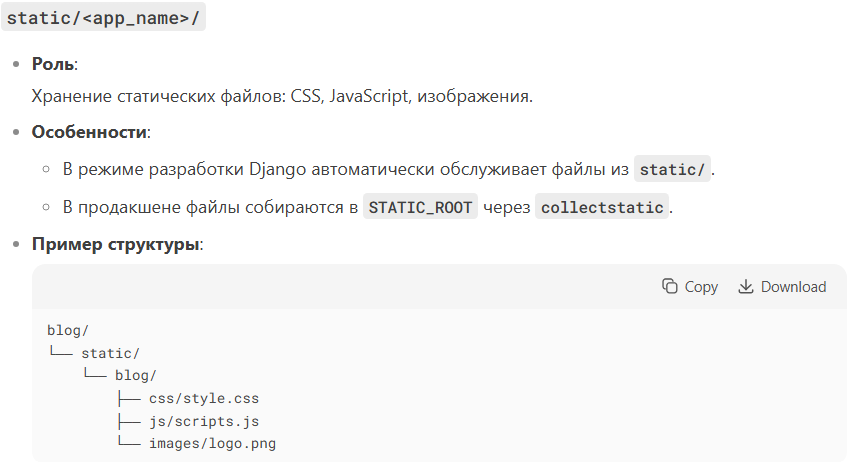


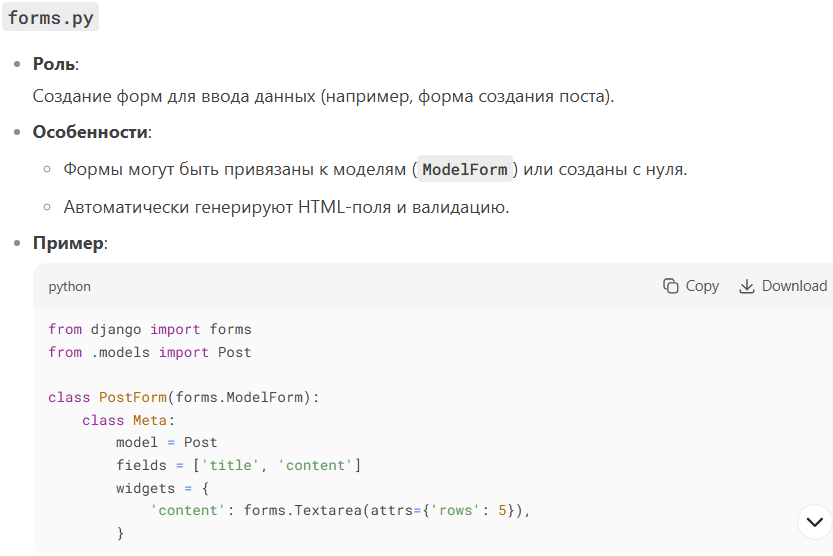


2. Файлы, создаваемые вручную







2.4 Архитектура Django и создание приложения для интернет-магазина автозапчастей

## Процесс обработки запроса в Django

Когда пользователь делает запрос к вашему интернет-магазину автозапчастей, Django обрабатывает его по следующей схеме:

1. \*\*Маршрутизация запроса\*\*:

- Запрос поступает на URL, например:

- Главная страница: `http://127.0.0.1:8000/`

- Категория товаров: `http://127.0.0.1:8000/category/engine/`

- Конкретная запчасть: `http://127.0.0.1:8000/parts/bosch-spark-plug-1234/`

2. \*\*Поиск совпадения в URL-шаблонах\*\*:

- Django проверяет шаблоны URL в порядке их объявления

- Примеры шаблонов:

```python

path('', views.home), # главная страница

path('category/<slug:category\_slug>/', views.category),

path('parts/<slug:part\_slug>/', views.part\_detail),

```

3. \*\*Обработка не найденных страниц\*\*:

- Если совпадение не найдено - возвращается 404 ошибка

## Паттерн MTV в Django

Django использует паттерн Model-Template-View (MTV) для разделения ответственности:

1. \*\*Model (Модель)\*\*:

- Определяет структуру данных (например, товары, категории)

- Работает с базой данных через ORM

2. \*\*Template (Шаблон)\*\*:

- HTML-файлы с динамическими блоками

- Отвечают за отображение данных

3. \*\*View (Представление)\*\*:

- Обрабатывают запросы

- Получают данные из моделей

- Передают данные в шаблоны

- Формируют ответ

Пример для интернет-магазина автозапчастей:

```

Запрос: /parts/bosch-spark-plug-1234/

↓

View получает данные о свече зажигания Bosch из Model

↓

View передает данные в Template

↓

Template генерирует HTML страницу товара

