МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

РОСТОВСКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ (РИНХ)

**Факультет Компьютерных технологий и информационной безопасности**

**Кафедра Информационных технологий и программирования**

**Индивидуальная работа**

**по дисциплине**

**«Инженерное программирование»**

на тему:

**«Разработка утилиты на Python для загрузки и хранения видео через прокси-сервер»**

Выполнил:

студент гр. ПРИ-311

Направление 09.03.04 Программная инженерия Д. В. Маршалкин

Руководитель:

доцент Е. В. Жилина

Дата сдачи работы:

Дата защиты:

Оценка:

Ростов-на-Дону

2025

### ****Введение****

В современном цифровом пространстве передача, хранение и обработка мультимедийного контента, в частности видеоматериалов, становятся неотъемлемой частью информационных технологий. Видеофайлы занимают значительный объем в общем интернет-трафике, а их доступность и качество воспроизведения напрямую зависят от используемых методов загрузки, хранения и передачи данных.

С увеличением количества пользователей, работающих с видеоконтентом, возникает потребность в разработке специализированных инструментов, позволяющих эффективно загружать, сохранять и воспроизводить видеоматериалы. Это особенно актуально для случаев, когда пользователи хотят управлять своими видеофайлами, минимизируя зависимость от сторонних платформ, а также обеспечивать доступность контента для последующего просмотра.

В данном индивидуальном проекте рассматривается разработка программного решения на языке Python, которое позволит загружать видеоролики через прокси-сервер, сохранять их на выделенном сервере и обеспечивать пользователям доступ к загруженному контенту. Архитектура проекта включает в себя два сервера:

**- Первый сервер** выполняет роль посредника — он подключается к видеохостингам, загружает файлы и передает их на второй сервер.

**- Второй сервер** отвечает за хранение видеофайлов и предоставление удобного интерфейса для их последующего просмотра.

Такой подход обеспечивает гибкость и надежность работы с видеоконтентом, минимизируя нагрузку на клиентские устройства, упрощая процесс управления файлами и повышая удобство использования системы.

Проект ориентирован на студентов и преподавателей, занимающихся изучением сетевых технологий, распределенных систем и мультимедийных данных. Он также демонстрирует применение языка Python в области автоматизированной обработки видеоконтента, что делает его полезным не только в образовательных, но и в практических целях.

**Актуальность тематики исследования**

В современном мире видеоконтент играет ключевую роль в различных сферах деятельности, включая образование, науку, медиаиндустрию и информационные технологии. С увеличением объема видеоматериалов и их повсеместным распространением пользователи сталкиваются с рядом сложностей, связанных с загрузкой, хранением и воспроизведением видеофайлов.

- Во-первых, видеофайлы требуют значительных вычислительных ресурсов и стабильного интернет-соединения. Низкая пропускная способность сети, ограничения со стороны интернет-провайдеров, а также географические или организационные блокировки доступа к видеохостингам создают серьезные препятствия для пользователей.

- Во-вторых, централизованное хранение видеоконтента на сторонних платформах приводит к зависимости от их политики, технических ограничений и возможных сбоев. Это особенно актуально для образовательных и исследовательских проектов, где необходима надежная и контролируемая система хранения видеоданных.

- В-третьих, отсутствие удобных инструментов для автоматизированной загрузки и управления видеоконтентом усложняет процесс работы с большими объемами мультимедийных данных.

Такое решение позволит:

- Оптимизировать процесс загрузки видео за счет использования промежуточного сервера, снижая нагрузку на клиентские устройства.

- Обеспечить удобное хранение и доступ к видеофайлам без риска их удаления или блокировки.

- Гибко адаптировать систему под различные сценарии использования, включая образовательные, архивные и исследовательские задачи.

Таким образом, данное исследование направлено на решение одной из актуальных проблем информационных технологий, связанных с эффективным управлением видеоконтентом. Разработанная утилита может быть востребована как в образовательных учреждениях, так и в других сферах, где необходим доступ к видеоинформации в условиях ограниченного интернет-доступа или блокировок.

### ****Технологический стек проекта и теоретическое обоснование****

Разрабатываемый проект предназначен для загрузки видео через прокси-сервер и их последующего хранения на выделенном сервере. Для успешной реализации системы был выбран стек технологий, обеспечивающий надежность, масштабируемость и удобство работы с мультимедийными данными.

## **1. Теоретическая часть**

### ****1.1. Предметная область****

В современном цифровом мире видеоконтент играет ключевую роль в различных сферах: медиа, образовании, науке, маркетинге и развлечениях. Однако пользователи сталкиваются с рядом проблем:

**- Ограниченный доступ** – некоторые видеохостинги могут блокироваться в определенных регионах.

**- Высокая нагрузка на клиентские устройства** – загрузка и хранение видеофайлов требуют значительных ресурсов.

**- Проблемы с устойчивостью соединения** – нестабильные сети могут приводить к прерыванию загрузки.

Для решения этих задач разрабатываются инструменты, которые позволяют загружать, хранить и управлять видеоконтентом через прокси-серверы и удаленные хранилища.

