ВВЕДЕНИЕ

Современные интернет-магазины играют ключевую роль в глобальной экономике. С ростом числа пользователей интернета и развитием технологий, потребность в надежных и высококачественных платформах для электронной коммерции значительно возросла. Интернет-магазины должны обеспечивать удобный интерфейс, высокую безопасность данных и отличную производительность, чтобы удовлетворить потребности как покупателей, так и продавцов.

Django, популярный веб-фреймворк на языке Python, зарекомендовал себя как мощный инструмент для создания веб-приложений, включая интернет-магазины. Благодаря широкому набору встроенных функций, модульной архитектуре и возможности быстрой разработки, Django является идеальным выбором для создания сложных веб-приложений. Фреймворк предлагает разработчикам множество инструментов, таких как ORM (Object-Relational Mapping), административная панель, система аутентификации, кэширование и многое другое, что облегчает процесс разработки и позволяет сосредоточиться на бизнес-логике приложения.

Целью данной выпускной квалификационной работы является разработка серверной части интернет-магазина на основе фреймворка Django. В рамках работы будут рассмотрены теоретические аспекты разработки веб-приложений, проведен анализ существующих решений и выбран наиболее подходящий стек технологий. В практической части работы будет спроектирована и реализована модель продукта, которая включает в себя базу данных, обработчики запросов, а также интеграцию с платежными системами и другими внешними сервисами.

Актуальность данной темы обусловлена возрастающим спросом на качественные решения для электронной коммерции. Разработка интернет-магазинов с использованием современных технологий, таких как Django,

позволяет создавать гибкие, масштабируемые и безопасные платформы, способные удовлетворить запросы самых требовательных пользователей.

Задачи:

1. Анализ теоретических аспектов разработки интернет-магазинов
2. Отбор и обоснование выбора инструментов для разработки
3. Проектирование модели данных интернет-магазина
4. Разработка серверной части интернет-магазина

Таким образом, данная работа не только демонстрирует практическое применение фреймворка Django, но и предоставляет подробное руководство по разработке серверной части интернет-магазина, что может быть полезно как для начинающих разработчиков, так и для опытных профессионалов, стремящихся расширить свои знания и навыки в области веб-разработки.

Объект: процесс разработки серверной части веб-приложений, в частности интернет-магазинов.

Предмет: использование фреймворка Django для создания серверной части интернет-магазина, включая выбор инструментов, проектирование архитектуры приложения, реализация функциональных компонентов.

1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ И АНАЛИЗ ИНСТРУМЕНТОВ ДЛЯ РАЗРАБОТКИ ИНТЕРНЕТ-МАГАЗИНА

1.1 О бэкенд разработке

В первую очередь, перед началом разработки бэкенда (серверной части) для интернет-магазина, следует рассмотреть основные понятия, которые характеризуют бэкенд разработку, чтобы грамотно выстроить этот процесс.

Бэкенд (от англ. Backend) – это такая область веб-разработки, которая работает на сервере и отвечает за управление данными и выполнение логики приложения. В отличие от клиентской части (фронтенда) которая отображается в браузере пользователя, бэкенд скрыт и невидим для конечного пользователя. Бэкенд-программисты пишут код на языках программирования, таких как Python, Java, Ruby, PHP или JavaScript (Node.js).

Бэкенд состоит из серверных приложений, баз данных и API, которые совместно обеспечивают работу веб-приложения. Он управляет процессами, такими как регистрация и авторизация пользователей, а также обработка данных и бизнес-логика. Например, при создании нового аккаунта бэкенд проверяет введённые данные, сохраняет их в базу данных и отправляет подтверждение пользователю. Он также обрабатывает ресурсоёмкие вычисления, которые слишком сложны для выполнения на стороне клиента. Надежная работа бэкенда важна для стабильности и производительности всего приложения. Бэкенд также включает меры безопасности, чтобы защитить данные пользователей и предотвратить несанкционированный доступ.

