**Дипломная работа по теме: Разработать простые веб-приложения с использованием Django, Flask и FastAPI, провести их сравнение.**  
  
**Автор: Шпакович Сергей Евгеньевич**

**Оглавление дипломной работы:**  
  
1. Введение. 3  
  
Обоснование выбора темы: 4  
  
Определение цели и задач исследования: 4  
  
2. Основные понятия и определения. 4  
  
3. Методы и подходы к разработке. 5  
  
Архитектура веб-приложения. 5  
4. Обзор популярных инструментов для разработки веб-приложений на Python 6

5. Проектирование приложения. 7  
  
Планирование и анализ требований. 7

Основные требования. 8  
  
Технические требования. 8  
  
6. Разработка в соответствии с созданной документацией. 8  
  
Разработка. 9  
  
7. Анализ и интерпретация результатов. 15

Рекомендации по выбору фреймворка в зависимости от проекта. 17  
  
8. Заключение. 18  
  
Обзор выполненной работы.18

**1.Введение.**  
  
**Обоснование выбора темы:**  
  
1. Веб приложение- это инновационный способ организации и предоставления пользователю широкого спектра услуг и функционала, доступных в онлайн режиме. Основное отличие веб приложений от обычных сайтов заключается в их возможности взаимодействовать и обрабатывать данные в режиме реального времени, без необходимости обновления страницы.

2. Потребности рынка: веб приложения обладают широким спектром применения, например, они могут использоваться для создания интернет-магазинов, системы управления контентом, социальных сетей и многое другое. Благодаря возможности взаимодействия в реальном времени, веб приложения предоставляют пользователям новые уровни комфорта и удобства, а также позволяют эффективно использовать ресурсы Интернета. Компании стремятся использовать наиболее эффективные модели для улучшения качества своих продуктов и услуг.  
  
3. Практическая значимость: Работа в области разработки веб приложения позволит применить полученные знания на практике и даст опыт использования различных приложений в зависимости от поставленных задач. Это позволит создавать интересные и полезные для пользователей возможности.  
  
4. Личный интерес и потенциальные перспективы: Использование различных инструментов и создание приложений на различных фреймворках позволяет развить и отработать опыт создания интересных и разносторонних проектов. Кроме того, такие навыки востребованы на рынке труда, что открывает перспективы для карьерного роста и развития.  
  
Таким образом, выбор темы "Разработать простые веб-приложения с использованием Django, Flask и FastAPI, провести их сравнение.”  
обусловлен ее актуальностью, потребностями рынка, практической значимостью и личным интересом, что делает эту тему подходящей для проведения дипломной работы.  
  
**Определение цели и задач исследования:**  
  
Цель исследования: Разработать простые веб-приложения с использованием Django, Flask и FastAPI, провести их сравнение.  
  
Задачи исследования:  
  
1. Обзор фреймворков для написания веб приложений объектов: Провести анализ существующих инструментов, определить и выбрать наиболее подходящие под имеющуюся задачу.  
  
2. Разработка сценариев сравнения: Разработать алгоритм сравнения эффективности моделей, определить критерии сравнения и набрать минимальный набор данных для первичной проверки.  
  
3. Создание платформ и проведение тестирования: реализовать платформы в формате веб-приложений, оценить эффективность различных фреймворков, и провести их тестирование.  
  
4. Написание дипломной работы: Составить дипломную работу, включающую в себя введение, обзор литературы, методологию и результаты исследования, анализ результатов, выводы и рекомендации.  
  
Цели и задачи исследования направлены на получение практических результатов, которые позволят сформировать условия выбора применения различных фреймворков при создании веб-приложений.

**2. Основные понятия и определения**  
  
  
 **Фреймворк (Framework)**: Программная платформа, которая предоставляет готовые компоненты и инструменты для разработки приложений. В контексте веб-приложений часто используются Django, FastAPI или Flask.