### ****1.2. Анализ существующих решений****

На данный момент существует несколько категорий программных решений для работы с видеоконтентом. На рисунке 1 сравнение и **анализ существующих решений.**

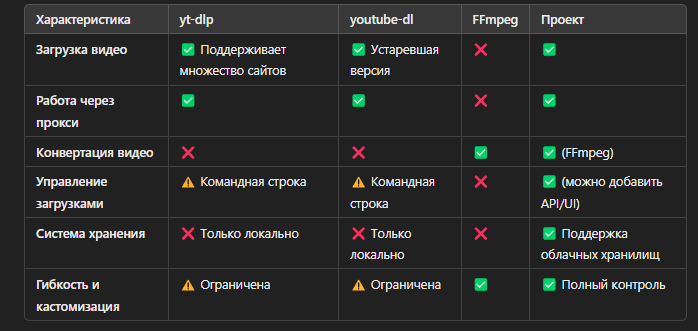


Рисунок 1 – сравнение и **анализ существующих решений**

Вывод:

- yt-dlp и youtube-dl удобны для разовых загрузок, но не предназначены для хранения и автоматизации.

- FFmpeg используется для обработки видео, но не поддерживает скачивание.

- Разрабатываемый проект объединяет возможности загрузки, прокси, хранения и обработки видео в одном решении.

**2. Технологический стек проекта**

### ****2.1. Язык программирования – Python****

Python был выбран из-за его гибкости и большого количества библиотек:

- requests**,** aiohttp – работа с HTTP-запросами и прокси.

- yt-dlp – загрузка видео с видеохостингов.

- ffmpeg-python – обработка и конвертация видеофайлов.

- paramiko – работа с удалёнными серверами по SSH.

### ****2.2. Серверная инфраструктура****

Используются два сервера:

1. **Основной сервер (Linux)** – для хранения и обработки видео.
2. **Второй сервер (TBD)** – в зависимости от задач (облако, база данных и т. д.).

### ****2.3. Система хранения видео****

**- Локальная файловая система** – для быстрого доступа.

**- Облачное хранилище (S3, MinIO)** – для масштабируемости.

**- SFTP/WebDAV** – для передачи файлов.

### ****2.4. Обработка видео****

**- FFmpeg** – кодирование, изменение форматов, сжатие.

**- Pymediainfo** – получение метаданных.

### ****2.5. Веб-интерфейс и API****

**- Flask / FastAPI** – для REST API.

**- Django** – если потребуется сложная система.

**- React / Vue.js** – если понадобится веб-интерфейс.

### ****2.6. Очереди задач****

**- Celery + Redis** – асинхронные задачи.

**- RQ (Redis Queue)** – более простая альтернатива.

### ****2.7. Контейнеризация и деплой****

**- Docker / Docker Compose** – удобное развертывание.

### ****2.8. Мониторинг и логирование****

Для успешной разработки и поддержки проекта крайне важно отслеживать производительность системы, а также эффективно логировать все ошибки и события, которые могут возникнуть в процессе работы. В этом контексте использование GitHub и других инструментов играет важную роль в организации мониторинга и логирования.

#### **Использование GitHub для мониторинга**

1. **GitHub Actions**:

**- Автоматизация CI/CD процессов**: GitHub Actions позволяет автоматизировать процесс тестирования, сборки и деплоя проекта. Каждое изменение в репозитории может инициировать автоматическое выполнение тестов, что гарантирует стабильность кода.

**- Мониторинг билдов**: GitHub предоставляет возможность отслеживать состояние каждого билда и теста, включая успешные и неудачные сборки. Это помогает своевременно выявлять проблемы на ранних этапах разработки.

1. **GitHub Issues и Pull Requests**:

**- Использование Issues для отслеживания багов**: Для более удобного отслеживания проблем и дефектов можно использовать GitHub Issues. Каждую проблему можно категоризовать, назначать ответственным и отслеживать процесс решения.

**- Pull Requests для контроля изменений**: При внесении изменений в код через PR можно отслеживать, кто и какие изменения внес, а также оставить комментарии для обсуждения с командой. Это помогает сохранять прозрачность и эффективность в управлении проектом.

1. **Интеграция с внешними инструментами для мониторинга**:

**- Sentry**: Интеграция с Sentry позволяет отслеживать ошибки в реальном времени, отправляя уведомления о возникших проблемах.

**- Prometheus + Grafana**: Для более глубокого мониторинга и визуализации производительности можно использовать Prometheus и Grafana. Эти инструменты позволяют собирать метрики и строить графики, что помогает понять, какие части системы требуют оптимизации.

#### **Логирование**

Для качественного логирования можно использовать такие решения, как:

**- Loguru**: Это библиотека для Python, которая обеспечивает простое и гибкое логирование. Она поддерживает уровни логирования (info, warning, error, critical), форматирование и отправку логов в различные системы.

**- Elasticsearch + Kibana**: Для продвинутого логирования можно настроить Elasticsearch и Kibana для хранения и визуализации логов, что поможет эффективно отслеживать состояние проекта в реальном времени.

### ****2.9. Графические средства оформления и текста (Figma + Photoshop)****

Кроме того, важным аспектом проекта является создание графических материалов, таких как интерфейсы, схемы и изображения. Для этого удобно использовать инструменты дизайна, которые позволяют создавать высококачественные визуальные элементы.

#### **Figma**

**Figma** — это современный инструмент для создания UI/UX дизайнов, который предлагает следующие преимущества:

**- Совместная работа**: Все изменения в проекте видны в реальном времени, что позволяет команде работать одновременно над проектом, независимо от их местоположения.