Языки программирования – это инструменты, посредством которых разработчики создают код для своих приложений. Существует большое разнообразие языков, подходящих для бэкенд-разработки, но наиболее популярными и востребованными в настоящее время являются Python, Java, JavaScript (Node.js), PHP, Ruby и C#. Каждый из этих языков обладает своими уникальными преимуществами и недостатками, а также предназначен для решения определенных задач и применения в различных областях.

Python, например, известен своей простотой и читаемостью, что делает его идеальным выбором для быстрого прототипирования и разработки сложных научных и аналитических приложений. Java, с другой стороны, славится своей производительностью и масштабируемостью, что делает его популярным в крупномасштабных корпоративных проектах. JavaScript, в особенности Node.js, позволяет использовать один и тот же язык как на стороне клиента, так и на стороне сервера, что упрощает процесс разработки и улучшает производительность. PHP часто используется для создания динамических веб-сайтов и приложений благодаря своей простоте и широкому распространению в веб-разработке. Ruby известен своим элегантным синтаксисом и удобством для быстрого создания веб-приложений, особенно с использованием фреймворка Ruby on Rails. C# отлично интегрируется с платформой .NET и часто применяется в разработке корпоративных приложений и игр.

Фреймворки (от англ. Framework – «каркас, структура») – набор инструментов и библиотек для конкретного языка программирования. Они упрощают и ускоряют разработку веб-приложений, предлагая готовые решения для популярных задач и проблем. Использование фреймворков позволяет разработчикам сосредоточиться на уникальных аспектах проекта, не тратя время на рутинные задачи, такие как маршрутизация, аутентификация или работа с базами данных.

Фреймворки часто включают в себя структурированные шаблоны и стандарты, которые помогают поддерживать код в чистоте и порядке. Они также могут предоставлять множество встроенных функций, таких как системы управления пользователями, формирование шаблонов, валидация данных и защита от распространенных уязвимостей. Например, Django, один из популярных фреймворков для Python, предлагает встроенные средства для обеспечения безопасности, включая защиту от XSS и CSRF атак, а также мощную ORM-систему для работы с базами данных.

Использование фреймворков способствует улучшению масштабируемости проектов и упрощает командную работу, поскольку большинство разработчиков знакомы с популярными фреймворками и могут быстро включиться в работу над проектом. Это делает фреймворки незаменимыми в современном процессе разработки веб-приложений, существенно повышая их эффективность и качество. Более того, фреймворки часто имеют обширные сообщества, предлагающие поддержку и множество дополнительных пакетов, расширяющих возможности разработчиков. Таким образом, фреймворки играют ключевую роль в создании надёжных, масштабируемых и безопасных веб-приложений, обеспечивая быструю и эффективную разработку.

Таким образом, нам удалось рассмотреть понятие бэкенд разработки. Мы выяснили, что для построения этого процесса обязательно следует выбрать язык программирования и фреймворк. Для интернет-магазина изначально был выбран фреймворк – Django, так как имеет ряд преимуществ. Следовательно язык программирования для данного фреймворка будем использовать Python. Также в качестве СУБД будет использоваться – PostgreSQL. Далее подробнее рассказывается об этих технологиях и их преимуществах для разработки.

1.2 Обзор технологий и фреймворков для разработки

Разработка интернет-магазина требует использование различных технологий и инструментов, которые должны обеспечивать надежность, стабильность работы, высокую производительность в случае большого количества пользователей и возможность масштабируемости. Рассмотрим основные технологии и фреймворки, которые применяются в разработке интернет-магазинов, в особенности рассмотрим Django и сравним его с другими фреймворками.

1.2.1 Django и его возможности

Django — это фреймворк, то есть набор готовых инструментов и функций. С его помощью можно быстрее и проще реализовывать на Python сайты и приложения, которые работают в браузере.