```

## Создание приложения для интернет-магазина

Для нашего магазина автозапчастей создадим основное приложение `goods`:

```bash

python manage.py startapp goods

```

Структура созданного приложения:

```

goods/

├── \_\_init\_\_.py

├── admin.py # Настройки админки

├── apps.py # Конфигурация приложения

├── models.py # Модели данных (товары, категории)

├── tests.py # Тесты

└── views.py # Обработчики запросов

migrations/ # Файлы миграций БД

```

## Регистрация приложения

Добавляем приложение в `settings.py`:

```python

INSTALLED\_APPS = [

...

'goods.apps.GoodsConfig', # Регистрируем наше приложение

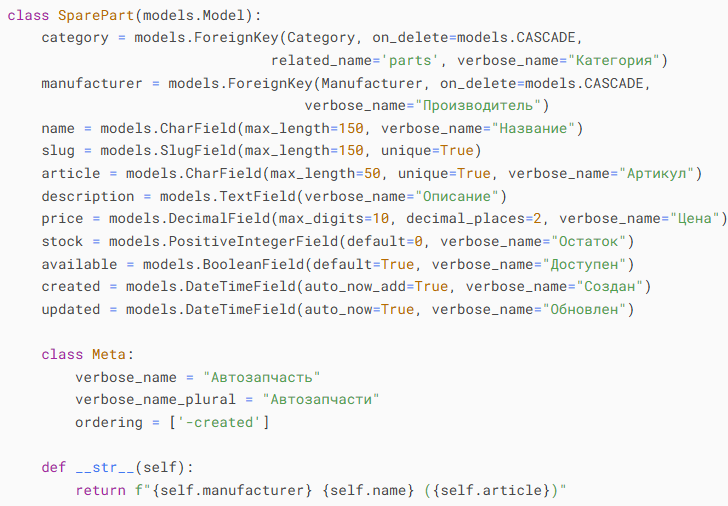
]

```

## Настройка моделей для магазина автозапчастей

В `models.py` создаем базовые модели:





## Настройка админ-панели

В `admin.py` регистрируем модели для управления через админку:

```python

from django.contrib import admin

from .models import Category, Manufacturer, SparePart

@admin.register(Category)

class CategoryAdmin(admin.ModelAdmin):

list\_display = ('name', 'slug')

prepopulated\_fields = {'slug': ('name',)}

@admin.register(Manufacturer)

class ManufacturerAdmin(admin.ModelAdmin):

list\_display = ('name', 'country')

list\_filter = ('country',)

@admin.register(SparePart)

class SparePartAdmin(admin.ModelAdmin):

list\_display = ('name', 'manufacturer', 'category', 'price', 'stock', 'available')

list\_filter = ('available', 'category', 'manufacturer')

list\_editable = ('price', 'stock', 'available')

prepopulated\_fields = {'slug': ('name', 'article')}

search\_fields = ('name', 'article', 'description')

```

## Создание представлений

В `views.py` добавляем базовые обработчики:

```python

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

from .models import Category, SparePart

def product\_list(request, category\_slug=None):

category = None

categories = Category.objects.all()

products = SparePart.objects.filter(available=True)

if category\_slug:

category = get\_object\_or\_404(Category, slug=category\_slug)

products = products.filter(category=category)

return render(request,

'goods/product/list.html',

{'category': category,

'categories': categories,

'products': products})

def product\_detail(request, id, slug):

product = get\_object\_or\_404(SparePart,

id=id,

slug=slug,

available=True)

return render(request,

'goods/product/detail.html',

{'product': product})

```

## Настройка URL-маршрутов

Создаем файл `goods/urls.py`:

```python

from django.urls import path

from . import views

app\_name = 'goods'

urlpatterns = [

path('', views.product\_list, name='product\_list'),

path('<slug:category\_slug>/', views.product\_list, name='product\_list\_by\_category'),

path('<int:id>/<slug:slug>/', views.product\_detail, name='product\_detail'),

]

```

Подключаем эти маршруты в основном `urls.py` проекта:

```python

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

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

]

```

## Следующие шаги

1. Создать миграции и применить их:

```bash

python manage.py makemigrations

python manage.py migrate

```

2. Создать суперпользователя для доступа к админке:

```bash

python manage.py createsuperuser

```

3. Разработать HTML-шаблоны в папке `goods/templates/goods/`

4. Настроить статические файлы (CSS, JS, изображения)

5. Реализовать корзину покупок (можно как отдельное приложение `cart`)