**- Прототипирование**: Возможность быстро создавать прототипы интерфейсов и тестировать их на пользователях.

**- Гибкость**: Figma поддерживает векторную графику, что позволяет создавать адаптивные интерфейсы и элементы.

**- Библиотеки и плагины**: Удобные библиотеки компонентов и сторонние плагины для расширения возможностей работы.

**Преимущества Figma:**

**- Легкость в использовании**: Доступность всех инструментов для начинающих и профессионалов.

**- Работа в браузере**: Не нужно устанавливать дополнительные приложения, так как все работает в браузере.

**- Облачное хранилище**: Все работы сохраняются в облаке, что делает их доступными для всех участников проекта.

#### **Photoshop**

**Photoshop** — это мощное приложение для редактирования растровой графики, которое широко используется для создания высококачественных изображений. В проекте Photoshop может быть использован для:

**- Обработки изображений**: Изменение размеров, коррекция цвета, добавление текстур и других элементов.

**- Создания визуальных эффектов**: С помощью Photoshop можно создавать сложные графические элементы, которые могут быть использованы в интерфейсе или маркетинговых материалах.

**- Векторная графика**: Хотя Photoshop в основном используется для работы с растровыми изображениями, он также поддерживает создание векторных объектов для создания иконок и элементов интерфейса.

**Преимущества Photoshop:**

**- Мощные инструменты**: Photoshop предоставляет обширный набор инструментов для детальной работы с изображениями.

**- Высокая точность**: Программное обеспечение позволяет работать с пиксельной точностью, что важно для дизайнеров.

**- Интеграция с другими продуктами Adobe**: Легкая интеграция с Illustrator, After Effects и другими программами Adobe для создания комплексных проектов.

### ****2.10. Канбан-доска для управления задачами (YouGile)****

Для эффективного управления задачами в проекте используется **YouGile** — современная и удобная канбан-доска, которая позволяет отслеживать прогресс работы, организовывать задачи и назначать ответственных. В YouGile удобно визуализировать рабочий процесс, что помогает команде быстрее достигать целей и эффективно решать возникающие задачи.

#### **Преимущества использования YouGile**:

1. **Гибкость в настройке**:

- YouGile позволяет настроить доски под различные этапы разработки и легко адаптировать их под нужды команды. Можно настроить категории, колонки и фильтры для более детальной организации задач.

1. **Интуитивно понятный интерфейс**:

- Доски в YouGile имеют простой и понятный интерфейс, что позволяет быстро организовывать задачи. Перетаскивание карточек между колонками помогает наглядно отслеживать текущий статус каждой задачи.

1. **Совместная работа**:

- YouGile позволяет команде работать совместно, обсуждать задачи, оставлять комментарии и загружать файлы. Это помогает улучшить коммуникацию внутри команды и ускоряет решение возникающих проблем.

1. **Уведомления и интеграция с другими сервисами**:

- YouGile поддерживает уведомления о статусах задач, что помогает держать всех участников проекта в курсе изменений. Также инструмент может быть интегрирован с другими сервисами, такими как GitHub или Slack, для улучшения взаимодействия с кодом и командой.

1. **Простота в использовании**:

- Платформа имеет интуитивно понятный интерфейс и не требует длительного обучения для работы. Это позволяет команде сосредоточиться на решении задач, а не на освоении инструмента.

#### **Использование YouGile в проекте**:

1. **Управление задачами**:

- С помощью YouGile можно организовать задачи по стадиям разработки, таким как "To Do", "In Progress" и "Done". Это помогает визуализировать и отслеживать прогресс проекта.

1. **Назначение ответственных**:

- Для каждой задачи можно назначить ответственного разработчика, что помогает распределить нагрузку и гарантировать, что каждый этап работы будет выполнен своевременно.

1. **Отчеты и аналитика**:

- YouGile предоставляет простые средства для генерации отчетов и анализа выполнения задач, что позволяет команде и руководителям проекта видеть, как идет работа и где могут возникнуть проблемы.

**2.11. Блок-схема проекта**

Блок-схема проекта визуализирует основные процессы работы системы и взаимодействие её компонентов. Она помогает понять логику работы утилиты, упрощает разработку и облегчает дальнейшую поддержку кода. На рисунке 2 представлена

блок-схема проекта.

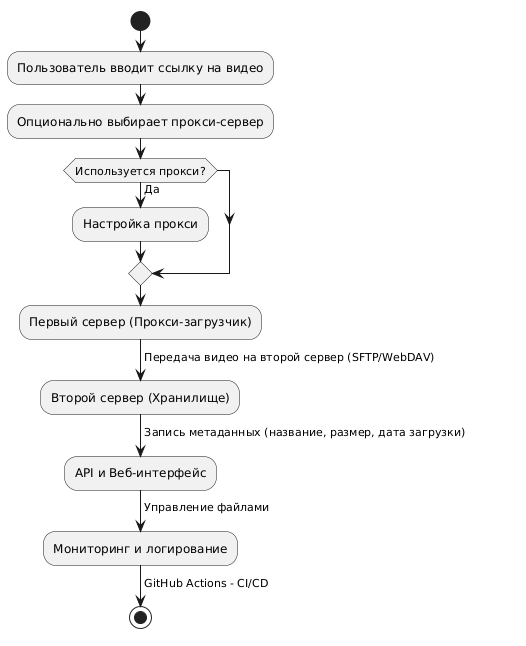


Рисунок 2 – блок-схема проекта

**Основные элементы блок-схемы:**

1. Начало работы (ввод данных пользователем)

- Пользователь вводит URL видео и при необходимости указывает прокси-сервер.

- Эти данные передаются в систему для дальнейшей обработки.

1. Первый сервер – Прокси-загрузчик

- Подключается к видеохостингу через указанный прокси.

- Загружает видеофайл с помощью библиотеки yt-dlp.

- Проверяет корректность загрузки (наличие ошибок, статус HTTP-запросов).

- (Опционально) Обрабатывает видео (ffmpeg-python), например, меняет формат или снижает качество.

- После успешной загрузки передаёт файл на второй сервер хранения.