Конечно, можно создать веб-сервис на Python и без Django, но тогда очень многое придется писать и настраивать с нуля самостоятельно. Если использовать программирование на Django, можно сосредоточиться только на уникальных функциях веб-сервиса.[6]

Фундаментальная структура данного фреймворка включает в себя следующие ключевые элементы:

- Маршрутизаторы URL, которые направляют HTTP-запросы от браузера или других веб-клиентов к соответствующим представлениям;
- представление, которое занимается обработкой запроса, обращается к модели и сообщает ей, какие именно данные из базы данных нужно задействовать, чтобы удовлетворить запрос;
- модель (менеджер базы данных, ORM), «вытаскивающую» нужную информацию из базы данных и передающую ее представлению;
- HTML-шаблоны, используемые представлениями для отображения данных, полученных из модели, пользователю;

Эти компоненты тесно взаимодействуют между собой, обеспечивая гибкость и эффективность в разработке веб-приложений на данном фреймворке.

Рассмотрим ключевые возможности Django:

- **MVC-архитектура.** Django использует архитектурных шаблон Model-View-Controller (MVC), это обеспечивает четкое разделение логики, данных и представления. Это упрощает поддержку и расширение кода.
- **Административная панель.** Django дает встроенную административную панель которая генерируется на основе моделей данных и предоставляет удобный интерфейс для непосредственно работы с данными (наполнение, изменение, обновление данных), что особенно полезно для администраторов интернет-магазинов.
- **ORM (Object-Relational Mapping).** Django обладает мощной системой ORM, благодаря которой разработчики могут взаимодействовать с базами

данных с помощью объектно-ориентированных интерфейсов. Это значительно облегчает процесс создания и управления базами данных, устраняя необходимость в написании сложных SQL-запросов.

- **Безопасность.** Django предлагает встроенные механизмы защиты от распространенных веб-угроз, таких как XSS, CSRF и SQL-инъекции. Это позволяет разработчикам создавать безопасные приложения без необходимости самостоятельно внедрять защитные меры.
- **Масштабируемость и производительность.** Django отлично подходит для разработки масштабируемых приложений благодаря своей модульной архитектуре и поддержке различных баз данных и серверных технологий (например, PostgreSQL, MySQL, SQLite)
- **Аутентификация и авторизация.** Django предоставляет полноценные механизмы аутентификации и авторизации, которые не только позволяют управлять пользователями, но и контролировать групповые права и разрешения. Это делает процесс создания системы регистрации, аутентификации и управления доступом к разным компонентам приложения более интуитивным и эффективным. Таким образом, разработчики могут сосредоточиться на основной функциональности приложения, не тратя время на рутинные задачи по управлению пользователями и их правами.

1.2.2 Встроенные инструменты и библиотеки Django

Django — это мощный веб-фреймворк, который включает в себя обширный набор встроенных инструментов и библиотек. Эти ресурсы значительно облегчают и ускоряют процесс создания веб-приложений. В этом разделе мы подробно изучим основные встроенные инструменты и библиотеки Django, рассмотрим их возможности и приведём примеры использования.

Django ORM (Object-Relational Mapping)

Для создания базы данных (и всех взаимодействий с ней) Django использует ORM (объектно-реляционное представление). ORM – это своеобразная прослойка, которая позволяет работать с базой данных, используя классы и методы вместо написания сложных SQL-запросов [8]. Пример использования изображен ниже на Рисунке 1.

Возможности:

1. **Моделирование данных:** Определение таблиц базы данных с помощью классов Python. Использование встроенных типов данных, таких как CharField, TextField, IntegerField, DateTimeField, и других для создания полей таблиц. Возможность добавления пользовательских полей и валидации данных на уровне модели.
1. **Запросы к базе данных:** Выполнение CRUD-операций (создание, чтение, обновление, удаление) через методы модели. Поддержка сложных запросов, таких как фильтрация (filter), сортировка (order_by), агрегация (aggregate, annotate) и объединение (select_related, prefetch_related).
2. **Миграции:** Автоматическое создание и применение миграций для изменения структуры базы данных. Команды для создания новых миграций (makemigrations) и их применения (migrate). Возможность отката миграций и управления версией базы данных.
3. **Связи между моделями:** Поддержка различных типов отношений: один ко многим (ForeignKey), многие ко многим (ManyToManyField), один к одному (OneToOneField). Автоматическое создание связанных таблиц и управление связями через ORM. Возможность каскадного удаления и обновления связанных объектов.

```python
from django.db import models

class Product(models.Model):
    name = models.CharField(max_length=255)
    description = models.TextField()
    price = models.DecimalField(max_digits=10, decimal_places=2)
    stock = models.IntegerField()

# Запросы
products = Product.objects.all()
product = Product.objects.get(id=1)
```

Рисунок 1. Пример django ORM

Django Admin

Одной из самых мощных частей Django является автоматический интерфейс администратора. Он считывает метаданные из ваших моделей, обеспечивая быстрый, ориентированный на модель интерфейс, в котором доверенные пользователи могут управлять контентом вашего сайта [9]. Пример использования изображен ниже на Рисунке 2.

