### **Анализ и сравнение написания веб-приложений с использованием Django, Flask и FastAPI**

### 

### 

[**Анализ и сравнение написания веб-приложений с использованием Django, Flask и FastAPI 1**](#_6mbjs6gfldf1)

[**1. Введение**](#_bwarub3fwefv)  2

[**2. Основные понятия и определения 2**](#_vkcwrdyt0b51) 3

[**3. Методы и подходы к разработке 3**](#_a8fg325q42ct)

[3](#_y68a5jrdusyi)

[**4. Обзор инструментов для разработки веб-приложений на Python: Django, Flask, FastAPI 3**](#_8zaf2764ff2a)

[4](#_f87sp1kpxou4)

[**5. Проектирование приложения 4**](#_k340l87vnnbh)

[4](#_o52mcnvgj423)

[**6. Разработка в соответствии с созданной документацией 5**](#_p6brm37qbcoi)

[Django 5](#_8l9l0km7ua6z)

[Flask 5](#_rpk9eaec3iy7)

[FastAPI 5](#_vb7xaew46zb2)

[**7. Анализ написания web-приложений с использованием разных фреймворков 5**](#_q430xqset8uv)

[Django 6](#_a1uqznex9miz)

[Flask 6](#_x053v5qc0wpf)

[FastAPI 6](#_90volo2d1i8q)

[**Анализ и сравнение веб-приложений 7**](#_n92ia2v55adl)

[1. Django 7](#_2z9sued94ku6)

[2. Flask 8](#_imbspgminbod)

[3. FastAPI 9](#_pwkh6r19ut9v)

[**Сравнительная таблица 10**](#_ky2898hewou0)

[**Вывод 10**](#_ovblu44jls0q)

[**Заключение 11**](#_xzdnaok1113e)

[Обзор выполненной работы 11](#_d7q1pcak8zf2)

[Дальнейшие планы 11](#_nilaxtazlobr)

### 

## **1. Введение**

### **Обоснование выбора темы**

### Выбор фреймворка для разработки web-приложений влияет на продуктивность команды, структуру проекта и последующую поддержку кода. Django, Flask и FastAPI являются одними из самых популярных Python-фреймворков для этих целей.

### Тема дипломной работы была выбрана в связи с необходимостью практического сравнения работы с различными веб-фреймворками для поиска наиболее подходящих инструментов для решения задач разных стилей и размеров проектов.

### **Цель и задачи исследования**

### Цель: создать пример приложения в сфере продажи видеоигр и проанализировать различия в подходах к проектированию и реализации с использованием Django, Flask и FastAPI.

### **Задачи:**

### Изучить концепции и подходы к разработке web-приложений на базе Python.

### Создать магазин видеоигр на Django, Flask и FastAPI.

### Проанализировать сложности и различия в организации кода и функциональных возможностей между тремя фреймворками.

### Провести сравнительный анализ производительности и удобства разработки.

### Оценить перспективы использования каждого фреймворка для различных типов проектов.

## **2. Основные понятия и определения**

### **Web-приложение**

### Web-приложение представляет собой программное обеспечение, работающее на веб-сервере и предоставляющее пользователям доступ к функциональности через веб-браузер. Такие приложения включают фронтенд (интерфейс пользователя) и бэкенд (серверная логика).

### **Фреймворк**

### **Фреймворк -** это программная платформа, предоставляющая разработчику набор инструментов и библиотек для ускорения разработки и стандартизации процесса.

### Django: полнофункциональный фреймворк "все в одном" для разработки масштабируемых приложений.

### Flask: минималистичный фреймворк с высокой степенью гибкости.

### FastAPI: современный фреймворк для создания высокопроизводительных API.

## **3. Методы и подходы к разработке**

### **Для реализации web-приложений использовались три подхода:**

### Использование Django: акцент на встроенные возможности, такие как ORM, аутентификация, и админ-панель.

### Использование Flask: реализация всех компонентов с использованием сторонних библиотек для достижения гибкости.

### Использование FastAPI: создание API с авто-документацией и асинхронной обработкой запросов.

### Каждый подход был реализован на основе общего примера: магазина видеоигр, включающего главную страницу, каталог, корзину и оформление заказа.

## **4. Обзор инструментов для разработки веб-приложений на Python: Django, Flask, FastAPI**

### **Django**

### Django предоставляет множество встроенных компонентов:

### ORM: для работы с базами данных.

### Аутентификация: система управления пользователями и правами доступа.

### Админ-панель: автоматическая генерация интерфейса для управления данными.

### Маршрутизация: гибкая система маршрутов на основе регулярных выражений.

### Преимущества: высокая скорость разработки благодаря встроенным инструментам, безопасность, сообщество и экосистема.

### Недостатки: высокая сложность настройки для небольших проектов, более низкая производительность в сравнении с FastAPI.