1. Второй сервер – Хранилище видео

- Получает файл от прокси-загрузчика.

- Сохраняет видео локально или в облачное хранилище (S3, MinIO).

- Записывает метаданные (размер, название, дата загрузки).

- Позволяет пользователю получать список файлов и скачивать видео через API.

1. API и Веб-интерфейс

- Обеспечивает доступ к загруженным видео через Flask/FastAPI.

- Позволяет пользователю просматривать, скачивать и управлять файлами.

- При необходимости можно добавить React/Vue.js для веб-интерфейса.

1. Мониторинг и логирование

- Все процессы логируются через Loguru (ошибки, успешные загрузки).

- Для мониторинга работы серверов используются Prometheus и Grafana.

- CI/CD процессы автоматизируются с помощью GitHub Actions.

Зачем нужна блок-схема?

- Упрощает понимание архитектуры проекта.

- Позволяет разработчикам быстрее вникнуть в работу системы.

- Помогает выявить потенциальные узкие места и улучшить структуру проекта.

- Облегчает поддержку и масштабирование системы.

Блок-схема демонстрирует логический поток данных в системе, обеспечивая чёткую и понятную структуру проекта.

### ****2.12.**** Прототипирование, дизайн и функциональная демонстрация программного продукта

В ходе сбора требований был разработан общий концепт дизайна приложения, включающий ключевые элементы пользовательского интерфейса. На основе этого концепта были созданы подробные прототипы, позволяющие визуализировать и оценить будущий функционал системы.

На рисунке 3 мы можем увидеть первоначальные наработки интерфейса программы.

### 

Рисунок 3 - Первоначальные наработки интерфейса программы

### При вставке ссылки на видеоролик происходит уведомление пользователя, посредством логов о том, что происходит взаимодействие с программой: получение и последующее распознавание данных, при помощи которых мы можем получить сведения о корректности загружаемого ролика. На изображении 4 представлен интерфейс программы после первоначального взаимодействия с пользователем.

### 

Рисунок 4 - Первоначальные наработки интерфейса программы

### После получения первоначальных данных пользователь может настроить необходимые данные для получения от видеоролика: выбрать сохранить видеоролик, аудио из него или же обложку видеоролика. Также для всего вышеперечисленного пользователь может выбрать необходимое качество, формат и разрешение. Примеры выбора приведены на рисунках 5

### 

Рисунок 5 – Выбор качества и формата видеоролика

### 

Рисунок 6 – Выбор режима программы

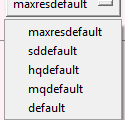


Рисунок 7 – Выбор разрешения обложки к ролику

### После выбора параметров пользователь может выбрать директорию, в которую будет сохранено видео или обложка к ролику. На рисунке 8-9 изображен процесс выбора директории для видео или обложки в необходимую директорию.

### 

Рисунок 8 – Процесс выбора директории для сохранения

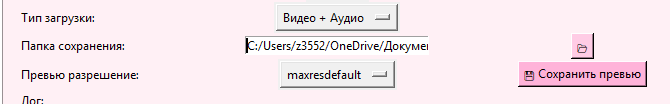


Рисунок 9 – Успешный выбор директории

В случае если пользователь не выбирает директорию для сохранения или забывает, то видеоролик будет сохранен в директорию по умолчанию. Это можно увидеть на рисунке 10.

### 

Рисунок 10 – Ролик сохранен в директорию по умолчанию

### Для демонстрации работоспособности программного продукта необходимо продемонстрировать его работоспособность. Для этого мы выполним все вышеописанные действия. В качестве папки для сохранения мною будет выбрана папка «Загрузки» из-за своей легкодоступности для тестирования, а качество укажу «Premium-mp4», так как оно предназначено для наивысшего качества загрузки. На рисунке 11 изображены выбранные параметры.

### 

### Рисунок 11 – Выбранные параметры загрузки

### Теперь нажмём на кнопку для скачивания видеоролика. В ходе загрузки мы видим процесс загрузки в информационном поле «Логи», что нам позволяет увидеть сколько нам осталось ожидать загрузки видео. В конце успешной загрузки программа выдаст нам сообщение о том, что видео сохранено. Пример такого сообщения можно увидеть на рисунке 12.

### 

### Рисунок 12 – Сообщение об успешной загрузки

### Теперь необходимо убедится, что видеоролик действительно был сохранен. На рисунке 13 продемонстрировано, что видеоролик действительно был успешно сохранен.

### 

### Таким образом, была произведена проверка работоспособности программного продукта, его возможности, а также был продемонстрирован интерфейс.

**3. Разработка программного продукта**

**3.1. Инструментальные средства разработки и программная реализация**

**3.1.1. Инструментальные средства разработки**

Задача заключалась в разработке программного продукта для упрощения загрузки видео и аудио. Программа должна была предоставлять интуитивно понятный графический интерфейс, позволяющий пользователю вводить ссылку на видео, выбирать качество и формат загрузки, просматривать и сохранять миниатюру (превью), а также вести лог всех операций. Целью разработки было создание удобного инструмента, подходящего как для технически неподготовленных пользователей, так и для продвинутых. В рамках проекта использовались следующие программные средства:

Visual Studio Code (VS Code) — это кроссплатформенный текстовый редактор с открытым исходным кодом от Microsoft, который предоставляет широкие возможности для разработки программ. Благодаря богатой экосистеме расширений и поддержке отладки, автодополнения кода и интеграции с системами контроля версий, VS Code стал идеальной средой разработки для проекта. Он особенно удобен при работе с Python, поскольку позволяет настраивать окружения, управлять зависимостями и запускать скрипты прямо из редактора.