**Фронтенд** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *front end, frontend*) — презентационная часть [информационной](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) или [программной](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) системы, её [пользовательский интерфейс](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81)

**Бэкенд** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *back end, backend*) — базисная часть системы, её внутренняя реализация (простым языком, *серверная часть программы, что не видит пользователь*).  
  
**Веб-сайт** — это набор логически связанных между собой веб-страниц, расположенных в Сети под единым именем — доменом..  
  
 **Веб-приложение (Web Application)**: Программное приложение, которое работает на веб-сервере и доступно через браузер. В контексте данного проекта, веб-приложения будет использоваться для сравнения используемых фреймворков .

**Шаблоны** - это HTML - файлы, которые содержат динамические элементы и могут изменяться в зависимости от переданных данных.  
  
  
**3. Методы и подходы к разработке**  
  
 После выбора модели, необходимо интегрировать её в веб-приложение. Это может включать в себя загрузку модели, настройку её параметров и подключение к фреймворку для выполнения анализа.  
  
**Архитектура веб-приложения**  
  
Frontend и Backend: фронтенд (интерфейс пользователя) и бэкенд (серверная логика). Фронтенд может быть разработан с использованием HTML, CSS и JavaScript, а бэкенд может быть реализован на Python с использованием фреймворков, таких как Flask, FastAPI или Django.  
  
  
  
  
**4. Обзор популярных инструментов для разработки разработки веб-приложений на Python**  
  
Основные – Django, Flask, FastAPI, ниже более подробное описание.  
  
**Django**  
  
Django —фреймворк для веб-разработки на Python.  
  
Основные возможности:  
  
· Встроенная страница администрирования для управления данными.  
  
· ORM (Object-Relational Mapping) для работы с базами данных.  
  
· Поддержка аутентификации пользователей и управления доступом.  
  
· Мощная система маршрутизации URL.  
  
· Шаблонизатор для создания HTML-шаблонов.  
  
Особенности:  
  
Идеально подходит для больших и сложных проектов благодаря своей полной экосистеме.  
  
**Flask**  
  
Это фреймворк для веб-разработки на Python. Обеспечивает простоту и гибкость.  
  
Основные возможности:  
  
Минималистичный подход: только основные компоненты без лишних зависимостей.  
  
Высокая гибкость для добавления нужных библиотек и модулей.  
  
Jinja2 для шаблонов и Werkzeug для маршрутизации и обработки запросов.  
  
Особенности:  
  
Подходит для небольших и средних проектов, где требуется большая гибкость.  
  
**FastAPI**  
  
Это современный, высокопроизводительный фреймворк для создания API с Python 3.7+ на основе стандартов OpenAPI и JSON Schema.  
  
Основные возможности:  
  
· Высокая производительность благодаря использованию асинхронного программирования (async/await).  
  
· Автоматическая генерация интерактивной документации API.  
  
· Поддержка валидирования и аннотаций типов данных.  
  
· Интеграция с различными библиотеками и инструментами, такими как Pydantic и SQLAlchemy.  
  
Особенности:  
  
Часто используется для создания API и микросервисов.  
  
  
  
**5. Проектирование приложений**  
  
**Планирование и анализ требований**  
  
Выбор фреймворков и инструментов: Определение наиболее подходящего фреймворка для определенной задачи ( Django, Flask или FastAPI) и инструментов для разработки.  
  
Определение структуры приложения: Разработка схемы архитектуры, включающей фронтенд, бэкенд, базу данных .  
  
Разработка прототипов.  
  
Реализация основного функционала: Создание базовых версий приложений с минимально необходимым функционалом для тестирования и демонстрации.  
  
  
Создание API для взаимодействия с фронтендом.  
  
Реализация пользовательского интерфейса: Создание форм и страниц для загрузки изображений, отображения результатов и взаимодействия с пользователем.  
  