### **Flask**

### Flask предоставляет минималистичный подход:

### Подключение только необходимых библиотек.

### Простота структуры проекта.

### Преимущества: гибкость, лёгкость освоения, высокая производительность для простых задач.

### Недостатки: необходимость ручной настройки множества компонентов, сложность масштабирования для больших проектов.

### **FastAPI**

### FastAPI фокусируется на создании API:

### Автоматическая генерация документации (Swagger UI).

### Валидация данных с использованием Pydantic.

### Асинхронная обработка запросов.

### Преимущества: высокая производительность, современный подход, удобство работы с типами данных.

### Недостатки: меньшая экосистема по сравнению с Django, необходимость освоения новых концепций.

## **5. Проектирование приложения**

### **Основные требования**

* Разработка приложения для продажи ключей видеоигр.
* Реализация главной страницы, каталога игр, корзины и оформления заказа.
* Поддержка добавления и удаления игр из корзины.

### **Технические требования**

1. Использование Python.
2. Реализация приложения на Django, Flask и FastAPI.
3. Подключение баз данных для хранения данных о товарах и заказах.
4. Использование HTML-шаблонов для отображения интерфейса.

## **6. Разработка в соответствии с созданной документацией**

### **Django**

Разработка приложения на Django включала:

* Использование встроенных моделей для хранения данных об играх и заказах.
* Реализация маршрутов с помощью функций-представлений.
* Настройка шаблонов для отображения страниц.
* Использование админ-панели для управления данными.

**Структура проекта**

**Домашняя страница**

Домашняя страница презентует нам удобное меню из главной страницы, магазина и корзины. Также тут сразу представлены игры из магазина для удобства покупателей.

