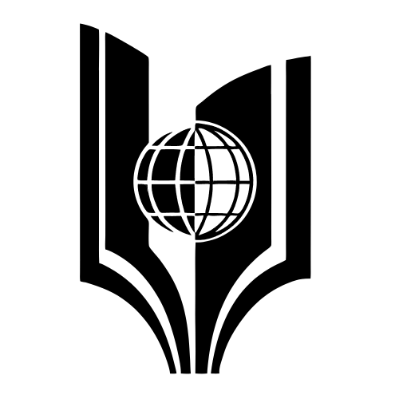
Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

**«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТУРИЗМА И СЕРВИСА»**

**КОЛЛЕДЖ**



**КУРСОВОЙ ПРОЕКТ**

по дисциплине «Инструментальные средства разработки программного обеспечения»

На тему: Разработка ИС “Нейросетевой поисковый движок”

По специальности: 09.02.07 Информационные системы и программирование

Студент: Попков Артем Алексеевич

Руководитель: Демидов Виктор Александрович

АННОТАЦИЯ

Данная пояснительная записка описывает курсовой проект, посвященный разработке веб-платформы "Каталог инди-игр". Проект представляет собой клиент-серверное веб-приложение, предназначенное для каталогизации, демонстрации и обсуждения инди-игр. Платформа призвана служить инструментом поддержки для начинающих разработчиков, предоставляя им площадку для повышения видимости своих проектов и получения обратной связи от сообщества.

Система реализована с использованием стека технологий: Python с фреймворком Flask для серверной части, SQLAlchemy для взаимодействия с базой данных SQLite, и стандартные веб-технологии (HTML, CSS, JavaScript) для клиентской части. Функционал включает регистрацию и авторизацию пользователей, просмотр каталога игр с фильтрацией по тегам (жанрам), добавление новых игр разработчиками (включая загрузку изображений и файлов игры), просмотр детальной информации об игре и профилей пользователей.

В работе рассмотрены актуальность задачи, постановка задачи, выбор и обоснование средств реализации, архитектура системы, детали программной реализации ключевых модулей, приведен контрольный пример работы и инструкция пользователя.

СОДЕРЖАНИЕ

АННОТАЦИЯ](аннотация)

СОДЕРЖАНИЕ](содержание)

[ВВЕДЕНИЕ](введение)

[1. ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ](1-теоретический-раздел)

[1.1. Постановка задачи](11-постановка-задачи)

[1.1.1. Актуальность и цели](111-актуальность-и-цели)

[1.1.2. Описание входной и выходной информации](112-описание-входной-и-выходной-информации)

[1.2. Схема и описание алгоритмов](12-схема-и-описание-алгоритмов)

[1.2.1. Общая архитектура системы](121-общая-архитектура-системы)

[1.2.2. Основные сценарии использования](122-основные-сценарии-использования)

[1.3. Обоснование выбора средств реализации](13-обоснование-выбора-средств-реализации)

[1.4. Описание среды выполнения](14-описание-среды-выполнения)

[1.5. Описание структуры Базы Данных](15-описание-структуры-базы-данных)

[2. ПРАКТИЧЕСКИЙ (РЕАЛИЗАЦИОННЫЙ) РАЗДЕЛ](2-практический-реализационный-раздел)

[2.1. Описание реализации программного кода](21-описание-реализации-программного-кода)

[2.1.1. Структура проекта](211-структура-проекта)

[2.1.2. Серверная часть (Backend - app.py)](212-серверная-часть-backend---apppy)

[2.1.3. Модели данных (models.py)](213-модели-данных-modelspy)

[2.1.4. Клиентская часть (Frontend - templates, static)](214-клиентская-часть-frontend---templates-static)

[2.1.5. Работа с файлами (загрузка изображений и игр)](215-работа-с-файлами-загрузка-изображений-и-игр)

[2.1.6. Безопасность](216-безопасность)

[2.2. Описание процесса отладки](22-описание-процесса-отладки)

[2.3. Характеристика программы](23-характеристика-программы)

[2.4. Контрольный пример](24-контрольный-пример)

[2.5. Инструкция пользователя](25-инструкция-пользователя)

[ЗАКЛЮЧЕНИЕ](заключение)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ](список-использованных-источников)

[ПРИЛОЖЕНИЯ](приложения)

ВВЕДЕНИЕ

Индустрия разработки игр стремительно развивается, и особое место в ней занимают инди-проекты – игры, создаваемые независимыми разработчиками или небольшими студиями без финансовой поддержки крупных издателей. Эти проекты часто отличаются оригинальностью идей, смелыми экспериментами с геймплеем и визуальным стилем. Однако, несмотря на творческий потенциал, начинающие инди-разработчики сталкиваются со значительными трудностями в продвижении своих игр. Крупные цифровые площадки (такие как Steam, Epic Games Store, Google Play, App Store) перенасыщены контентом, и новым играм без существенного маркетингового бюджета крайне сложно выделиться и найти свою аудиторию среди тысяч релизов и доминирующих ААА-проектов.

Кроме проблемы видимости, начинающие разработчики нуждаются в обратной связи от игроков для улучшения своих продуктов и в поддерживающем сообществе для обмена опытом и мотивации. Существующие платформы не всегда предоставляют удобные и сфокусированные инструменты для удовлетворения этих потребностей именно инди-сегмента.

Таким образом, актуальность данного курсового проекта обусловлена необходимостью создания специализированных веб-платформ, ориентированных на поддержку инди-сообщества. Такие платформы могут стать важным инструментом для повышения видимости малобюджетных игр, облегчения коммуникации между разработчиками и игроками, и в целом способствовать развитию независимой игровой разработки.

Цель работы: Разработать функциональную веб-платформу "Каталог инди-игр", предоставляющую инструменты для демонстрации, обсуждения и продвижения инди-игр, ориентированную на поддержку начинающих разработчиков.

Для достижения поставленной цели были решены следующие задачи:

1. Проанализированы существующие платформы и каталоги игр (Steam, itch.io, Game Jolt и др.), выявлены их сильные и слабые стороны применительно к потребностям инди-разработчиков.

2. Спроектирована архитектура веб-приложения и структура базы данных для хранения информации об играх, пользователях и связанных данных (теги/жанры).

3. Разработана серверная часть (backend) приложения с использованием фреймворка Flask (Python), реализующая основную бизнес-логику, включая регистрацию/авторизацию пользователей, управление данными игр и обработку файловых загрузок.

4. Разработана клиентская часть (frontend) приложения с использованием стандартных веб-технологий (HTML, CSS, JavaScript) и шаблонизатора Jinja2, обеспечивающая пользовательский интерфейс для просмотра каталога, добавления игр и взаимодействия с платформой.

5. Реализован функционал каталогизации игр с возможностью фильтрации по тегам (жанрам).

6. Реализован функционал добавления игр зарегистрированными пользователями, включая загрузку изображений (обложек) и файлов самой игры (архивов).

7. Проведено базовое тестирование работоспособности основных функций платформы.

Объект исследования: Процесс поддержки и продвижения инди-игр в цифровой среде.