**Основные требования**

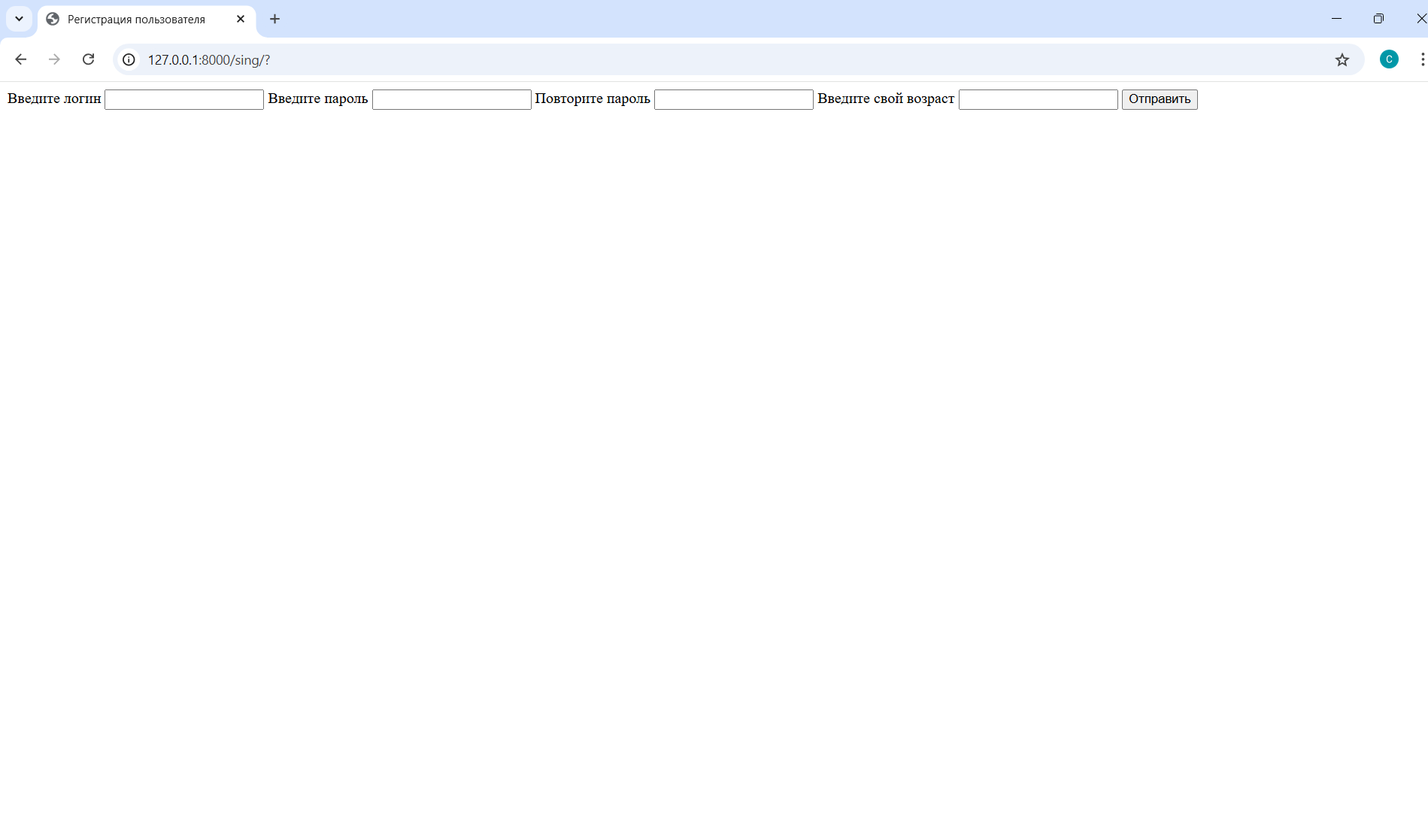
**Технические требования**  
  
Фронтенд: использовать HTML, CSS, JavaScript, возможно Bootstrap  
  
Бэкенд: Python с использованием фреймворков Django, FastAPI, Flask.  
  
База данных: Использование базы данных для хранения данных о пользователях и истории загрузок для Django, FastAPI.  
  
API: Разработка API для взаимодействия между фронтендом и бэкендом.  
  