Возможности:

1. **Управление данными:** CRUD-операции (создание, чтение, обновление, удаление) для всех моделей. Поддержка сложных операций, таких как массовое редактирование и удаление записей.
2. **Фильтрация и поиск:** Встроенные фильтры для быстрого поиска и фильтрации данных по различным критериям. Настраиваемые поля поиска для улучшения пользовательского опыта.
3. **Настройка интерфейса:** Возможность настройки отображения полей, форм и списков через классы администратора (ModelAdmin).
4. **Аутентификация и авторизация:** Встроенная поддержка управления пользователями и группами, настройка прав доступа на уровне моделей и объектов. Возможность ограничения доступа к административной панели на основе ролей и разрешений.

```python
from django.contrib import admin
from .models import Product

class ProductAdmin(admin.ModelAdmin):
    list_display = ('name', 'price', 'stock')
    search_fields = ('name',)

admin.site.register(Product, ProductAdmin)
```

Рисунок 2. Пример django Admin

Система маршрутизации (URL routing)

URL-адресация в Django является важной частью разработки веб-приложений, поскольку она определяет, как приложение отвечает на запросы пользователей. В Django URL-адресация осуществляется с помощью механизма маршрутизации, который связывает конкретные URL-адреса с функциями представления (views), обрабатывающими эти запросы [10]. Пример использования изображен ниже на Рисунке 3.

Возможности:

1. **Определение маршрутов:** Настройка URL-шаблонов в файле urls.py для каждого приложения.
2. **Динамические параметры:** Поддержка передачи параметров через URL, что позволяет создавать динамические маршруты. Автоматическая обработка и валидация параметров, таких как идентификаторы объектов, строки и числа.
3. **Маршрутизация на основе имен:** Возможность именования маршрутов для упрощения их использования в шаблонах и коде. Обратная маршрутизация (reverse URL resolution) для генерации URL на основе имен маршрутов, что повышает гибкость и надежность приложения.

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

urlpatterns = [
    path('products/', views.product_list, name='product_list'),
    path('products/<int:id>/', views.product_detail, name='product_detail'),
]
```

Рисунок 3. Пример системы маршрутизации

Система шаблонов (Template system)

Будучи веб-фреймворком, Django нуждается в удобном способе динамической генерации HTML. Наиболее распространенный подход основан на шаблонах. Шаблон содержит статические части желаемого вывода HTML, а также некоторый специальный синтаксис, описывающий, как будет вставлено динамическое содержимое [11]. Пример использования изображен ниже на Рисунке 4.

Возможности:

1. **Шаблоны и наследование:** Поддержка базовых шаблонов и их наследования для создания единообразного внешнего вида приложения. Возможность переопределения и расширения блоков контента в дочерних шаблонах.
2. **Теги и фильтры:** Встроенные теги для выполнения различных операций, таких как циклы (for), условные конструкции (if), включение других шаблонов (include). Встроенные фильтры для форматирования данных, таких как дата и время, текстовые преобразования и математические операции. Поддержка пользовательских тегов и фильтров для расширения возможностей шаблонов.
3. **Контекст:** Передача данных из view-функций в шаблоны через контекст, что позволяет динамически отображать данные на страницах. Использование контекстных процессоров для

автоматической передачи данных в шаблоны, таких как информация о текущем пользователе, настройки сайта и т.д.