В качестве языка программирования был выбран **Python**, благодаря его простоте и широким возможностям. Основной функционал скачивания видео реализован с использованием библиотеки yt-dlp — это активно поддерживаемый форк youtube-dl, предоставляющий расширенные возможности загрузки с различных видеохостингов.  
Библиотека позволяет получать информацию о видео, выбирать доступные форматы, а также управлять процессом загрузки. Для конвертации медиафайлов использовалась интеграция с ffmpeg, путь к которому указывается вручную в коде.

Для создания графического интерфейса была использована стандартная библиотека tkinter, поставляемая вместе с Python. Интерфейс включает элементы управления вводом (текстовые поля, выпадающие списки, кнопки), просмотр миниатюры видео, а также окно для вывода логов, что делает взаимодействие с программой удобным и наглядным.

Дополнительно использовались библиотеки Pillow и requests для обработки изображений и получения превью видео с YouTube.  
Пользователь может выбрать разрешение миниатюры (например, maxresdefault), просмотреть её в окне приложения и при желании сохранить в локальную папку. Обработка изображений выполняется средствами Pillow, а загрузка превью — через requests.

Весь процесс разработки велся с использованием системы контроля версий Git, а размещение проекта и отслеживание изменений осуществлялось через платформу GitHub.  
Использование GitHub обеспечило:

* контроль версий и историю изменений;
* возможность отката к стабильным версиям проекта;
* прозрачность процесса разработки;
* резервное копирование кода;
* публикацию проекта в открытый доступ для обратной связи и демонстрации возможностей.

Также проект оформлен с использованием файла README.md, в котором описана инструкция по запуску, список зависимостей и скриншоты интерфейса. Это делает приложение легко устанавливаемым и понятным для конечного пользователя. На рисунке 3-4 представлены скриншоты репозитория и коммитов.

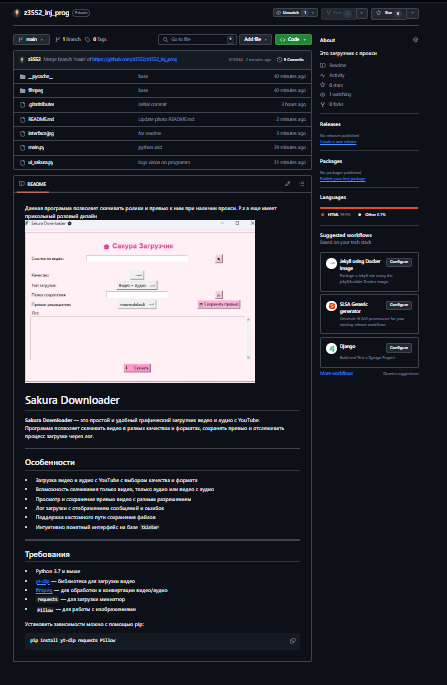


Рисунок 3 – Скриншот репозитория GitHub.