Предмет исследования: Веб-платформа "Каталог инди-игр" как инструмент для каталогизации, демонстрации, продвижения инди-игр и поддержки сообщества разработчиков и игроков.

Методы исследования: Анализ предметной области, сравнительный анализ существующих решений, системный анализ, проектирование информационных систем, методы веб-разработки (клиент-серверная архитектура, RESTful-принципы для некоторых элементов), объектно-ориентированное программирование (при работе с моделями данных), тестирование программного обеспечения.

# Практическая значимость:

Разработанная платформа может быть использована начинающими разработчиками инди-игр как бесплатный инструмент для повышения видимости своих проектов, получения обратной связи от аудитории и формирования онлайн-портфолио.

Платформа представляет интерес для игроков, целенаправленно ищущих уникальные и небюджетные инди-проекты.

Проект демонстрирует применение фреймворка Flask и сопутствующих технологий для создания полнофункционального веб-приложения и может служить образовательным примером.

Результаты работы могут служить основой для дальнейшего развития платформы путем добавления новых функций

Структура работы: Пояснительная записка состоит из введения, теоретического раздела (постановка задачи, описание архитектуры и алгоритмов, обоснование выбора технологий, описание БД), практического (реализационного) раздела (описание реализации кода, отладки, характеристик программы, контрольный пример, инструкция пользователя), заключения, списка использованных источников и приложений (при необходимости).

1. ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

1.1. Постановка задачи

1.1.1. Актуальность и цели

Как было отмечено во введении, основной проблемой, на решение которой направлен проект, является недостаточная представленность и сложность продвижения инди-игр на фоне крупных коммерческих проектов и перенасыщенности существующих цифровых платформ. Начинающим разработчикам часто не хватает ресурсов и инструментов для того, чтобы их игры были замечены целевой аудиторией.

Цель: Создание веб-платформы, которая:

Предоставляет централизованный каталог специально для инди-игр.

Позволяет разработчикам легко публиковать информацию о своих играх (описание, теги, изображения, ссылки или файлы для скачивания).

Дает возможность игрокам находить и открывать для себя новые инди-проекты, используя фильтрацию по тегам (жанрам).

Формирует пространство для сообщества, где пользователи (разработчики и игроки) могут зарегистрироваться и взаимодействовать (в перспективе - через комментарии, оценки).

Служит инструментом поддержки для начинающих разработчиков, повышая их шансы на получение обратной связи и признания.

Задача заключается в разработке прототипа такой платформы с базовым, но ключевым функционалом, который может быть расширен в будущем.

1.1.2. Описание входной и выходной информации

# 1. Данные пользователя:

При регистрации: логин, email, пароль.

При авторизации: логин/email, пароль.

Действия пользователя: просмотр страниц, применение фильтров, добавление игры, загрузка файлов.

# 2. игры (при добавлении/редактировании разработчиком)

Название игры (текст).

Описание игры (текст).

Теги/Жанры (текст, например, "Экшен", "Головоломка", "RPG"). В текущей реализации до 3 тегов.

Изображение-обложка (файл изображения: png, jpg, jpeg, gif).

Файл игры (архив: zip, rar, 7z, tar, gz) (опционально).

# 3. Системные данные:

HTTP-запросы от браузера пользователя (GET, POST).

Конфигурационные параметры сервера (секретный ключ Flask, путь к БД, настройки загрузки файлов).

Выходная информация

# 1. Веб-страницы (HTML), стили (CSS), скрипты (JS): Динамически генерируемые сервером или статические файлы, отображаемые в браузере пользователя. Включают

Главная страница с каталогом игр.

Страница детального просмотра игры.

Страница профиля пользователя (пока базовая).

Формы для регистрации, авторизации, добавления игры.

# 2. Данные для пользователя:

Список игр (с изображениями, названиями, тегами).

Детальная информация о выбранной игре.

Сообщения об успехе (например, "Игра успешно добавлена", "Регистрация прошла успешно").

Сообщения об ошибках (например, "Пользователь с таким логином уже существует", "Неверный пароль", "Недопустимый формат файла").

Загружаемый файл игры (при запросе на скачивание).

# 3. Данные, сохраняемые в системе:

Записи в базе данных (новые пользователи, новые игры, связи между ними).

Загруженные файлы (изображения, архивы игр) на сервере.

# 4. Данные сессии: Идентификатор авторизованного пользователя, хранящийся в cookie браузера для поддержания состояния входа.

1.2. Схема и описание алгоритмов

1.2.1. Общая архитектура системы

Система построена по классической клиент-серверной архитектуре с использованием веб-фреймворка Flask.

Клиент (Frontend): Веб-браузер пользователя. Отправляет HTTP-запросы на сервер и отображает полученные HTML-страницы, стили и скрипты. Взаимодействие реализуется через стандартные механизмы веб (формы, ссылки). Используются HTML-шаблоны (`templates/`), отрисовываемые на сервере с помощью шаблонизатора Jinja2, статические файлы CSS (`static/главня.css` и др.) и JavaScript (`static/главная.js`) для оформления и некоторой клиентской логики (например, асинхронная проверка доступности логина/email).

Сервер (Backend) Приложение на Python с использованием фреймворка Flask (`app.py`). Отвечает за:

Обработку HTTP-запросов (маршрутизация URL с помощью декораторов `@app.route`).

Бизнес-логику: аутентификация пользователей, получение данных из БД, фильтрация игр, обработка форм, валидация данных.

Взаимодействие с базой данных: через SQLAlchemy ORM (`models.py`) для выполнения операций CRUD (Create, Read, Update, Delete) с данными пользователей и игр.

Управление сессиями пользователей: с использованием механизма сессий Flask для отслеживания авторизованных пользователей.

Обработку загрузки файлов: сохранение изображений и архивов игр в файловой системе сервера (`static/uploads/`).

Генерацию HTML-ответов: рендеринг Jinja2-шаблонов с передачей необходимых данных.

База данных (Database): SQLite (`instance/site.db`). Хранит информацию о пользователях (таблица `Account`) и играх (таблица `Database`). Выбор SQLite обусловлен простотой настройки и использования для прототипа. Структура БД описана в моделях SQLAlchemy (`models.py`).

Файловое хранилище: Директория на сервере (`static/uploads/`) для хранения загруженных пользователями изображений и файлов игр.

(Блочная диаграмма архитектуры)

```

+-------------------+ HTTP +---------------------+ SQLAlchemy +----------------+

| Веб-браузер |<--------------->| Flask Backend |<--------------------->| SQLite |

| (HTML, CSS, JS) | (Запросы/Ответы)| (app.py) | (Запросы к БД) | (instance/site.db)|

+-------------------+ +---------------------+ +----------------+

| |

| Загрузка/Отправка файлов | Чтение/Запись файлов

+------------------------------------->+---------------------+

| Файловая система |

| (static/uploads/) |

+---------------------+

```

1.2.2. Основные сценарии использования

1. Регистрация пользователя:

1. Клиент: Пользователь заполняет форму регистрации (логин, email, пароль) на главной странице (или отдельной странице `/register`) и отправляет ее (POST-запрос на `/` или `/register`). На клиенте может выполняться JavaScript-валидация и асинхронная проверка доступности логина/email через вызов `/check\_availability`.