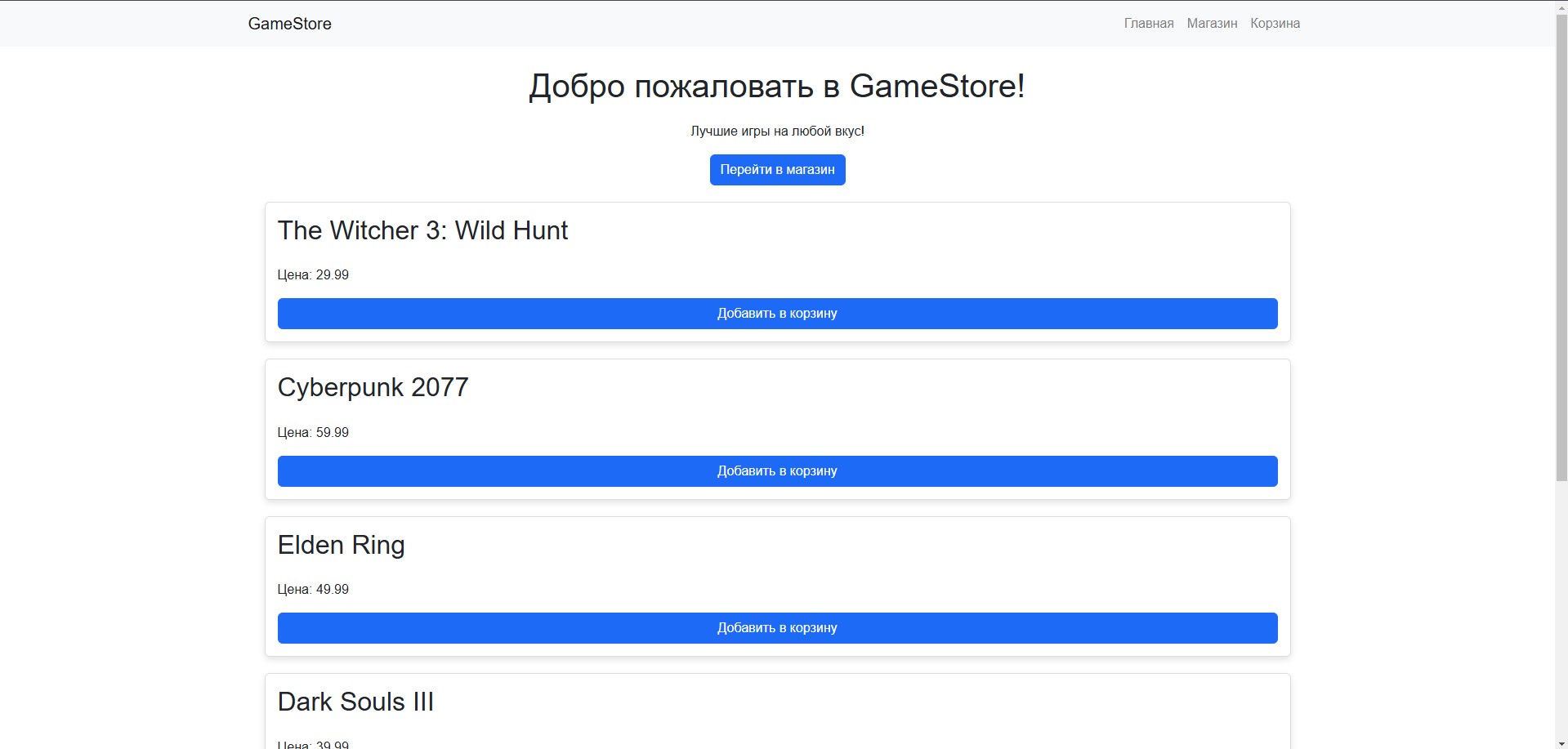
****

рисунок 1. Домашняя страница Django

**Страница Магазина**

Тут же представлен весь каталог игр магазина GameStore, есть возможность добавить игры в корзину.

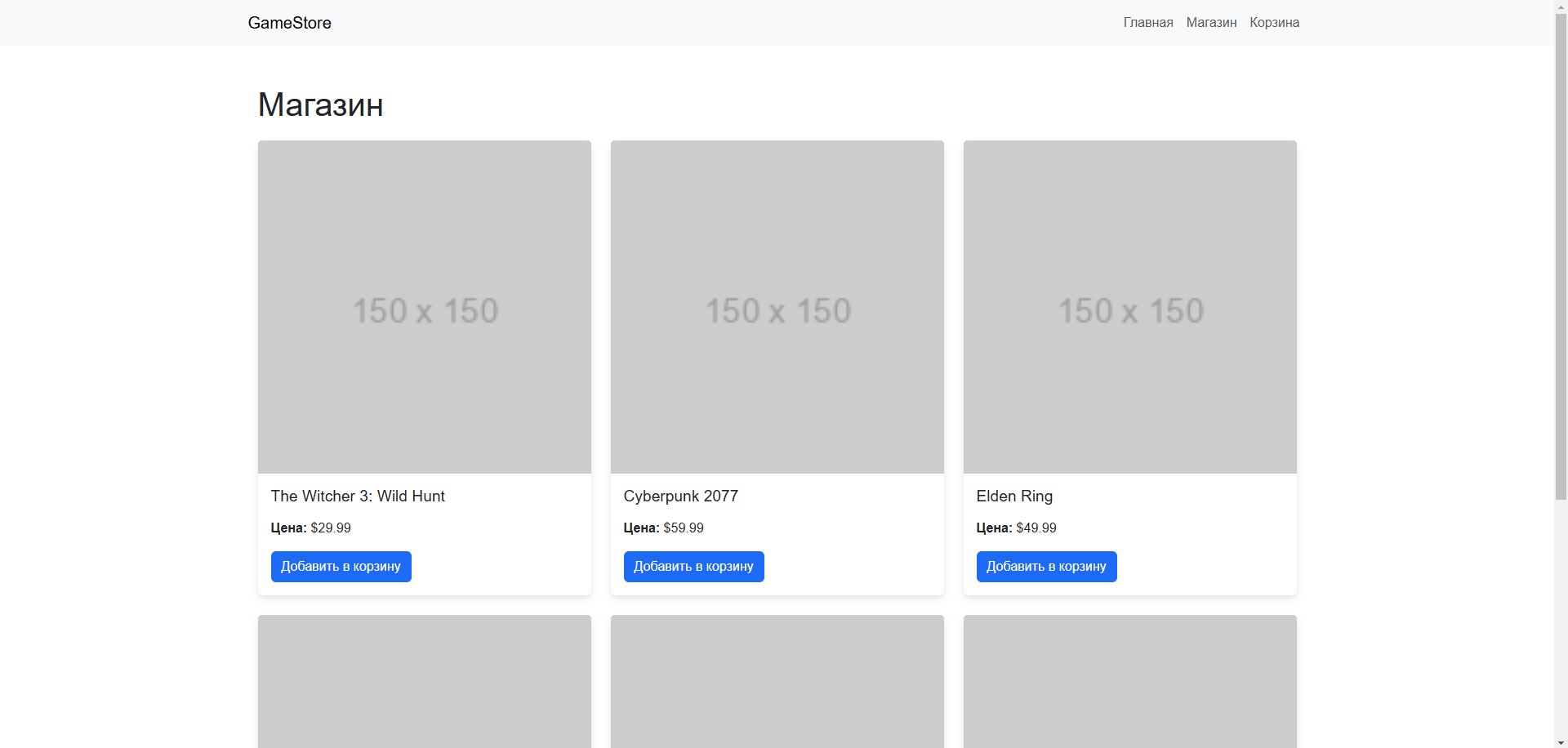


рисунок 2. Страница Магазина Django

**Пустая корзина**

Здесь можно увидеть, что ваша корзина пуста, а значит вы ещё не добавили ни одну игру для покупки.