```
<!-- base.html -->
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}My Shop{% endblock %}</title>
</head>
<body>
    {% block content %}{% endblock %}
</body>
</html>

<!-- product_list.html -->
{% extends 'base.html' %}

{% block title %}Product List{% endblock %}

{% block content %}
<h1>Product List</h1>
<ul>
    {% for product in products %}
    <li>{{ product.name }} - ${{ product.price }}</li>
    {% endfor %}
</ul>
{% endblock %}
```

Рисунок 4. Пример системы шаблонов

Django Form

Django Form представляет собой мощный инструмент для создания и обработки веб-форм. Формы позволяют валидировать данные, генерировать HTML-код и управлять взаимодействием с пользователем. Пример использования изображен ниже на Рисунке 5.

Возможности:

1. **Определение форм:** Создание форм через классы с полями для ввода данных, такими как текстовые поля, выпадающие списки, чекбоксы и другие. Валидация данных на уровне формы с

использованием встроенных валидаторов и пользовательских методов.

2. **Валидация данных:** Поддержка встроенных валидаторов для проверки корректности данных, таких как проверка длины, формата, диапазона значений и уникальности. Возможность создания пользовательских валидаторов для сложных правил валидации.
3. **Связь с моделями:** Создание форм на основе моделей (ModelForm) для автоматического связывания полей формы с полями модели.

```python
from django import forms
from .models import Product

class ProductForm(forms.ModelForm):
    class Meta:
        model = Product
        fields = ['name', 'description', 'price', 'stock']