2. Сервер (Flask):

Получает POST-запрос с данными формы.

Валидирует данные (проверка заполненности полей).

Проверяет уникальность логина и email в таблице `Account` базы данных.

Если данные валидны и уникальны:

Хеширует пароль с использованием `werkzeug.security.generate\_password\_hash`.

Создает новую запись в таблице `Account` с логином, хешем пароля и email.

Сохраняет изменения в БД (`db.session.add`, `db.session.commit`).

Сохраняет `id` и `login` нового пользователя в сессии (`session['user\_id']`, `session['user\_login']`).

Отправляет клиенту редирект на главную страницу с сообщением об успехе (`flash`).

Если данные невалидны или не уникальны:

Отправляет клиенту ту же страницу регистрации с сообщением об ошибке (`flash`).

2. Авторизация пользователя:

1. Клиент: Пользователь заполняет форму входа (логин/email, пароль) на странице `/login` и отправляет ее (POST-запрос).

2. Сервер (Flask):

Получает POST-запрос.

Находит пользователя в таблице `Account` по логину или email.

Если пользователь найден:

Сравнивает введенный пароль с хешем в БД с использованием `werkzeug.security.check\_password\_hash`.

Если пароль совпадает:

Сохраняет `id` и `login` пользователя в сессии.

Отправляет клиенту редирект на главную страницу (или в профиль) с сообщением об успехе.

Если пароль не совпадает:

Отправляет клиенту страницу входа с сообщением об ошибке.

Если пользователь не найден:

Отправляет клиенту страницу входа с сообщением об ошибке.

3. Просмотр каталога игр:

1. Клиент: Пользователь открывает главную страницу (`/`, GET-запрос). Может выбрать теги/жанры для фильтрации (параметры в URL, например, `/?genre=Экшен&genre=RPG`).

2. Сервер (Flask):

Получает GET-запрос.

Извлекает все игры из таблицы `Database`.

Если в запросе есть параметры `genre`:

Фильтрует список игр, оставляя только те, у которых хотя бы один из тегов (`teg1`, `teg2`, `teg3`) совпадает с выбранными жанрами (регистронезависимое сравнение).

Передает отфильтрованный (или полный) список игр в шаблон `glavna.html`.

Отрисовывает шаблон и отправляет HTML-страницу клиенту.

4. Добавление новой игры (авторизованным пользователем):

1. Клиент: Пользователь (предполагается, что он авторизован) заполняет форму добавления игры (название, описание, теги, выбирает файл изображения, выбирает файл игры) и отправляет ее (POST-запрос на `/add\_game`).

2. Сервер (Flask):

Проверяет, авторизован ли пользователь (наличие `user\_id` в сессии).

Получает POST-запрос с данными формы и файлами.

Валидирует текстовые поля.

Обрабатывает загруженный файл изображения:

Проверяет наличие файла и его расширение (`allowed\_image\_file`).

Генерирует уникальное имя файла (например, с timestamp).

Сохраняет файл в папку `static/uploads/`.

Сохраняет имя файла для записи в БД.

Обрабатывает загруженный файл игры (аналогично изображению, но с `allowed\_archive\_file`).

Создает новую запись в таблице `Database`, связывая ее с `user\_id` текущего пользователя.

Сохраняет изменения в БД.

Отправляет клиенту редирект (например, на главную страницу) с сообщением об успехе.

В случае ошибок валидации или сохранения - возвращает форму с сообщением об ошибке.

5. Скачивание файла игры:

1. Клиент: Пользователь на странице игры (`/karta1/<id>`) нажимает кнопку "Скачать" (отправляется POST-запрос на `/download\_game/<id>`).

2. Сервер (Flask):

Получает POST-запрос.

Находит игру по `id` в таблице `Database`.

Проверяет наличие имени файла игры (`game\_file`) в записи БД.

Проверяет физическое существование файла в `static/uploads/`.

Если файл найден:

Использует `send\_from\_directory` для отправки файла клиенту как вложения (`as\_attachment=True`) с понятным именем.

Если файл не найден или запись о нем отсутствует - возвращает редирект на страницу игры с сообщением об ошибке.

1.3. Обоснование выбора средств реализации

Выбор технологического стека был продиктован требованиями задачи, необходимостью быстрой разработки прототипа и распространенностью технологий.

Backend (Python / Flask):

Python: Выбран как основной язык программирования благодаря своей простоте, читаемости, большому количеству библиотек и активному сообществу. Отлично подходит для веб-разработки.

Flask: Микрофреймворк Flask был выбран вместо более комплексных фреймворков (как Django) из-за его гибкости, минимализма и простоты освоения. Он предоставляет базовые инструменты для маршрутизации, обработки запросов, работы с шаблонами и сессиями, позволяя при этом легко подключать необходимые расширения по мере необходимости (например, SQLAlchemy, Flask-Login - хотя последний не используется явно в текущем коде `app.py`, но мог бы быть интегрирован). Flask хорошо подходит для создания прототипов и небольших/средних веб-приложений.

SQLAlchemy: Мощная и популярная ORM (Object-Relational Mapper) для Python. Позволяет работать с базой данных, используя объекты Python, а не писать SQL-запросы вручную (хотя и это возможно). Упрощает миграцию между различными СУБД (хотя в данном проекте используется только SQLite). Предоставляет удобные инструменты для определения моделей данных (`models.py`) и выполнения запросов.

SQLite: Легковесная файловая СУБД. Идеально подходит для разработки и прототипирования, так как не требует отдельного сервера баз данных. Вся база хранится в одном файле (`instance/site.db`). Для продакшн-среды с высокой нагрузкой, возможно, стоило бы перейти на PostgreSQL или MySQL, что легко сделать благодаря SQLAlchemy.

Werkzeug: Библиотека, лежащая в основе Flask, предоставляет важные утилиты, включая безопасное хеширование паролей (`generate\_password\_hash`, `check\_password\_hash`) и обработку файловых загрузок (`secure\_filename`).

Frontend (HTML / CSS / JavaScript / Jinja2):

HTML: Стандартный язык разметки для создания структуры веб-страниц.

CSS: Используется для стилизации и визуального оформления страниц (`static/главня.css` и др.).

JavaScript: Применяется для добавления интерактивности на стороне клиента (например, валидация форм, асинхронная проверка доступности логина/email в `glavna.html`, возможно, другие элементы интерфейса).

Jinja2: Мощный шаблонизатор для Python, интегрированный во Flask. Позволяет создавать динамические HTML-страницы, вставляя данные с сервера в шаблоны (`templates/`), использовать циклы, условия, наследование шаблонов (`base.html`).

Другие инструменты:

Git: Система контроля версий для управления исходным кодом.

pip и `requirements.txt`: Менеджер пакетов Python для управления зависимостями проекта.

Этот стек технологий является сбалансированным решением для создания веб-приложения данного типа, обеспечивая достаточную производительность, гибкость и скорость разработки прототипа.