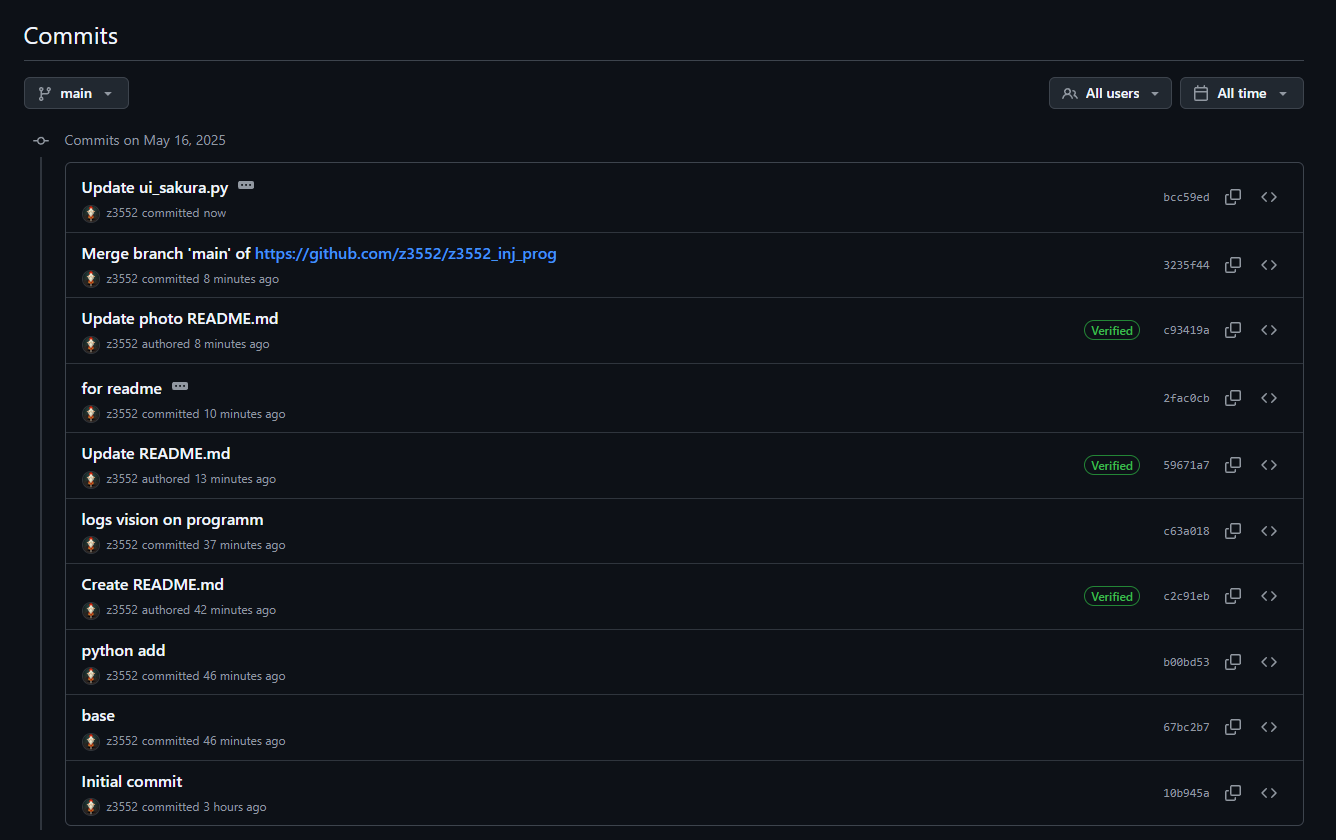


Рисунок 4 – Скриншот коммитов GitHub

Yougile представляет собой инструмент управления проектами, который получил множество положительных отзывов от пользователей. Вот несколько причин, по которым Yougile может быть хорошим выбором для управления проектами:

- удобный интерфейс: пользователи отмечают, что интерфейс Yougile удобен в использовании и разработчики учитывают пожелания пользователей.

- поддержка: сервис предоставляет быструю и качественную поддержку пользователей.

- высокая стабильность: пользователи сообщают о высокой стабильности работы приложения, как настольной, так и веб-версии.

- ценовая политика: для небольших компаний Yougile предлагает бесплатный тариф без ограничений, что является большим плюсом.

- отечественный продукт: Yougile является примером качественного российского софта с активной поддержкой.

Эти аспекты делают Yougile привлекательным выбором для тех, кто ищет эффективное и надежное решение для управления проектами. На рисунке 5 представлена доска Yougile.

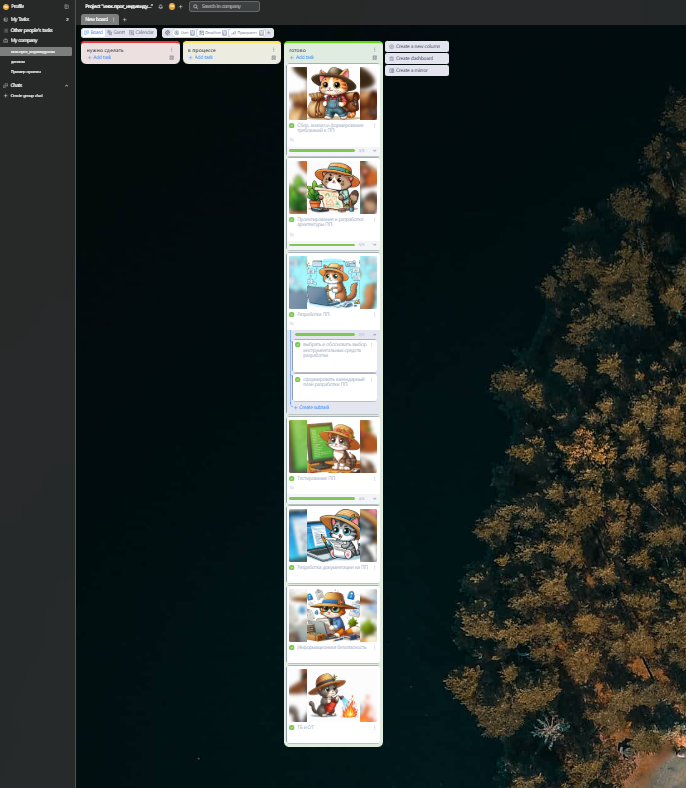


Рисунок 5 -Доска Yougile

**3.1.2 Программная реализация**

Здесь будут рассмотрены ключевые элементы кода, который работает с программой. Полный текст программы доступен в приложении А.

from tkinter import (

Label, Entry, Button, StringVar, OptionMenu, filedialog,

Text, Scrollbar, Frame

)

import yt\_dlp

import os

import threading

import requests

from PIL import Image, ImageTk

from io import BytesIO

import re

import random

import string

Импортируются модули:

* tkinter — для создания графического интерфейса;
* yt\_dlp — для скачивания видео с YouTube;
* os, threading, requests, PIL и другие — для работы с файловой системой, загрузки картинок, многопоточности и генерации случайных имён файлов.

def generate\_random\_filename(length=10):

return ''.join(random.choices(string.ascii\_lowercase + string.digits, k=length))

Функция создает случайное имя из букв и цифр длиной length. Используется, если название видео содержит недопустимые символы для имени файла.

class YTDLogger:

def \_\_init\_\_(self, log\_func):

self.log\_func = log\_func

def debug(self, msg):

if msg.strip():

self.log\_func(msg)

def warning(self, msg):

self.log\_func(f"⚠️ {msg}")

def error(self, msg):

self.log\_func(f"❌ {msg}")

Этот класс позволяет перенаправить сообщения от библиотеки yt\_dlp в графический интерфейс, а не просто в консоль. Метод log\_func указывает, куда выводить текст.

class SakuraDownloader:

def \_\_init\_\_(self, root):

self.root = root

self.url = StringVar()

self.quality = StringVar()

self.download\_type = StringVar(value="Видео + Аудио")

self.output\_path = StringVar()

self.video\_title = StringVar()

self.thumb\_url\_map = {}

self.thumb\_resolution = StringVar(value="maxresdefault")

self.formats = []

self.thumbnail = None

self.log\_output = None

self.build\_ui()