# В view-функции
def add_product(request):
    if request.method == 'POST':
        form = ProductForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('product_list')
    else:
        form = ProductForm()
    return render(request, 'add_product.html', {'form': form})
```

Рисунок 5. Пример представления Form

1.3 Сравнение Django с другими фреймворками

Для разработки интернет-магазинов также широко используются другие фреймворки, такие как Ruby on Rails, Laravel и Express.js. Рассмотрим их особенности и сравним с Django. Результат сравнения продемонстрирован в таблице 1.1.

Таблица 1.1 – Сравнительный анализ фреймворков

Фреймворк	**Язык**	**Архитектура**	**Основные преимущества**	**Основные недостатки**
Django	Python	MVC	Внушительная ORM, встроенная административная панель, безопасность	Медленная адаптация к новым технологиям, большой объем памяти для фунционировния
Ruby on Rails	Ruby	MVC	Высокая скорость разработки, интуитивно понятный синтаксис	Меньшая производительность по сравнению с Django, ограниченная поддержка асинхронного программирования
Express.js	JavaScript (Node.js)	Middleware	Высокая производительность, легковесность, возможность использования одного языка (JavaScript) на клиентской и серверной части	Менее структурированный подход по сравнению с MVC-фреймворками, необходимость настройки многих компонентов вручную
Laravel	PHP	MVC	Удобные инструменты для работы с базами данных, элегантный синтаксис, мощные возможности для создания API	Медленная производительность по сравнению с Django и Rails, сложность масштабирования больших приложений

Выбор подходящего фреймворка для разработки интернет-магазина — это сложное решение, которое зависит от множества факторов. Среди них — требования проекта, предпочтения разработчиков и специфика задачи. Django привлекает внимание своей мощной функциональностью, высоким уровнем безопасности и интуитивно понятным процессом разработки, что делает его превосходным выбором для создания даже самых сложных и масштабируемых интернет-магазинов. Однако стоит помнить, что каждый фреймворк имеет свои особенности, и для достижения оптимальных результатов важно тщательно анализировать их, выбирая тот, который наилучшим образом соответствует уникальным потребностям и целям вашего проекта.

1.4 Выбор СУБД

Выбор системы управления базами данных (СУБД) является критически важным этапом при разработке серверной части интернет-магазина. Правильный выбор СУБД влияет на производительность, масштабируемость, надежность и безопасность системы. В условиях современного рынка, где интернет-магазины должны обрабатывать большие объемы данных, обеспечивать высокую доступность и защищать конфиденциальную информацию пользователей, выбор подходящей СУБД становится особенно актуальным.

SQLite — это изумительная библиотека, встраиваемая в приложение, которое её использует. Будучи файловой БД, она предоставляет отличный набор инструментов для более простой (в сравнении с серверными БД) обработки любых видов данных. Когда приложение использует SQLite, их связь производится с помощью функциональных и прямых вызовов файлов, содержащих данные (например, баз данных SQLite), а не какого-то интерфейса, что повышает скорость и производительность операций [12]. Основные особенности SQLite включают:

- **Простота и легкость:** SQLite не требует установки и настройки сервера, что делает ее идеальной для разработки и тестирования.

- **Небольшой размер:** База данных SQLite хранится в одном файле, что упрощает управление данными и их переносимость.
- **Производительность:** Несмотря на свою легкость, SQLite обеспечивает хорошую производительность для небольших и средних объемов данных.
- **Отсутствие необходимости в сервере:** SQLite работает без сервера, что снижает сложность развертывания и обслуживания.