1.4. Описание среды выполнения

Приложение разработано для запуска в стандартной среде выполнения Python и веб-браузере.

Требования к серверной части (Backend):

Операционная система: Кроссплатформенное приложение, может быть запущено на Windows, macOS, Linux. Разработка и первичное тестирование проводились в ОС Windows.

Интерпретатор Python: Версия 3.6 или выше (рекомендуется 3.8+).

Зависимости Python: Необходимо установить пакеты, перечисленные в файле `requirements.txt`. Основные зависимости:

`Flask`: веб-фреймворк.

`Flask-SQLAlchemy`: интеграция SQLAlchemy с Flask.

`SQLAlchemy`: ORM для работы с БД.

`Werkzeug`: утилиты для веб-сервера (включая хеширование).

(Другие зависимости, если они есть в `requirements.txt`)

Установка выполняется командой: `pip install -r requirements.txt`

Доступ к сети: Не требуется для базовой работы (кроме скачивания зависимостей), так как нет интеграции с внешними API.

Требования к клиентской части (Frontend):

Веб-браузер: Любой современный веб-браузер с поддержкой HTML5, CSS3 и JavaScript (например, Google Chrome, Mozilla Firefox, Safari, Microsoft Edge).

Развертывание:

Для разработки и локального запуска используется встроенный сервер разработки Flask (`flask run`). Для продуктивного развертывания рекомендуется использовать более надежный WSGI-сервер (например, Gunicorn или uWSGI) в связке с веб-сервером (например, Nginx или Apache).

1.5. Описание структуры Базы Данных

Для хранения данных приложения используется реляционная база данных SQLite. Взаимодействие с БД осуществляется через SQLAlchemy ORM. Структура базы данных определяется двумя основными моделями (таблицами), описанными в файле `models.py`:

1. `Account` (Таблица `account`): Хранит информацию о зарегистрированных пользователях.

`id` (Integer, Primary Key): Уникальный идентификатор пользователя. Автоматически генерируется.

`login` (String(20), Not Null, Unique): Логин пользователя. Ограничен 20 символами, должен быть уникальным и не может быть пустым.

`password` (String(128), Not Null): Хеш пароля пользователя. Хранится хеш (сгенерированный `generate\_password\_hash` с методом `pbkdf2:sha256`), а не сам пароль. Длина 128 позволяет хранить достаточно длинные хеши. Не может быть пустым.

`pochta` (String(50), Not Null, Unique): Адрес электронной почты пользователя. Ограничен 50 символами, должен быть уникальным и не может быть пустым.

Методы модели `Account`:

`\_\_init\_\_(self, login, password, pochta)`: Конструктор. Принимает логин, пароль и почту. Пароль автоматически хешируется при создании объекта.

`check\_password(self, password)`: Метод для проверки введенного пароля. Сравнивает хеш введенного пароля с хешем, хранящимся в БД, используя `check\_password\_hash`. Возвращает `True`, если пароли совпадают, иначе `False`.

`\_\_repr\_\_(self)`: Возвращает строковое представление объекта для отладки.

2. `Database` (Таблица `database`): Хранит информацию об играх, добавленных на платформу.

`id` (Integer, Primary Key): Уникальный идентификатор игры. Автоматически генерируется.

`name` (String(100), Not Null): Название игры. Ограничено 100 символами, не может быть пустым.

`opis` (Text, Not Null): Описание игры. Текстовое поле без ограничения длины (в SQLite), не может быть пустым.

`teg1` (String(50)): Первый тег/жанр игры. Ограничен 50 символами. Может быть пустым.

`teg2` (String(50)): Второй тег/жанр игры. Ограничен 50 символами. Может быть пустым.

`teg3` (String(50)): Третий тег/жанр игры. Ограничен 50 символами. Может быть пустым.

`image` (String(255)): Имя файла изображения-обложки игры. Хранится только имя файла, путь формируется на сервере (`static/uploads/`). Ограничено 255 символами. Может быть пустым.

`game\_file` (String(255)): Имя файла архива игры. Хранится только имя файла. Ограничено 255 символами. Может быть пустым.

`user\_id` (Integer, ForeignKey('account.id')): Внешний ключ, связывающий игру с пользователем (автором), который ее добавил. Ссылается на поле `id` таблицы `account`. Может быть пустым (если связь не установлена или не требуется).

`login` (String(80)): Поле выглядит избыточным, так как есть `user\_id`. Возможно, используется для отображения логина автора напрямую?

`password` (String(255)): Это поле выглядит очень странно и небезопасно в таблице игр. Хранение пароля (даже хешированного) здесь не имеет смысла. Вероятно, это остаток от копирования или недоразумение. В конструкторе оно хешируется, но его назначение неясно. Рекомендуется удалить это поле.

`created\_at` (DateTime, Default: `datetime.utcnow`): Дата и время добавления записи об игре в базу данных. Устанавливается автоматически при создании записи.

Методы модели `Database`:

`\_\_init\_\_(...)`: Конструктор. Принимает основные поля игры. Хеширует поле `password`, если оно передано, что подтверждает его неуместность здесь.

`\_\_repr\_\_(self)`: Возвращает строковое представление объекта для отладки.

Связи между таблицами:

Установлена связь один-ко-многим между `Account` и `Database` через поле `Database.user\_id`, ссылающееся на `Account.id`. Это означает, что один пользователь (`Account`) может добавить несколько игр (`Database`), но каждая игра добавляется одним конкретным пользователем.

(ER-диаграмма - Текстовое представление)

\*\*(ER-диаграмма - Текстовое представление)\*\*

```

+-------------+ +-----------------+

| Account | | Database |

+-------------+ +-----------------+

| PK id |---<--| FK user\_id |

| login | | PK id |

| password | | name |

| pochta | | opis |

+-------------+ | teg1 |

| teg2 |

| teg3 |

| image |

| game\_file |

| login (?) |

| password (X) |

| created\_at |

```

(PK = Primary Key, FK = Foreign Key)

(?) - Поле сомнительной полезности

(X) - Поле, рекомендуемое к удалению

Эта структура данных позволяет эффективно хранить и извлекать информацию, необходимую для работы каталога игр.

2. ПРАКТИЧЕСКИЙ (РЕАЛИЗАЦИОННЫЙ) РАЗДЕЛ

2.1. Описание реализации программного кода

2.1.1. Структура проекта

Проект имеет следующую основную структуру директорий и файлов:

```

isdj-1/

│

├── app.py Основной файл приложения Flask (backend)

├── models.py Модели данных SQLAlchemy (описание таблиц БД)

├── requirements.txt Список зависимостей Python

├── .gitignore Файл для исключения ненужных файлов из Git

├── README.md Описание проекта (техническое)

├── Методичка по курсовой.md Методические указания

├── Poyasnitelnaya\_Zapiska.md Данный документ

│

├── instance/ Директория для файлов экземпляра Flask

│ └── site.db Файл базы данных SQLite

│

├── static/ Директория для статических файлов

│ ├── glavnya.css Основные стили

│ ├── karta1.css Стили для страницы игры

│ ├── Profile.css Стили для страницы профиля

│ ├── главная.js Основной JavaScript

│ └── uploads/ Директория для загруженных файлов (изображения, игры)

│ ├── ... (файлы)

│

└── templates/ Директория для HTML-шаблонов Jinja2

├── base.html Базовый шаблон

├── glavna.html Шаблон главной страницы (каталог)

├── karta1.html Шаблон страницы детального просмотра игры

├── login.html Шаблон страницы авторизации

├── register.html Шаблон страницы регистрации

└── Profile.html Шаблон страницы профиля пользователя

```

Такая структура является стандартной для многих проектов на Flask и разделяет логику (Python), данные (SQLite), статические ресурсы (CSS, JS, изображения) и представление (HTML-шаблоны).