  
  
**6. Разработка в соответствии с созданной документацией**  
  
**Планирование разработки**  
  
Разработка была разделена на несколько основных этапов: проектирование интерфейса, реализация серверной логики .

**Разработка**

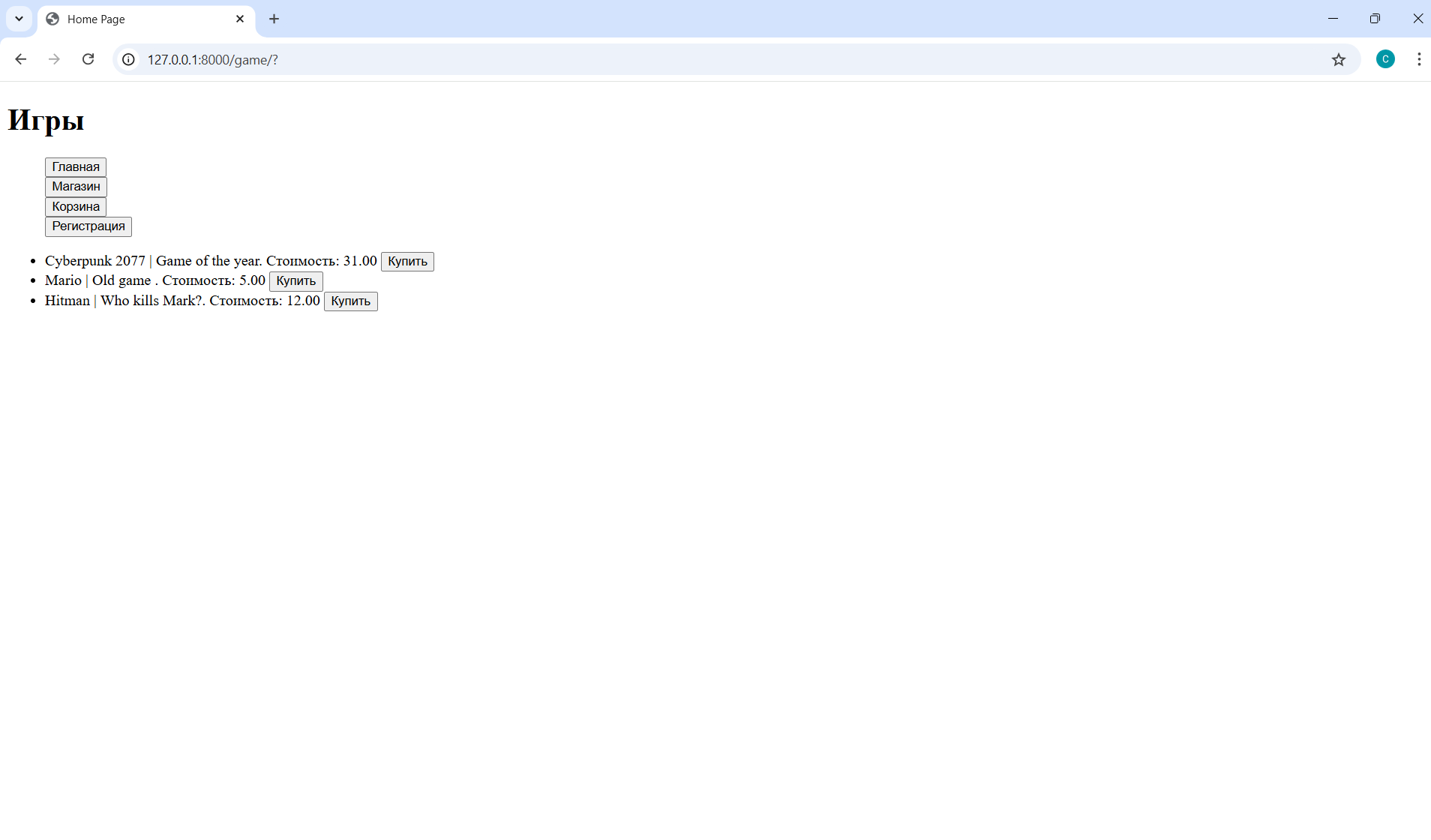
Фронтенд-разработка:

**Django**

1.Создана страница авторизации (Рис. 1)



Создан пользовательский интерфейс магазина игр (Рис.2)

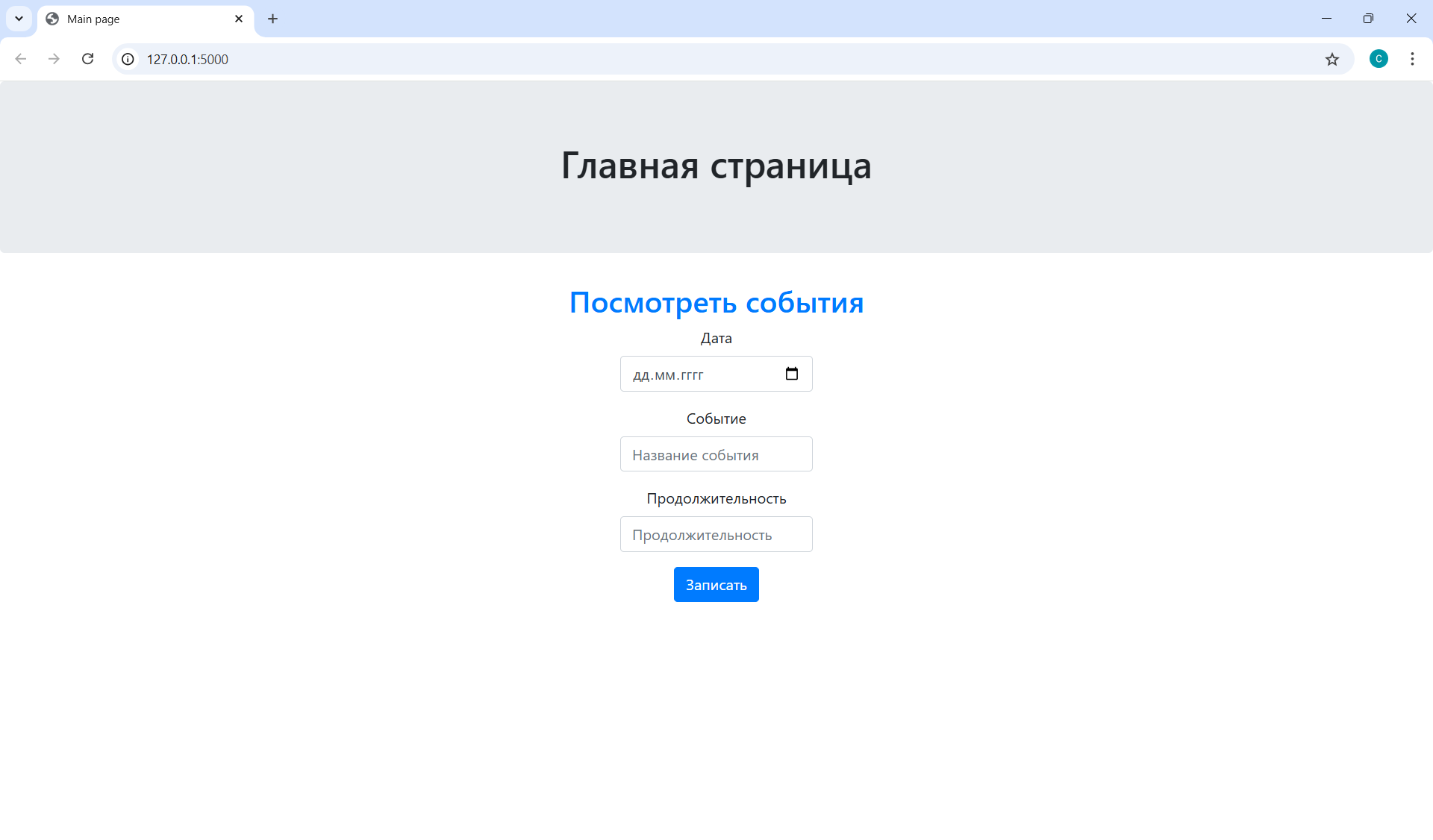
  
**FastApi**

Создан пользовательский интерфейс (Рис.3)