MySQL — это самая популярная из всех крупных серверных БД. Разобраться в ней очень просто, да и в сети о ней можно найти большое количество информации. Хотя MySQL и не пытается полностью реализовать SQL-стандарты, она предлагает широкий функционал [12]. Основные характеристики MySQL включают:

- **Высокая производительность:** MySQL оптимизирована для быстрого выполнения запросов, что делает ее подходящей для высоконагруженных приложений.
- **Простота использования:** MySQL имеет интуитивно понятный синтаксис и легко настраивается, что делает ее доступной даже для начинающих разработчиков.
- **Масштабируемость:** Поддержка репликации данных и кластеризации позволяет легко масштабировать базы данных горизонтально.
- **Сообщество и поддержка:** Широкое сообщество пользователей и обширная документация облегчают решение возникающих проблем и внедрение новых функций.

PostgreSQL — это самая продвинутая РСУБД, ориентирующаяся в первую очередь на полное соответствие стандартам и расширяемость. PostgreSQL, или Postgres, пытается полностью соответствовать SQL-стандартам ANSI/ISO. Будучи основанным на мощной технологии Postgres отлично справляется с одновременной обработкой нескольких заданий. Поддержка конкурентности реализована с использованием MVCC (Multiversion Concurrency Control), что

также обеспечивает совместимость с ACID. Хотя эта РСУБД не так популярна, как MySQL, существует много сторонних инструментов и библиотек для облегчения работы с PostgreSQL [12]. Основные характеристики PostgreSQL включают:

- **Поддержка расширений:** PostgreSQL поддерживает множество расширений, таких как PostGIS для работы с географическими данными.
- **Соответствие стандартам SQL:** PostgreSQL полностью соответствует стандартам SQL, что обеспечивает высокую совместимость и переносимость кода.
- **Масштабируемость и производительность:** Система поддерживает сложные транзакции, параллельное выполнение запросов и репликацию данных, что обеспечивает высокую производительность и масштабируемость.
- **Безопасность:** PostgreSQL предоставляет развитую систему управления доступом и поддерживает SSL для защиты данных в процессе передачи.

Для выбора наиболее подходящей СУБД для разработки интернет-магазина необходимо учитывать такие факторы, как производительность, масштабируемость, поддержка транзакций и расширяемость. Ниже в таблице 1.2 приведено сравнение основных характеристик PostgreSQL, MySQL и SQLite:

Таблица 1.2 – Сравнительный анализ СУБД

Характеристика	**SQLite**	**MySQL**	**PostgreSQL**
Тип СУБД	Объектно-реляционная	Реляционная	Встроенная реляционная

Соответствие стандартам SQL	Высокое	Среднее	Среднее
Производительность	Высокая	Высокая	Средняя
Масштабируемость	Высокая	Высокая	Низкая
Расширяемость	Высокая (поддержка расширений)	Ограниченная	Отсутствует
Управление транзакциями	Высокая (ACID)	Высокая (ACID)	Высокая (ACID)
Управление доступом	Развитая система управления	Простая система	Ограниченная
Установка и настройка	Сложная	Простая	Очень простая
Сообщество и поддержка	Широкое	Широкое	Ограниченное

Для разработки серверной части интернет-магазина выбран PostgreSQL. Этот выбор обоснован несколькими ключевыми факторами:

1. **Надежность и соответствие стандартам SQL**: PostgreSQL полностью соответствует стандартам SQL, что обеспечивает высокую надежность и переносимость кода. Это особенно важно для интернет-магазинов, где корректность данных и совместимость играют критическую роль.
2. **Масштабируемость и производительность**: PostgreSQL поддерживает сложные транзакции и параллельное выполнение запросов, что позволяет эффективно обрабатывать большие объемы данных и обеспечивать высокую производительность системы.
3. **Расширяемость**: Возможность использования различных расширений, таких как PostGIS, позволяет значительно расширить функциональные возможности базы данных без необходимости перехода на другие решения.
4. **Безопасность**: PostgreSQL предоставляет развитую систему управления доступом и поддерживает SSL для защиты данных, что является важным фактором для обеспечения безопасности пользовательских данных в интернет-магазине.