2.1.2. Серверная часть (Backend - app.py)

Файл `app.py` содержит основную логику серверной части приложения.

Инициализация и конфигурация:

Импортируются необходимые модули Flask, SQLAlchemy, Werkzeug, os, datetime и др.

Создается экземпляр приложения Flask: `app = Flask(\_\_name\_\_)`.

Задаются конфигурационные параметры:

`SQLALCHEMY\_DATABASE\_URI`: Путь к файлу базы данных SQLite (`sqlite:///instance/site.db`). Директория `instance/` создается автоматически, если не существует.

`SECRET\_KEY`: Секретный ключ для подписи сессионных cookie. Генерируется случайно (`os.urandom(24).hex()`) или берется из переменной окружения для безопасности.

`SQLALCHEMY\_TRACK\_MODIFICATIONS`: Отключается для экономии ресурсов.

`UPLOAD\_FOLDER`: Путь к папке для загрузки файлов (`static/uploads/`). Папка создается автоматически.

Определяются разрешенные расширения для изображений (`ALLOWED\_IMAGE\_EXTENSIONS`) и архивов (`ALLOWED\_ARCHIVE\_EXTENSIONS`).

Инициализируется расширение SQLAlchemy: `db.init\_app(app)`.

В контексте приложения создаются таблицы базы данных, если они еще не существуют: `db.create\_all()`.

Вспомогательные функции:

`allowed\_image\_file(filename)`: Проверяет, имеет ли имя файла разрешенное расширение для изображений.

`allowed\_archive\_file(filename)`: Проверяет, имеет ли имя файла разрешенное расширение для архивов.

Маршруты (Routes) и функции-обработчики:

`@app.route('/', methods=['GET', 'POST']) def glavna():`

Обрабатывает главную страницу.

POST: Логика регистрации нового пользователя (получение данных из формы, проверка уникальности, хеширование пароля, создание записи в `Account`, сохранение в сессии, вывод flash-сообщений). Примечание: Размещение регистрации на главной странице может быть не лучшим UX-решением; обычно для этого есть отдельная страница `/register`. Код регистрации дублируется в маршруте `/register`.

GET: Логика отображения каталога игр. Получает все игры из `Database`. Применяет фильтрацию по жанрам (`teg1`, `teg2`, `teg3`), если в URL переданы параметры `genre`. Рендерит шаблон `glavna.html`, передавая список игр и статус авторизации пользователя.

`@app.route('/karta1/<int:id>') def karta1(id):`

Отображает страницу детального просмотра игры.

Получает игру из `Database` по её `id`. Если игра не найдена, возвращает ошибку 404.

Рендерит шаблон `karta1.html`, передавая данные найденной игры.

`@app.route('/download\_game/<int:game\_id>', methods=['POST']) def download\_game(game\_id):`

Обрабатывает запрос на скачивание файла игры.

Находит игру по `id`.

Проверяет наличие `game\_file` в записи БД и существование файла на диске.

Если файл найден:

Использует `send\_from\_directory` для отправки файла клиенту как вложения (`as\_attachment=True`) с понятным именем.

Если файл не найден или запись о нем отсутствует - возвращает редирект на страницу игры с сообщением об ошибке.

`@app.route('/Profile/<int:user\_id>') def prof(user\_id):`

Назначение этого маршрута неясно. Он пытается получить игру (`Database.query.get(user\_id)`) по `user\_id`, что нелогично. Вероятно, он должен был бы получать данные пользователя (`Account.query.get(user\_id)`) или игры, добавленные этим пользователем. В текущем виде он, скорее всего, не работает корректно. Требует исправления. Есть также маршрут `@app.route('/profile/<int:user\_id>')` ниже, который выглядит более правильно.

`@app.route('/check\_availability', methods=['POST']) def check\_availability():`

API-эндпоинт для асинхронной проверки доступности логина или email при регистрации (используется JavaScript'ом на клиенте).

Принимает JSON с `type` ('login' или 'email') и `value`.

Проверяет наличие записи в `Account` и возвращает JSON `{'available': True/False}`.

`@app.route('/add\_game', methods=['POST']) def add\_game():`

Обрабатывает добавление новой игры.

Получает данные из формы (`name`, `opis`, `teg1`, `teg2`, `teg3`).

Обрабатывает загрузку файла изображения (`request.files['image']`): проверка расширения, генерация уникального имени, сохранение в `UPLOAD\_FOLDER`, запись имени в БД.

Обрабатывает загрузку файла игры (`request.files['game\_file']`) аналогично изображению.

Создает новую запись в `Database`, связывая с `user\_id` из сессии.

Сохраняет в БД и редиректит с flash-сообщением. Обрабатывает возможные ошибки.

`@app.route('/search') def search():`

Реализует поиск игр по названию или описанию.

Получает поисковый запрос `q` из параметров URL.

Использует `or\_` из SQLAlchemy для поиска по полям `name` и `opis` (регистронезависимый поиск `ilike`).

Рендерит шаблон `glavna.html` с результатами поиска.

`@app.route('/register', methods=['GET', 'POST']) def register():`

Обрабатывает страницу регистрации.

GET: Отображает шаблон `register.html`.

POST: Логика регистрации, аналогичная той, что в `glavna()` (проверка данных, хеширование, создание `Account`, вход в сессию).

`@app.route('/login', methods=['GET', 'POST']) def login():`

Обрабатывает страницу авторизации.

GET: Отображает шаблон `login.html`.

POST: Логика авторизации (поиск пользователя по логину/email, проверка пароля с `check\_password\_hash`, сохранение в сессии).

`@app.route('/logout') def logout():`

Выход пользователя из системы.

Удаляет данные пользователя (`user\_id`, `user\_login`, `user\_logged\_in`) из сессии.

Редиректит на главную страницу с flash-сообщением.

`@app.route('/profile/<int:user\_id>') def profile(user\_id):`

Отображает профиль пользователя.

Находит пользователя (`Account`) по `user\_id`.

Находит игры, добавленные этим пользователем (`Database.query.filter\_by(user\_id=user\_id)`).

Рендерит шаблон `Profile.html`, передавая данные пользователя и список его игр.

`@app.route('/settings') def settings():`

Маршрут не реализован (содержит `pass`). Предполагалось, что здесь будут настройки пользователя.

`@app.route('/uploads/<path:filename>') def uploaded\_file(filename):`

Служебный маршрут для отдачи загруженных файлов напрямую из папки `uploads`. Позволяет отображать загруженные изображения в HTML (`<img src="{{ url\_for('uploaded\_file', filename=game.image) }}">`).