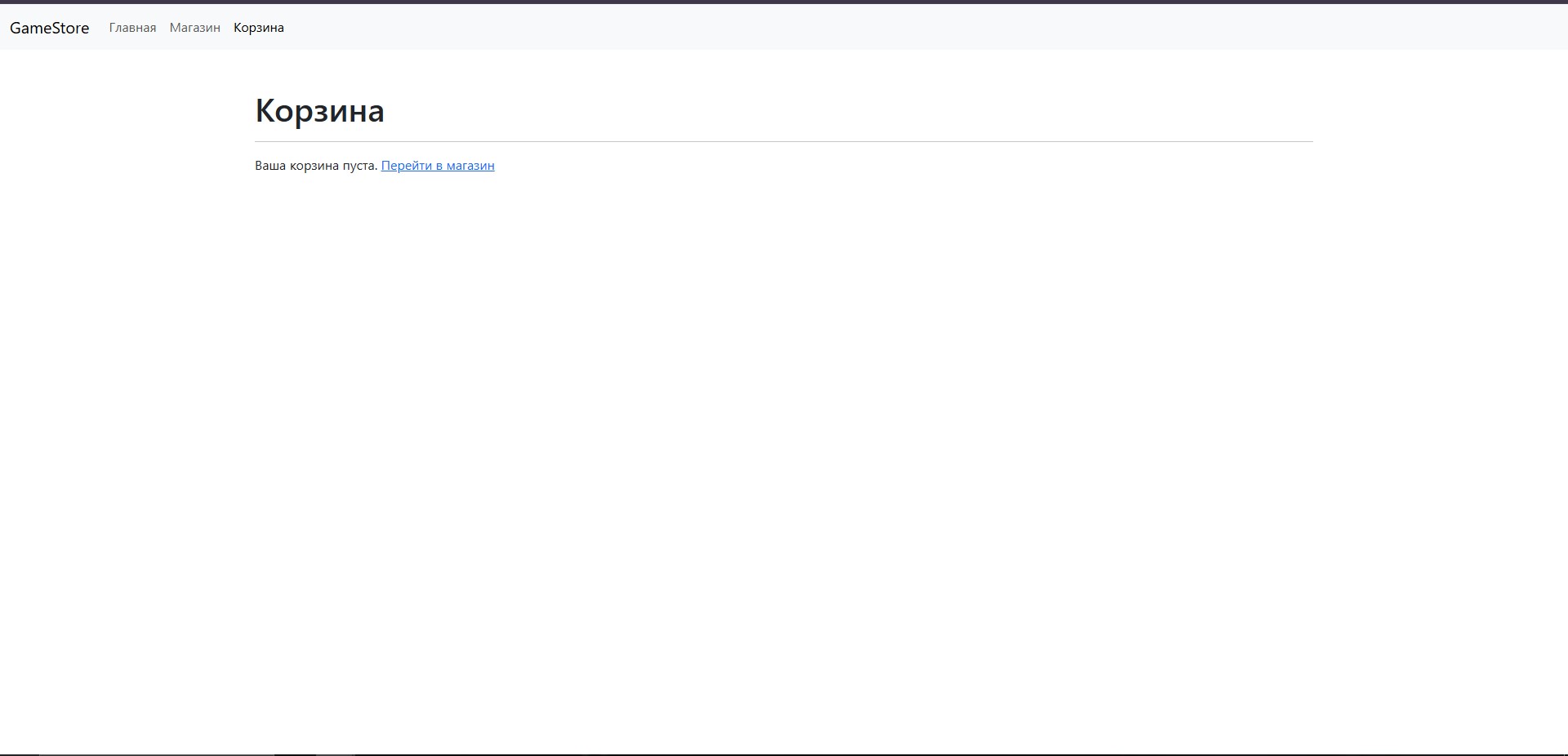


рисунок 3. Пустая корзина Django

**Полная корзина**

Здесь мы можем видеть, что добавили несколько игр в свою корзину, а одну игру даже взяли дважды (для себя и друга). Отсюда мы можем нажать на “продолжить покупки” и нас перекинет на вкладку “Магазин”, либо же перейти к оформлению заказа.