**Flask**

Создан ежедневник (Рис.4)



**Структура приложений.(с пояснениями)**

|  |
| --- |
| **FastAPI** |
| **app** |
| app  │ │ │ │─ backend │ │ │ │ │ │─ db.py  │ │ │ │ │ │─ db\_depends.py │ │  │ │  │ │─ migrations  │ │ │ │ │ │─ versions │ │ │ │ │ │ │ │5390fbed6e1b\_migration3....  │ │ │ │ │ │ │ │ a7547eb146db\_.py  │ │ │ │ │ │─ README  │ │ │ │ │ │─ env.py │ │ │ │ │ │─ script.py.mako │ │  │ │─ models # Модели данных (Pydantic) │ │ │ │ │ │─ \_\_init\_\_.py  │ │ │ │ │ │─ task.py  │ │ │ │ │ │─ user.py  │ │ │ │─ routers # Роутеры │ │ │ │ │ │─ \_\_init\_\_.py  │ │ │ │ │ │─ task.py  │ │ │ │ │ │─ user.py │  │ │ │ │─ \_\_init\_\_.py  │ │  │ │  │ │─ schemas.py │ # Cхемы валидации  │ │  │  │main.py # Основной файл приложения  │  │taskmanager.db │  alembic.ini |

|  |
| --- |
| **Django** |
| **UrbanDjango\_DB** |
| UrbanDjango  │ │ │ │─ static  │ │  │ │─ task1  │ │ │ │ │ │─ migrations # Миграции базы данных  │ │ │ │  │ │ │ │─0001\_initial.py │ │ │ │ │ │ │ │─ \_\_init\_\_.py  │ │ │  │ │ │─ \_\_init\_\_.py  │ │ │ │ │ │─ admin.py # Конфигурация админки  │ │ │ │ │ │─ apps.py │ │ │ │ │ │─ forms.py │ │ │  │ │ │─ models.py # Модели данных │ │ │ │ │ │─ tests.py  │ │ │ │ │ │─ views.py # Логика представления │ │ │  │ │ │ │ │─ templates # Папка для HTML - шаблонов │ │ │ │ │ │  │ │ │─ cart.html  │ │ │  │ │ │─ games.html  │ │ │  │ │ │─ menu.html  │ │ │  │ │ │ ─ platform.html  │ │ │  │ │ │─ registration\_page.html  │ │  │ │─ UrbanDjango  │ │  │ │ │─ \_\_init\_\_.py  │ │ │ │ │ │─ asgi.py  │ │ │  │ │ │─ settings.py # Настройки проекта.  │ │ │ │ │ │─ urls.py # Маршруты приложения  │ │ │  │ │ │─ wsgi.py # WSGI интерфейс │ │  │ │─ db.sqilite3  │ │  │ │─ manage.py # Скрипт управления проектом  │  requirements.txt |

|  |
| --- |
| **Flask** |
| **Flacon\_app** |
| Flacon\_app  │ │ │ │─ instance │ │ │ │ │ │─ db.py  │ │  │ │─ static # Папка для статических файлов (CSS, JS, изображений)  │ │  │ │─ templates # Папка для HTML - шаблонов  │ │ │ │ │ │─ bootstrap │ │ │ │ │ │ │ │base.html  │ │ │  │ │ │─ about.html  │ │ │ │ │ │─ events.html  │ │ │ │ │ │─ index.htmly │ │  │ │ ─ \_\_init\_\_.py │ │ │ │ ─ app.py # Основной файл приложения  │ │ |