2.1.3. Модели данных (models.py)

Как было подробно описано в разделе [1.5. Описание структуры Базы Данных](15-описание-структуры-базы-данных), файл `models.py` определяет две SQLAlchemy модели: `Account` и `Database`, которые соответствуют таблицам `account` и `database` в SQLite. Ключевыми особенностями являются:

Использование `db.Column` для определения полей таблицы с типами данных и ограничениями (primary\_key, nullable, unique, ForeignKey).

Автоматическое хеширование пароля в конструкторе `Account` и метод `check\_password` для безопасной аутентификации.

Связь "один-ко-многим" между пользователем (`Account`) и его играми (`Database`) через `user\_id`.

Наличие потенциально избыточных (`login`) и небезопасных (`password`) полей в модели `Database`, требующих ревизии.

2.1.4. Клиентская часть (Frontend - templates, static)

Клиентская часть реализована с помощью:

HTML-шаблонов Jinja2 (в папке `templates/`):

`base.html`: Базовый шаблон, определяющий общую структуру страниц (шапка, подвал, подключение CSS/JS). Остальные шаблоны наследуют его (`{% extends 'base.html' %}`).

`glavna.html`: Главная страница с каталогом игр. Использует цикл `{% for item in glavna %}` для отображения карточек игр. Содержит форму регистрации/логина (или ссылки на них) и фильтры по жанрам.

`karta1.html`: Страница детального просмотра игры. Отображает полное описание, теги, изображение, кнопку скачивания.

`login.html`, `register.html`: Стандартные формы для входа и регистрации.

`Profile.html`: Страница профиля пользователя. Отображает информацию о пользователе и список добавленных им игр.

Шаблоны используют переменные, переданные из Flask (`{{ user\_logged\_in }}`, `{{ item.name }}`), и встроенные теги/фильтры Jinja2. `url\_for()` используется для генерации URL к маршрутам и статическим файлам. Flash-сообщения отображаются с помощью `{% with messages = get\_flashed\_messages() %}`.

CSS-файлов (в папке `static/`):

`главня.css`, `karta1.css`, `Profile.css`: Определяют внешний вид соответствующих страниц.

JavaScript-файлов (в папке `static/`):

`главная.js`: Используется, вероятно, для интерактивных элементов на главной странице, таких как асинхронная проверка доступности логина/почты (функция `checkAvailability`), возможно, для работы фильтров или других элементов UI.

2.1.5. Работа с файлами (загрузка изображений и игр)

Реализована в маршруте `/add\_game`:

1. Получение файла: Файлы извлекаются из объекта `request.files`.

2. Валидация: Проверяется наличие файла (`if file and file.filename:`). Затем расширение файла проверяется с помощью функций `allowed\_image\_file` или `allowed\_archive\_file`. При недопустимом формате выводится flash-сообщение.

3. Генерация имени: Для избежания конфликтов имен и проблем с безопасностью используется `secure\_filename()` для очистки имени файла от недопустимых символов. К имени добавляется временная метка (`int(time.time())`) для обеспечения уникальности.

4. Сохранение: Папка `UPLOAD\_FOLDER` (`static/uploads/`) создается, если не существует (`os.makedirs(..., exist\_ok=True)`). Файл сохраняется в эту папку с помощью метода `file.save(full\_path)`.

5. Запись в БД: В базу данных (`Database.image` или `Database.game\_file`) записывается только сгенерированное имя файла, а не полный путь.

Для отдачи файлов используется:

Маршрут `/uploads/<path:filename>` и функция `send\_from\_directory` для прямого доступа к файлам по URL (например, для тега `<img>`).

Маршрут `/download\_game/<int:game\_id>` и функция `send\_from\_directory` с параметром `as\_attachment=True` для принудительного скачивания файла игры.

2.1.6. Безопасность

В коде реализованы базовые меры безопасности:

Хеширование паролей: Пароли пользователей не хранятся в открытом виде, а хешируются с использованием `generate\_password\_hash` (с солью, метод `pbkdf2:sha256`). Проверка пароля осуществляется через `check\_password\_hash`.

Защита имен файлов: `secure\_filename` используется для предотвращения атак типа "Path Traversal" при загрузке файлов.

Уникальные имена файлов: Добавление timestamp к именам загружаемых файлов предотвращает перезапись файлов и возможные связанные с этим проблемы.

Управление сессиями: Flask использует подписанные cookie для хранения данных сессии (`SECRET\_KEY`), что защищает от подделки сессии на стороне клиента.

Проверка типов файлов: Ограничение на загрузку только разрешенных типов изображений и архивов снижает риск загрузки вредоносных файлов.

Обработка ошибок: Использование блоков `try...except` помогает перехватывать ошибки (например, при работе с БД или файлами) и предотвращать утечку чувствительной информации или падение сервера.

Потенциальные улучшения безопасности:

CSRF-защита: Рекомендуется добавить CSRF-токены во все формы, изменяющие состояние (регистрация, логин, добавление игры), например, с помощью расширения Flask-WTF или Flask-SeaSurf.

Валидация входных данных: Более строгая валидация всех данных, поступающих от пользователя (длина строк, формат email, содержимое полей), как на клиенте, так и на сервере.

Rate Limiting: Защита от перебора паролей и чрезмерной нагрузки на API (например, `/check\_availability`) с помощью ограничения частоты запросов.

Контроль доступа: Убедиться, что действия, требующие авторизации (например, добавление игры), строго проверяют наличие пользователя в сессии. В маршруте `/add\_game` такая проверка отсутствует, что является серьезной уязвимостью - любой может добавить игру. Необходимо добавить проверку `if 'user\_id' not in session: return redirect(url\_for('login'))`.

SQL Injection: Использование SQLAlchemy ORM обеспечивает хорошую защиту от SQL-инъекций, так как запросы параметризуются.

2.2. Описание процесса отладки

В ходе разработки и тестирования приложения могут возникать типовые проблемы:

1. Ошибки работы с БД:

Проблема: Ошибки подключения, неверные запросы, нарушение ограничений уникальности (unique constraints) при попытке добавить пользователя с существующим логином/email.

Отладка: Проверка строки подключения (`SQLALCHEMY\_DATABASE\_URI`), существования файла БД, правильности моделей (`models.py`), использование `try...except` блоков вокруг операций с `db.session` с `db.session.rollback()` в блоке `except` для отката транзакции при ошибке. Просмотр логов Flask и сообщений об ошибках SQLAlchemy.

2. Ошибки обработки форм:

Проблема: Неправильное получение данных из `request.form`, ошибки валидации, неверная обработка отсутствующих полей.

Отладка: Использование `request.form.get('field\_name')` для безопасного получения данных (возвращает `None`, если поля нет). Вывод полученных данных в консоль (`print()`) или использование дебаггера Flask для проверки содержимого `request.form`. Проверка логики валидации.

3. Ошибки загрузки файлов:

Проблема: Неправильная проверка типа файла, ошибки при сохранении файла (недостаточно прав доступа к папке `uploads/`, неверный путь), ошибки при генерации имени файла.