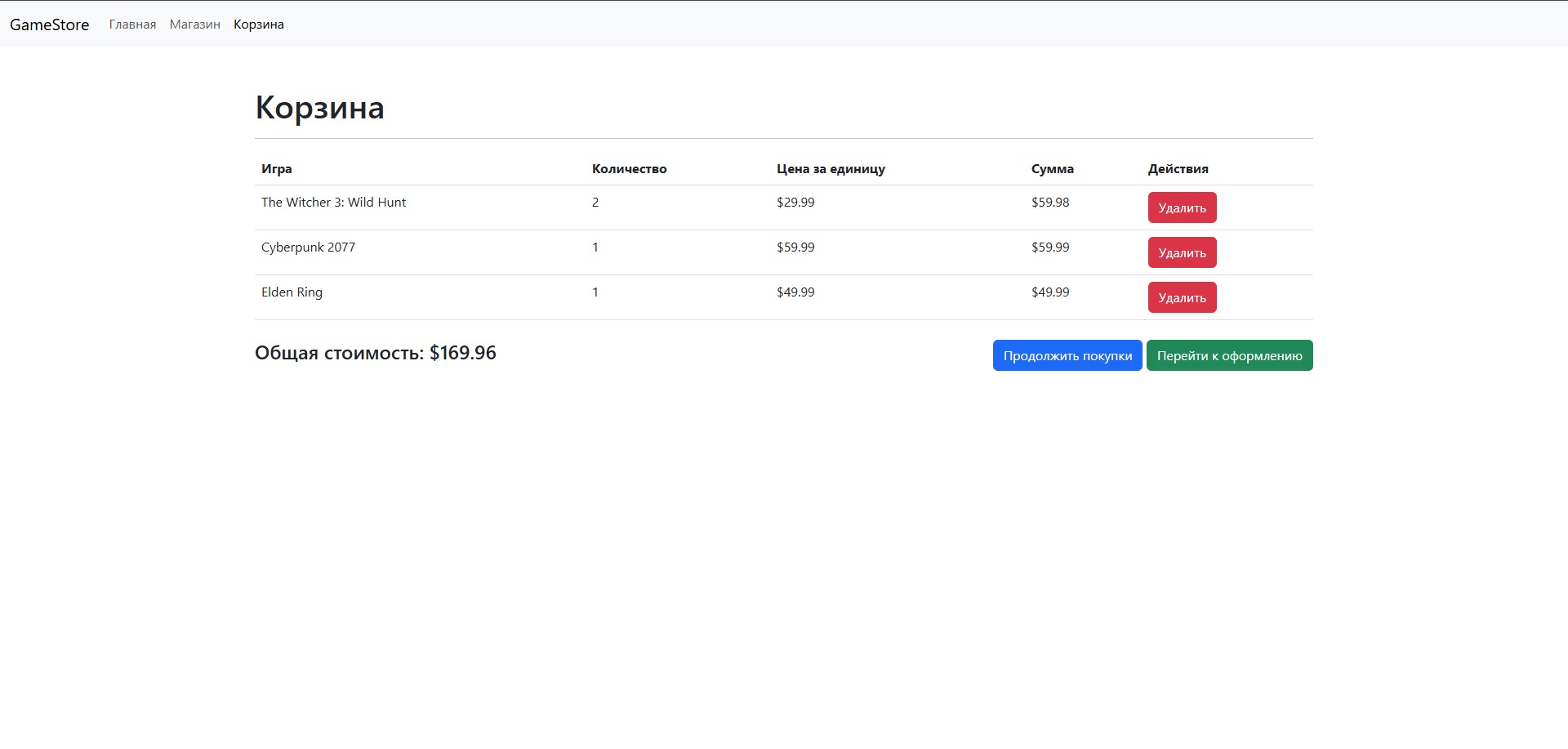


рисунок 4. Полная корзина Django

**Страница оформления заказа**

На данной странице мы видим игры, которые мы хотим приобрести, цены на них, а также количество выбранных копий. Также показана общая стоимость заказа.