**7. Анализ и сравнение результатов.**

|  |  |  |
| --- | --- | --- |
| **Django** | **Flask** | **FastApi** |
| **Характеристика.** | | |
| Полнофункциональный веб-фреймворк, предоставляющий готовые решения для наиболее популярных функций. | Легковесный фреймворк, позволяющий собирать необходимый функционал с помощью подключаемых плагинов | Современный, быстрый легковесный (высокопроизводительный) веб-фреймворк для создания API используя Python. |
| **Подход к разработке** | | |
| Объектно-ориентированный подход, который обеспечивает объектно-реляционное сопоставление (связывание баз данных и таблиц с классами). | Не поддерживает базовую систему управления базами данных, поэтому его лучше использовать для проектов с NoSQL или проектов без БД | Дает больше гибкости, так как не включает в себя какую-либо стандартизацию. |
| **Предпочитаемый объем проекта** | | |
| Отлично работает с крупными, многостраничными приложениями | Лучше всего подходит для небольших, компактных проектов | Подходит для небольших, но производительных  приложений |
| **Поддержка баз данных** | | |
| Поддерживает самые популярные системы управления реляционными БД, такие как MySQL, Oracle и т. д | Не поддерживает базовую систему управления базами данных, поэтому его лучше использовать для проектов с NoSQL или проектов без БД | Поддерживает работу с самыми разными системами баз данных: PostgreSQL, MySQL, SQLite, Oracle, Microsoft SQL Server и т. д. Причем мы не ограничены только реалиционными базами данных и равным образом использовать и нереляционные, так называемые NoSQL-системы баз данных. |
| **Возможность кастомизации** | | |
| Менее гибок из-за встроенных функций и инструментов. Разработчики не могут вносить изменения в модули. | Имеет расширяемые библиотеки, что дает хорошую гибкость в плане разработки | FastApi не ограничивает разработчика определенными правилами |
| **Отладка** | | |
| Не поддерживает виртуальную отладку | Имеет встроенный отладчик | Возможность подключения отладчика. |
| **Маршрутизация и представления** | | |
| Поддерживает сопоставление URL-адресов с представлениями через запрос | Позволяет отображать URL-адрес в представление на основе классов с помощью Werkzeug | Возможности фреймворка полностью направлены на разработку АРI |
|  | | |
| Поддерживает динамические HTML-страницы | Не поддерживает динамические HTML-страницы | Поддерживает динамические HTML-страницы |
| **Безопасность** | | |
| Имеет встроенную защиту от ряда распространенных векторов атак, таких как CSRF, XSS и SQL-инъекция. Эти меры безопасности помогают защитить код от уязвимостей. | Не предлагает встроенной системы безопасности. Настройки безопасности возможны, но с помощью дополнительных модулей. | Не предлагает встроенной системы безопасности. Настройки безопасности возможны, но с помощью дополнительных модулей. |
| **Главные преимущества** | | |
| Большое сообщество Быстрая разработка Простота в освоении Безопасность | Легкий  Обширная документация Минимум функций Полный контроль над процессом разработки | В создании сервисов REST компенсирует ряд недостатков других инструментов: он простой в использовании и в изучении, легко разворачивается, а также дает разработчику преимущество асинхронной обработки запросов. |

**Рекомендации по выбору фреймворка в зависимости от проекта.**

Итак, какой фреймворк использовать? Выбор между инструментами всегда и полностью зависит от контекста и проблемы.

Разработка на Django отлично подойдет, если вы хотите создать надежный и масштабный продукт, потому что он имеет множество встроенных функций и очень хорошо работает при большой нагрузке.

Если приложение нуждается в каких-то индивидуальных настройках, тогда лучше использовать Flask, так как он полностью кастомизируется. Приложение собирается с помощью модулей. Это увеличивает время первичных настроек, зато предоставляет огромную гибкость.

Если приложение должно быть небольшим, но производительным, и при этом простым в разработке и поддержке, то FastAPI — оптимальный выбор.

**8.Заключение.**

Обзор выполненной работы.

Разработаны простые веб-приложения с использованием фреймворков Django, Flask и FastAPI, проведено их сравнение, что позволяет определить сильные и слабые стороны каждого из них и выбрать необходимый для реализации конкретного проекта.