Отладка: Проверка корректности работы функций `allowed\_...\_file`, проверка прав доступа к директории `UPLOAD\_FOLDER`, вывод полных путей и имен файлов в лог, проверка наличия файла после `file.save()`.

4. Ошибки маршрутизации и шаблонов:

Проблема: Ошибки 404 (Not Found) из-за неправильных URL в `url\_for()` или опечаток в декораторах `@app.route`. Ошибки рендеринга шаблонов Jinja2 (неверные имена переменных, синтаксические ошибки в шаблоне).

5. Проблемы с сессиями:

Проблема: Пользователь "разлогинивается" после закрытия браузера (если сессия не постоянная), данные сессии не сохраняются или не читаются.

Отладка: Проверка наличия и правильности установки `SECRET\_KEY`. Использование инструментов разработчика в браузере для просмотра cookie сессии. Проверка логики записи (`session['key'] = value`) и чтения (`session.get('key')`) данных сессии.

6. Ошибки на стороне клиента (JavaScript):

Проблема: Не работает асинхронная проверка логина, ошибки в консоли браузера.

Отладка: Использование инструментов разработчика в браузере (вкладка "Консоль" для просмотра ошибок, вкладка "Сеть" для анализа AJAX-запросов к `/check\_availability`).

Отладка проводилась с использованием:

Встроенного дебаггера Flask (запуск с `flask run --debug`).

Вывода отладочной информации в консоль (`print()`).

Инструментов разработчика веб-браузера.

2.3. Характеристика программы

Структура: Приложение имеет четкое разделение на серверную часть (`app.py`, `models.py`) и клиентскую часть (`templates/`, `static/`), что упрощает разработку и поддержку. Используется монолитная структура бэкенда (вся логика в `app.py`), что приемлемо для прототипа, но для более крупных проектов рекомендуется разбиение на модули (blueprints).

Взаимодействие модулей:

Клиент (браузер) взаимодействует с сервером через HTTP-запросы.

Сервер (`app.py`) обрабатывает запросы, взаимодействует с моделями (`models.py`) для доступа к данным в БД (SQLite).

Модели (`models.py`) используют SQLAlchemy для абстрагирования работы с БД.

Сервер использует Jinja2 для рендеринга HTML-шаблонов (`templates/`) и отправки их клиенту.

Статические файлы (CSS, JS, загруженные файлы) отдаются напрямую из папки `static/`.

Объем оперативной памяти:

Бэкенд (Flask): Потребление памяти относительно невелико для самого процесса Flask. Зависит от количества одновременно обрабатываемых запросов и объема данных, извлекаемых из БД за один раз. Загрузка файлов может временно увеличивать потребление памяти во время сохранения.

Фронтенд (Браузер): Зависит от сложности отображаемых страниц, количества игр в каталоге, размера изображений. В целом, ожидается умеренное потребление памяти.

Время работы программы (отклик):

Время ответа на GET-запросы (просмотр каталога, страниц игр) зависит в основном от скорости выполнения запросов к БД SQLite и времени рендеринга шаблонов. Для небольшого количества данных SQLite работает очень быстро.

Время ответа на POST-запросы (регистрация, логин, добавление игры) включает время на валидацию, операции с БД (вставка, обновление) и, в случае добавления игры, время на сохранение загруженных файлов на диск.

Загрузка и сохранение файлов (особенно больших архивов игр) может быть наиболее длительной операцией.

В целом, для прототипа с небольшим количеством данных ожидается быстрый отклик. С ростом объема данных и количества пользователей может потребоваться оптимизация запросов к БД или переход на более производительную СУБД.

Масштабируемость:

Вертикальная масштабируемость (увеличение ресурсов сервера) возможна.

Горизонтальная масштабируемость (запуск нескольких экземпляров приложения) затруднена из-за использования SQLite (требуется общая файловая система или переход на серверную СУБД).

Для серьезного масштабирования потребуется переход на другую СУБД (PostgreSQL, MySQL) и использование WSGI-сервера (Gunicorn) за проксирующим веб-сервером (Nginx).

2.4. Контрольный пример

Рассмотрим сценарий добавления новой игры пользователем.

Предусловия:

Пользователь "dev1" зарегистрирован и авторизован в системе.

Сервер Flask запущен.

Шаги:

1. Пользователь (Клиент): Открывает главную страницу (`/`). Видит кнопку/ссылку "Добавить игру". Переходит по ней (или форма добавления находится прямо на главной).

2. Пользователь (Клиент): Заполняет форму добавления игры:

Название: "Pixel Quest"

Описание: "Простой платформер в пиксельном стиле."

Тег1: "Платформер"

Тег2: "Пиксельная графика"

Тег3: "Инди"

Изображение: Выбирает файл `pixel\_quest\_cover.png`.

Файл игры: Выбирает файл `PixelQuest\_v1.zip`.

Нажимает кнопку "Добавить". (Отправляется POST-запрос на `/add\_game`).

3. Сервер (Backend - `/add\_game`):

Получает POST-запрос.

Проверяет, что пользователь авторизован (находит `user\_id` в `session`).

Извлекает текстовые данные: `name`, `opis`, `teg1`, `teg2`, `teg3`.

Получает файл изображения `pixel\_quest\_cover.png` из `request.files['image']`.

Проверяет расширение `.png` (допустимо).

Генерирует уникальное имя, например `1678886400\_pixel\_quest\_cover.png`.

Сохраняет файл по пути `static/uploads/1678886400\_pixel\_quest\_cover.png`.

Получает файл игры `PixelQuest\_v1.zip` из `request.files['game\_file']`.

Проверяет расширение `.zip` (допустимо).

Генерирует уникальное имя, например `game\_1678886400\_PixelQuest\_v1.zip`.

Сохраняет файл по пути `static/uploads/game\_1678886400\_PixelQuest\_v1.zip`.

Создает новый объект `Database`:

```python

new\_game = Database(

name="Pixel Quest",

opis="Простой платформер в пиксельном стиле.",

teg1="Платформер",

teg2="Пиксельная графика",

teg3="Инди",

image="1678886400\_pixel\_quest\_cover.png",

game\_file="game\_1678886400\_PixelQuest\_v1.zip",

user\_id=session['user\_id'] ID пользователя "dev1"

)

```

Добавляет объект в сессию SQLAlchemy: `db.session.add(new\_game)`.

Сохраняет изменения в БД: `db.session.commit()`.

Создает flash-сообщение: `flash('Игра успешно добавлена!', 'success')`.

Возвращает клиенту редирект на главную страницу: `redirect(url\_for('glavna'))`.

4. Пользователь (Клиент): Браузер перенаправляется на главную страницу (`/`).

5. Сервер (Backend - `/`):

Обрабатывает GET-запрос к `/`.

Извлекает все игры из БД, включая только что добавленную "Pixel Quest".

Рендерит шаблон `glavna.html`, передавая обновленный список игр.

6. Пользователь (Клиент): Видит главную страницу с обновленным каталогом, где появилась карточка игры "Pixel Quest" с её изображением и тегами. Видит flash-сообщение "Игра успешно добавлена!".