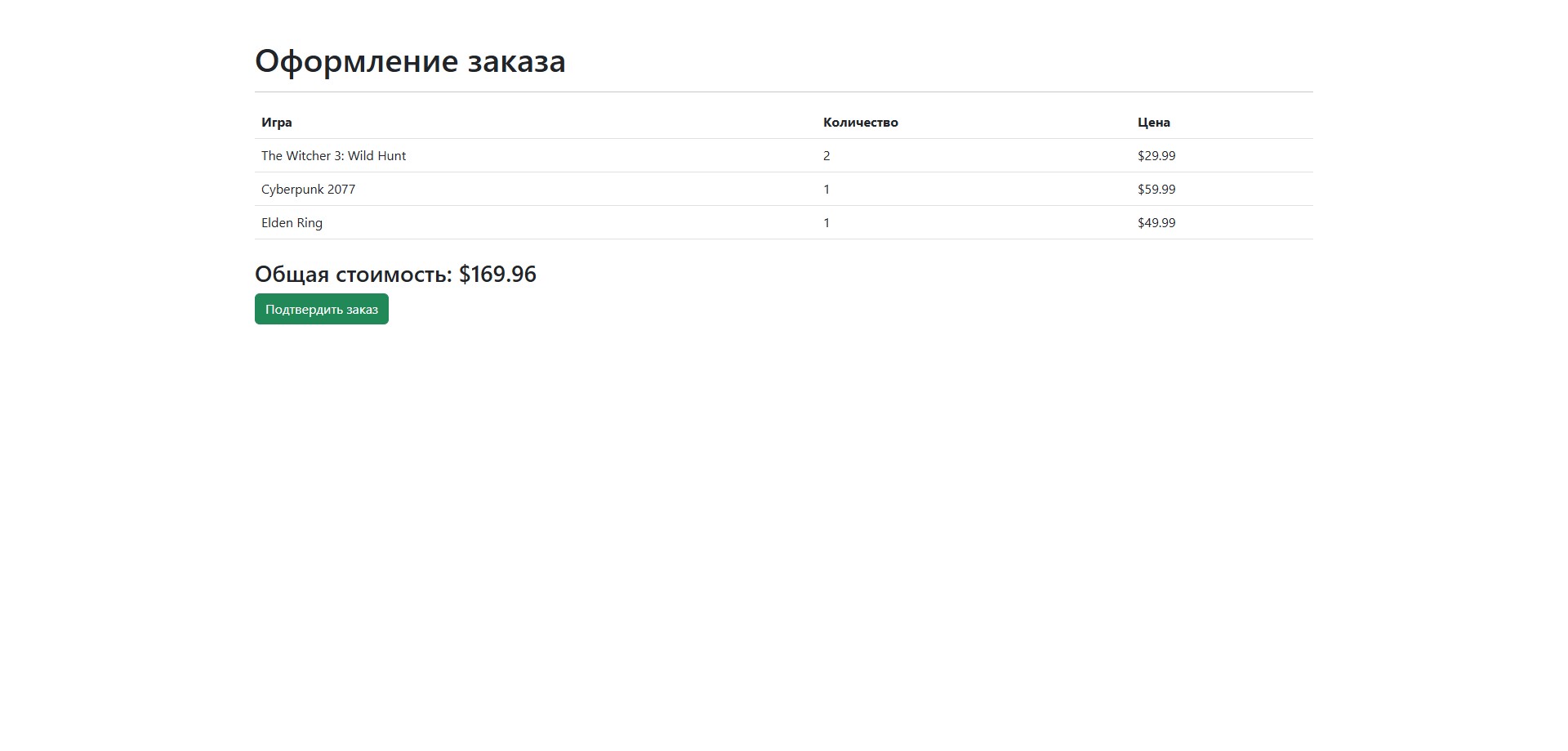


рисунок 5. Страница оформления заказа Django

**Шаблоны страниц**

base.html – базовый шаблон, подключающий Bootstrap, на его основании пишутся остальные. В нём прописана навигационная панель.

cart.html - шаблон корзины с которой можно перейти на другие страницы сайта.

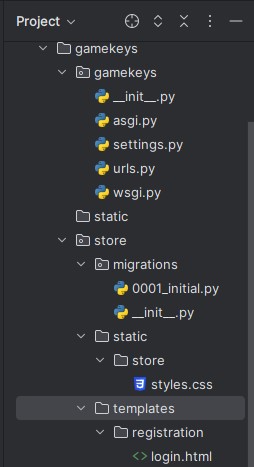
game\_list.html - шаблон главной страницы с приветствием посередине и списком игр.

game\_detail.html - шаблон по которому можно добавлять игры через superuser-a или Django shell.

store\_game\_list.html - шаблон страницы магазина, где показан более подробный список игр с картинками.

checkout.html - шаблон заполненной корзины с возможностью перейти в магазин, удалить игры или оформить свой заказ.

**Пример файловой структуры проекта:**

****

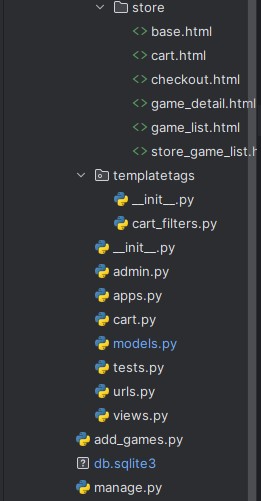
****

рисунок 6. файловая структура Django

### **Flask**

Разработка приложения на Flask включала:

* Подключение сторонних библиотек для работы с базами данных и аутентификацией.
* Настройка ручной маршрутизации.
* Создание кастомной логики для обработки корзины и оформления заказов.