Это основа программы. В \_\_init\_\_ создаются переменные для хранения введённых данных, будущих значений (качество, путь, тип загрузки и т.д.). Также вызывается метод build\_ui, чтобы отрисовать интерфейс.

def build\_ui(self):

self.root.configure(bg="#ffe6f0")

frame = Frame(self.root, bg="#fff0f5", padx=15, pady=15, bd=3, relief="ridge")

frame.pack(padx=10, pady=10)

Label(frame, text="🌸 Сакура Загрузчик", ...).grid(row=0, column=0, columnspan=3)

Label(frame, text="Ссылка на видео:").grid(row=1, column=0)

Entry(frame, textvariable=self.url, width=50).grid(row=1, column=1)

Button(frame, text="🔍", command=self.refresh\_video\_info).grid(row=1, column=2)

# ... и так далее (остальные элементы интерфейса)

def refresh\_video\_info(self):

url = self.url.get().strip()

if not url:

return

def task():

self.log("Получение информации о видео...")

try:

ydl\_opts = {"quiet": True, "skip\_download": True}

with yt\_dlp.YoutubeDL(ydl\_opts) as ydl:

info = ydl.extract\_info(url, download=False)

self.video\_title.set(info.get("title", ""))

self.formats = info.get("formats", [])

self.quality\_menu['menu'].delete(0, 'end')

for fmt in self.formats:

label = f"{fmt.get('format\_id')} - {fmt.get('format\_note', '')}"

self.quality\_menu['menu'].add\_command(label=label,

command=lambda v=fmt.get("format\_id"): self.quality.set(v))

if self.formats:

self.quality.set(self.formats[0]['format\_id'])

# Превью

video\_id = info.get("id")

if video\_id:

resolutions = ["maxresdefault", "sddefault", ...]

self.thumb\_url\_map = {

res: f"https://img.youtube.com/vi/{video\_id}/{res}.jpg"

for res in resolutions

}

chosen = self.thumb\_url\_map[self.thumb\_resolution.get()]

response = requests.get(chosen)

img = Image.open(BytesIO(response.content)).resize((120, 90))

self.thumbnail = ImageTk.PhotoImage(img)

self.thumb\_label.configure(image=self.thumbnail)

except Exception as e:

self.log(f"Ошибка: {e}")

threading.Thread(target=task).start()

Асинхронно получает информацию о видео: название, форматы, ID и картинку-превью. После получения информации обновляет выпадающее меню с качествами и загружает изображение в интерфейс.

def save\_thumbnail(self):

if not self.thumb\_url\_map:

self.log("Нет превью для сохранения.")

return

url = self.thumb\_url\_map.get(self.thumb\_resolution.get())

if not url:

return

folder = filedialog.askdirectory()

if not folder:

return

try:

response = requests.get(url)

title = self.video\_title.get()

if re.search(r'[\\/\*?:"<>|]', title):

filename = generate\_random\_filename()

self.log("Недопустимые символы. Название заменено.")

else:

filename = title.replace(" ", "\_")

filepath = os.path.join(folder, f"{filename}\_{self.thumb\_resolution.get()}.jpg")

with open(filepath, "wb") as f:

f.write(response.content)

self.log(f"Превью сохранено: {filepath}")

except Exception as e:

self.log(f"Ошибка: {e}")

Позволяет сохранить превью видео в выбранную пользователем папку. Проверяет, есть ли запрещённые символы в имени файла, и при необходимости заменяет название на случайное.

def download\_video(self):

def task():

path = self.output\_path.get() or "output"

os.makedirs(path, exist\_ok=True)

url = self.url.get().strip()

mode = self.download\_type.get()

try:

format\_id = self.quality.get()

ydl\_opts = {

"format": format\_id if mode != "Видео + Аудио" else "bestvideo+bestaudio",

"outtmpl": os.path.join(path, "%(title)s.%(ext)s"),

"ffmpeg\_location": FFMPEG\_PATH,

"postprocessors": [],

"logger": YTDLogger(self.log)

}

if mode == "Аудио":

ydl\_opts["postprocessors"].append({

'key': 'FFmpegExtractAudio',

'preferredcodec': 'mp3',

'preferredquality': '192',

})

with yt\_dlp.YoutubeDL(ydl\_opts) as ydl:

ydl.download([url])

self.log("✅ Загрузка завершена.")

except Exception as e:

self.log(f"❌ Ошибка при загрузке: {e}")

threading.Thread(target=task).start()

Скачивает видео или аудио в зависимости от выбранного режима. Используется yt\_dlp и при необходимости — ffmpeg для конвертации в mp3. Всё выполняется в отдельном потоке, чтобы не зависал интерфейс.

def log(self, msg):

self.log\_output.insert("end", msg + "\n")

self.log\_output.see("end")

Метод, который выводит текст в нижнюю текстовую область логов. Используется для вывода состояния программы: ошибок, этапов загрузки и сообщений от yt\_dlp.

**4. Тестирование программного продукта**

**4.1 Выбор метода обеспечения качества**

Для гарантии высокого качества отдельных компонентов программного продукта, было необходимо провести тестирование всех возможных вариантов использования и некоторых элементов. Ключевым аспектом является то, чтобы эти компоненты функционировали и вели себя в соответствии с первоначальным замыслом, обеспечивая беспроблемное использование для конечного пользователя.