Ожидаемый результат:

В базе данных (таблица `database`) появилась новая запись с данными игры "Pixel Quest".

Файлы `1678886400\_pixel\_quest\_cover.png` и `game\_1678886400\_PixelQuest\_v1.zip` сохранены в папке `static/uploads/`.

Пользователь видит новую игру в каталоге на главной странице.

2.5. Инструкция пользователя

2.5.1. Системные требования

Сервер:

Интерпретатор Python 3.6+

Установленные зависимости из `requirements.txt`

Доступ на запись к директории проекта (для создания `instance/site.db` и `static/uploads/`)

Клиент:

Современный веб-браузер (Chrome, Firefox, Edge, Safari)

2.5.2. Установка и запуск (для разработчика/локально)

1. Клонируйте репозиторий:

```bash

git clone <URL репозитория>

cd isdj-1

```

2. Создайте и активируйте виртуальное окружение (рекомендуется):

```bash

python -m venv .venv

Windows

.venv\Scripts\activate

macOS/Linux

source .venv/bin/activate

```

3. Установите зависимости:

```bash

pip install -r requirements.txt

```

4. Запустите приложение:

```bash

flask run

```

Сервер будет запущен (обычно по адресу http://127.0.0.1:5000/). При первом запуске будет создан файл базы данных `instance/site.db` и папка `static/uploads/`.

5. Откройте приложение в браузере: Перейдите по адресу, указанному в консоли (например, http://127.0.0.1:5000/).

2.5.3. Использование интерфейса

1. Просмотр каталога:

Откройте главную страницу. Вы увидите список доступных инди-игр.

Используйте флажки с жанрами/тегами (если они реализованы как фильтры) для отбора игр по интересующим категориям.

2. Просмотр игры:

Нажмите на карточку игры в каталоге, чтобы перейти на страницу с детальным описанием, тегами и изображением.

Если автор прикрепил файл игры, на странице будет кнопка "Скачать". Нажмите ее, чтобы загрузить архив с игрой.

3. Регистрация:

Найдите форму регистрации на главной странице или перейдите по ссылке "Регистрация" (если она есть).

Введите желаемый логин, действующий email и надежный пароль.

Нажмите кнопку "Зарегистрироваться". При успешной регистрации вы будете автоматически авторизованы.

4. Авторизация (Вход):

Найдите форму входа или перейдите по ссылке "Вход".

Введите ваш логин (или email) и пароль.

Нажмите кнопку "Войти".

5. Добавление игры (для авторизованных пользователей):

Найдите кнопку/ссылку "Добавить игру".

Заполните поля: название, подробное описание, до трех тегов (жанров).

Загрузите изображение-обложку (форматы: png, jpg, jpeg, gif).

При необходимости загрузите архив с игрой (форматы: zip, rar, 7z, tar, gz).

Нажмите кнопку "Добавить". Ваша игра появится в общем каталоге.

6. Профиль:

Перейдите по ссылке "Профиль" (обычно доступна после авторизации).

Вы увидите информацию о своем аккаунте и список игр, которые вы добавили.

7. Выход:

Нажмите на ссылку "Выход", чтобы завершить сеанс авторизации.

2.5.4. Остановка приложения (локально)

Вернитесь в терминал, где был запущен `flask run`.

Нажмите `Ctrl + C`.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была успешно решена поставленная задача: разработана веб-платформа "Каталог инди-игр", предназначенная для поддержки и продвижения начинающих разработчиков.

Основные результаты работы:

1. Проведен анализ существующих платформ и выявлена ниша для специализированного каталога инди-игр, ориентированного на нужды начинающих разработчиков.

2. Спроектирована и реализована клиент-серверная архитектура приложения с использованием фреймворка Flask (Python) для бэкенда, SQLAlchemy ORM для работы с базой данных SQLite, и стандартных веб-технологий (HTML, CSS, JS, Jinja2) для фронтенда.

3. Реализован основной функционал:

Регистрация и авторизация пользователей с безопасным хранением паролей.

Просмотр каталога игр с возможностью фильтрации по тегам.

Детальный просмотр информации об отдельной игре.

Добавление новых игр авторизованными пользователями, включая загрузку изображений-обложек и файлов игр (архивов).

Возможность скачивания прикрепленных файлов игр.

Базовый функционал профиля пользователя со списком его игр.

Поиск игр по названию и описанию.

4. Создана структура базы данных, позволяющая хранить информацию о пользователях и играх, включая необходимые связи.

5. Обеспечены базовые меры безопасности, такие как хеширование паролей и обработка загружаемых файлов.

Разработанная платформа достигает поставленной цели, предоставляя прототип инструмента, который может помочь начинающим инди-разработчикам повысить видимость своих проектов и получить потенциальную площадку для взаимодействия с аудиторией. Платформа также полезна для игроков, ищущих новые и оригинальные инди-игры.

Практическая значимость проекта заключается в создании рабочего прототипа, который может быть развернут и использован по назначению, а также служить основой для дальнейшего развития. Код проекта демонстрирует применение современных веб-технологий и может быть полезен в образовательных целях.

Возможные направления для дальнейшего развития:

Система комментариев и оценок: Добавление возможности для пользователей оставлять отзывы и ставить оценки играм.

Улучшение профилей: Расширение функционала профилей пользователей и разработчиков (аватары, описания, ссылки на соцсети).

Более сложная фильтрация и сортировка: Добавление сортировки по дате, популярности, рейтингу; фильтрация по нескольким критериям одновременно.

Система рекомендаций: Внедрение алгоритмов (например, на основе тегов или поведения пользователей) для предложения релевантных игр.

Улучшение UI/UX: Дальнейшая проработка дизайна и удобства использования интерфейса.

Административная панель: Создание интерфейса для администрирования пользователей и контента.

Модульное тестирование и CI/CD: Внедрение автоматизированного тестирования и процессов непрерывной интеграции/доставки.

Переход на более производительную СУБД: Миграция с SQLite на PostgreSQL или MySQL для лучшей масштабируемости.

Усиление безопасности: Реализация CSRF-защиты, более строгой валидации, rate limiting.

В целом, курсовая работа успешно завершена, цель достигнута, задачи выполнены. Создан функциональный прототип веб-платформы, имеющий практическую ценность и потенциал для дальнейшего развития.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

(Здесь должен быть список литературы, оформленный по ГОСТ или требованиям вашего ВУЗа. Укажите книги, статьи, документацию Flask, SQLAlchemy, веб-ресурсы, которые вы использовали.)

Документация Flask - [https://flask.palletsprojects.com/](https://flask.palletsprojects.com/)

Документация SQLAlchemy - [https://www.sqlalchemy.org/](https://www.sqlalchemy.org/)

[https://jinja.palletsprojects.com/](https://jinja.palletsprojects.com/)

Гринберг М. Разработка веб-приложений с использованием Flask на языке Python. - М.: ДМК Пресс, 2018.

(Другие книги по Python, веб-разработке)

(Статьи или блоги по теме инди-разработки, если ссылались)