**Структура проекта**

Основные страницы имеют тот же функционал, что и в Django. Поэтому не буду их расписывать, а просто покажу скриншоты.

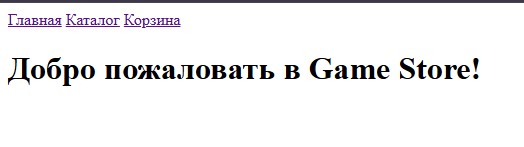


рисунок 7. Домашняя страница Flask



Рисунок 8. Страница Магазина Flask

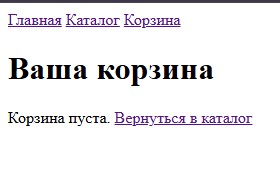


Рисунок 9. Пустая корзина Flask



рисунок 10. Полная корзина Flask

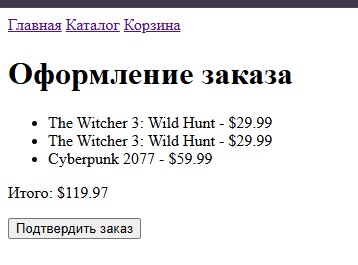


рисунок 11. Страница оформления заказа Flask

**Шаблоны страниц:**

base.html – базовый шаблон, в нём прописана навигационная панель.

index.html - Главная страница с приветствием.

catalog.html - каталог с играми и возможностью добавления их в корзину.

cart.html - шаблон пустой корзины.

checkout.html - шаблон заполненной корзины и возможностью удалять игры, перейти в каталог игр, либо же оформить заказ.

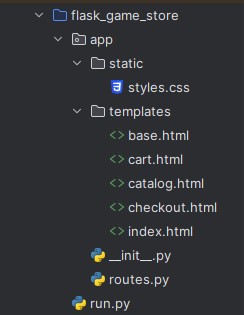


рисунок 12. Файловая структура Flask

### **FastAPI**

Разработка приложения на FastAPI включала:

* Реализацию асинхронных маршрутов.
* Использование Pydantic для валидации данных.
* Автоматическую генерацию документации API.
* Реализацию фронтенда с использованием Jinja2

**Структура проекта**

Сайт практически идентичен тому, что написан на Flask. Поэтому обойдемся без скриншотов.

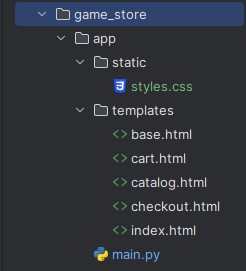
.

рисунок 13. Файловая структура FastAPI

**Шаблоны страниц** идентичны тому, что и на Flask.

## **7. Анализ web-приложений с использованием разных фреймворков**

### **Django**

* **Преимущества:** высокая скорость разработки, встроенные инструменты.
* **Недостатки:** избыточность для простых проектов, более сложная настройка.

### **Flask**

* **Преимущества:** гибкость, минимализм.
* **Недостатки:** увеличение времени разработки из-за необходимости настройки.

### **FastAPI**

* **Преимущества:** высокая производительность, современный подход.
* **Недостатки:** сложность освоения для новичков, ограниченная экосистема.

### 

### **Анализ и сравнение веб-приложений**

#### **1. Django**

**Характеристика:**

* Django — это фреймворк "всё-в-одном" с концепцией "батарейки включены".
* Предназначен для разработки крупных и структурированных веб-приложений.

**Плюсы:**

* **Множество встроенных инструментов:**
  + ORM для работы с базами данных.
  + Аутентификация, админ-панель, формы, управление сессиями.
* **Ускорение разработки:**
  + Большое количество готовых решений для типичных задач.
  + Простая реализация сложных операций благодаря встроенным инструментам.
* **Сообщество и экосистема:**
  + Большое количество пакетов, например, Django REST Framework для API.
* **Безопасность:**
  + Встроенная защита от SQL-инъекций, CSRF, XSS и других уязвимостей.

**Минусы:**

* **Сложность для простых приложений:**
  + Для небольших приложений Django может быть избыточным.
  + Требует строгого следования структуре проекта.
* **Скорость:**
  + Django медленнее, чем FastAPI, для приложений с интенсивной нагрузкой на API.

**Когда использовать:**

* Крупные веб-приложения с пользовательской аутентификацией, админ-панелью и сложными базами данных.
* Проекты, требующие быстрой разработки благодаря готовым компонентам.

#### **2. Flask**

**Характеристика:**

* Flask — это минималистичный веб-фреймворк с возможностью расширения.
* Даёт разработчику полный контроль над приложением.