Таким образом, PostgreSQL предоставляет оптимальное сочетание надежности, производительности, расширяемости и безопасности, что делает его наиболее подходящим выбором для реализации серверной части интернет-магазина на Django.

2. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ СЕРВЕРНОЙ ЧАСТИ ИНТЕРНЕТ-МАГАЗИНА

2.1 Моделирование данных и разработка моделей

2.1.1. Выбор архитектурного стиля

Для серверной части интернет-магазина на Django можно использовать различные архитектурные стили. Рассмотрим основные из них в таблице 2.1:

Название	Монолитная архитектура	Микросервисная архитектура	Сервис-ориентированная архитектура (SOA)
Описание	Вся логика приложения, включая пользовательский интерфейс, серверную часть и базу данных, разрабатывается и развертывается как единое целое.	Приложение делится на небольшие, автономные службы (микросервисы), каждая из которых отвечает за выполнение конкретной бизнес-функции.	Приложение состоит из множества взаимосвязанных сервисов, которые взаимодействуют через определённые протоколы и стандарты.
Преимущества	Простота разработки и развертывания, лёгкость отладки и тестирования.	Высокая масштабируемость, гибкость, возможность независимого развертывания и обновления отдельных сервисов.	Гибкость, возможность повторного использования сервисов, масштабируемость.
Недостатки	Ограниченная масштабируемость, сложность поддержания и внесения	Сложность координации и управления микросервисами, необходимость обеспечения	Сложность реализации и управления, необходимость в дополнительных средствах для

	изменений в крупные проекты.	надёжного взаимодействия между сервисами.	обеспечения взаимодействия и безопасности.
Применимость	Идеально подходит для небольших и средних проектов, где важны простота и скорость разработки.	Оптимально для крупных проектов с высокой нагрузкой и требованиями к масштабируемости.	Подходит для проектов, где важна интеграция с другими системами и повторное использование сервисов.

Для данного проекта наиболее подходящей является микросервисная архитектура. Она обеспечивает высокую масштабируемость и гибкость, что особенно важно для интернет-магазинов с большим количеством пользователей и продуктов. Микросервисы позволяют легко адаптироваться к изменениям и добавлять новые функции без нарушения работы всего приложения, что делает их отличным выбором для динамичных и развивающихся интернет-магазинов.

2.1.2. Определение основных компонентов системы

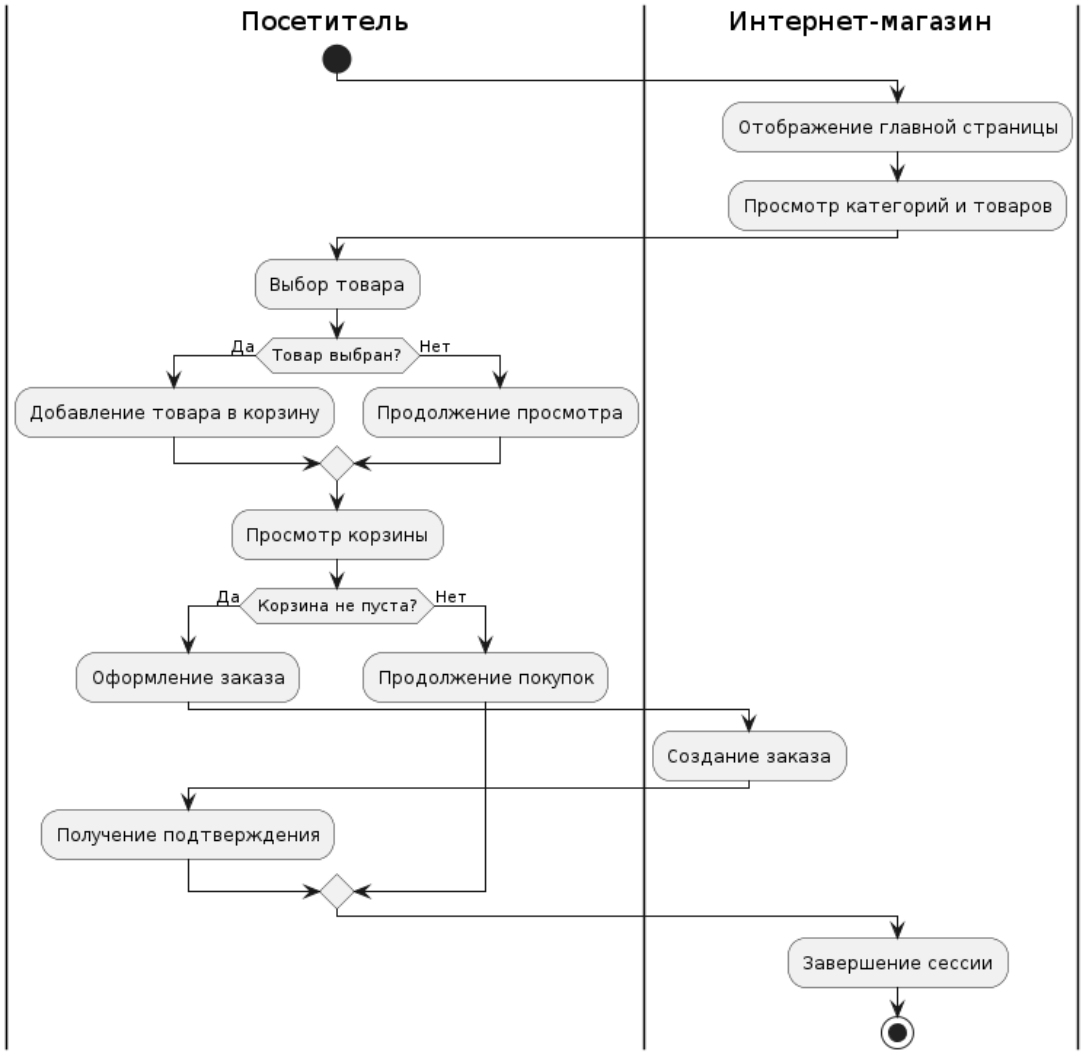
Проектирование архитектуры включает определение ключевых компонентов системы и их взаимодействие. Основные компоненты серверной части интернет-магазина:

1. Компонент аутентификации и авторизации:
    - Обеспечивает регистрацию, вход в систему, управление пользовательскими правами и ролями.
2. Каталог товаров:
    - Управляет информацией о товарах, включая категории, характеристики, изображения.
    - Включает модели для товаров, категорий, а также механизмы для фильтрации товаров.

3. Корзина и оформление заказа:
    - Обрабатывает добавление товаров в корзину, расчеты общей стоимости и оформление заказа.
    - Включает модели для корзины и заказов, а также логику обработки заказов.
4. Административная панель:
    - Предоставляет интерфейс для администраторов для управления пользователями, товарами, заказами и контентом сайта.
    - Использует встроенную административную панель Django для управления данными.
5. Фильтрация:

Обеспечивает быстрый и точный поиск товаров по различным критериям, а также фильтрацию результатов поиска.

2.1.3. Диаграммы взаимодействия компонентов и данных



Описывает шаги выполнения определенной задачи или процесса в приложении.

Позволяет выявить потоки управления и принятия решений.

Улучшает понимание работы системы с точки зрения выполнения конкретных операций