**Плюсы:**

* **Простота:**
  + Легко начать разработку.
  + Подходит для небольших и простых приложений.
* **Гибкость:**
  + Нет строгой структуры проекта, можно настроить всё под себя.
  + Возможность выбирать сторонние библиотеки.
* **Производительность:**
  + Лёгкий и быстрый, поскольку минимально использует ресурсы.

**Минусы:**

* **Ручная настройка:**
  + Для добавления аутентификации, ORM или админ-панели нужно использовать дополнительные библиотеки (например, Flask-SQLAlchemy, Flask-Login).
* **Сложность масштабирования:**
  + По мере роста проекта поддержка кода может стать сложной.

**Когда использовать:**

* Небольшие и средние проекты, где важна гибкость.
* Простые API или прототипы.

#### **3. FastAPI**

**Характеристика:**

* FastAPI — это современный фреймворк для создания API с использованием Python.
* Использует типизацию Python для автоматической генерации документации и валидации.

**Плюсы:**

* **Скорость:**
  + Высокая производительность благодаря использованию Starlette и Pydantic.
* **Типизация:**
  + Интеграция с аннотациями типов Python для автоматической проверки данных.
* **Документация:**
  + Автоматическая генерация документации API (Swagger UI и ReDoc).
* **Асинхронность:**
  + Полностью поддерживает асинхронные запросы (async/await), что делает его идеальным для приложений с высокой нагрузкой.
* **Современность:**
  + Легко интегрируется с Python 3.9+ и популярными библиотеками.

**Минусы:**

* **Ограниченная экосистема:**
  + Встроенных инструментов меньше, чем у Django.
* **Крутая кривая обучения для новичков:**
  + Требует понимания типизации и асинхронного программирования.

**Когда использовать:**

* Быстродействующие API и микросервисы.
* Современные проекты с активным использованием асинхронности.
* Если важна авто-документация.

### **Сравнительная таблица**

| **Характеристика** | **Django** | **Flask** | **FastAPI** |
| --- | --- | --- | --- |
| **Простота освоения** | Средняя | Высокая | Средняя |
| **Гибкость** | Низкая | Высокая | Средняя |
| **Интеграция инструментов** | Встроенные компоненты | Ручная настройка | Ограниченная |
| **Производительность** | Низкая для API | Средняя | Высокая |
| **Документация API** | Пакеты, например DRF | Ручная настройка | Автоматическая |
| **Поддержка асинхронности** | Ограниченная | Частичная | Полная |
| **Рекомендуемые проекты** | Крупные сайты | Простые приложения | Высоконагруженные API |

### 

### **Вывод**

* **Django:** идеален для больших проектов с пользовательской аутентификацией, сложной логикой и высокой безопасностью. Используется, когда важна быстрая разработка и встроенные решения.
* **Flask:** подходит для небольших и средних проектов, где важна простота и контроль. Он гибкий, но требует больше времени для настройки.
* **FastAPI:** лучший выбор для создания быстродействующих API и современных приложений с использованием типизации и асинхронности.

Ваш выбор зависит от специфики проекта:

* Выбирайте **Django**, если нужен полный набор инструментов "из коробки".
* Используйте **Flask** для простоты и гибкости.
* Остановитесь на **FastAPI**, если важны производительность и работа с API.

## **Заключение**

### **Обзор выполненной работы**

В рамках данной работы были изучены три фреймворка для разработки web-приложений: Django, Flask и FastAPI. На основе разработанных приложений были проанализированы их особенности, преимущества и недостатки.

### **Дальнейшие планы**

Будущая работа может быть направлена на более глубокое изучение производительности и масштабируемости приложений, а также интеграцию с другими технологиями, такими как контейнеризация и облачные сервисы.

**Приложение 1**

**Список необходимых библиотек:**

**annotated-types==0.7.0**

**anyio==4.7.0**

**asgiref==3.8.1**

**blinker==1.9.0**

**click==8.1.8**

**colorama==0.4.6**

**databases==0.9.0**

**Django==4.2.17**

**exceptiongroup==1.2.2**

**fastapi==0.115.6**

**Flask==3.1.0**

**greenlet==3.1.1**

**h11==0.14.0**

**idna==3.10**

**importlib\_metadata==8.5.0**

**itsdangerous==2.2.0**

**Jinja2==3.1.5**

**MarkupSafe==3.0.2**

**pydantic==2.10.4**

**pydantic\_core==2.27.2**

**sniffio==1.3.1**

**SQLAlchemy==2.0.36**

**sqlparse==0.5.3**

**starlette==0.41.3**

**typing\_extensions==4.12.2**

**tzdata==2024.2**

**uvicorn==0.34.0**

**Werkzeug==3.1.3**

**zipp==3.21.0**