4.2 Тестирование продукта

Для обеспечения качества и стабильной работы программного продукта Sakura Downloader был выбран метод ручного тестирования. Такой подход позволил эмулировать реальные сценарии использования программы в среде Visual Studio Code с последующим выполнением и отладкой кода на Python. Ручное тестирование особенно эффективно в GUI-приложениях, поскольку позволяет напрямую взаимодействовать с интерфейсом, оценивать корректность отображения данных и поведение при различных пользовательских действиях.

Программа активно использует библиотеку yt-dlp для загрузки мультимедийного контента, а также инструменты работы с сетью (requests) и изображениями (PIL). Особое внимание в ходе тестирования уделялось корректной работе многопоточности, стабильности загрузки, обработке ошибок и взаимодействию с внешними ресурсами (YouTube, файловая система).

Был разработан тестовый план, охватывающий следующие ключевые модули и пользовательские сценарии,которые отображены в таблице 1.

Таблица 1 - Тест-план для проведения тестирования проекта

|  |  |
| --- | --- |
| Вид тестирования | Способ тестирования |
| Тестирование функционала пользователя | Ручное |

Тест-кейсы для функционала владельца представлены в таблице 2.

Таблица 2 -Тест-кейс

|  |  |  |
| --- | --- | --- |
| Название: | Тест получения информации о видео и отображения превью | |
| Функция: | Загрузка метаданных видео с YouTube и отображение изображения превью | |
| Действие: | Ожидаемый результат | Результат теста:  выполнен |
| Предусловие: | Пользователь открыл программу Sakura Downloader | пройден |
| Ввести корректную ссылку на видео YouTube | Отображается название видео, список доступных форматов, загружается изображение превью | пройден |
| Изменить разрешение превью на другое (например, с maxresdefault на hqdefault) | Отображается новое изображение превью соответствующего качества | пройден |
| Ввести ссылку на видео без доступного превью (например, видео в обработке) | Отображается сообщение об ошибке загрузки изображения | пройден |

**Заключение**

В рамках данной работы была реализована программная система Sakura Downloader, предназначенная для загрузки видеоконтента с платформы YouTube с возможностью выбора качества и формата. Основной целью проекта являлось создание интуитивно понятного, функционального и визуально привлекательного интерфейса, способного упростить процесс скачивания видео- и аудиофайлов для конечного пользователя. Процесс разработки включал в себя как техническую реализацию программного кода, так и проектирование пользовательского интерфейса с учетом эргономики и удобства взаимодействия.

В процессе разработки использовались современные и надёжные программные средства: библиотека yt-dlp для обработки видеоконтента, модуль tkinter для реализации графического интерфейса, а также вспомогательные библиотеки, такие как Pillow, requests и threading, обеспечивающие стабильность, гибкость и высокую производительность приложения. Программа позволяет не только скачивать видео, но и сохранять эскизы (превью), выбирать разрешение обложки, а также следить за ходом выполнения через встроенный лог.

Вся логика приложения была реализована с учетом требований к читаемости, поддерживаемости и расширяемости кода. Благодаря использованию многопоточности пользовательский интерфейс остаётся отзывчивым даже при длительной загрузке, что повышает общее качество взаимодействия и удовлетворённость пользователей.

Проведённое ручное тестирование подтвердило корректность работы всех основных функций. Программа стабильно функционирует при различных вводных данных и условиях, обеспечивая пользователю положительный опыт использования. Были проанализированы возможные ошибки и исключения, и для большинства из них реализована система оповещений в логах, что упрощает процесс устранения неполадок.

Таким образом, можно сделать вывод, что поставленные задачи были успешно выполнены, а цели проекта — достигнуты. Созданный программный продукт соответствует современным требованиям к настольному прикладному ПО, является готовым к использованию и обладает потенциалом для дальнейшего развития.

Несмотря на достигнутый уровень реализации, программный продукт Sakura Downloader не является финальной версией. В процессе использования и тестирования были выявлены возможные направления усовершенствования, каждое из которых способно значительно повысить удобство, функциональность и универсальность системы.

Во-первых, перспективным направлением является реализация функции загрузки по плейлистам или каналам, что особенно актуально для пользователей, работающих с большим объемом видеоматериала. Это позволит скачивать несколько видео автоматически без необходимости ввода каждой ссылки вручную.

Во-вторых, внедрение системы очередей обеспечит более продвинутую логику управления загрузками. Пользователь сможет формировать список задач и отслеживать их выполнение в реальном времени, что значительно улучшит эффективность при массовой загрузке.

Следующим важным шагом может стать интеграция поддержки других видеохостингов — таких как Vimeo, TikTok, Dailymotion и других. Это позволит расширить круг пользователей программы и сделать её более универсальной.

Кроме того, полезной функцией станет хранение истории загрузок, с возможностью открытия, переименования или удаления файлов прямо из интерфейса приложения. Это сделает программу не только средством загрузки, но и полноценным менеджером мультимедийного контента.

Также возможно добавление системы автоматического обновления, что избавит пользователей от необходимости самостоятельно искать и устанавливать новые версии. Это обеспечит своевременное получение новых функций и улучшений безопасности.

С технической точки зрения, интерес представляет возможность портирования приложения на другие операционные системы (Linux, macOS), а также разработка мобильной версии для Android, что позволит использовать Sakura Downloader в любом месте и в любое время.

В целом, проект обладает высокой степенью гибкости, и за счёт открытости к доработкам может развиваться в различных направлениях в зависимости от требований конечных пользователей и актуальных технологических трендов.

**Список литературы**

1. **yt-dlp Documentation.** Современная библиотека для загрузки медиафайлов с YouTube и других платформ, форк youtube-dl. – GitHub, 2024.https://github.com/yt-dlp/yt-dlp
2. **Работа с графическим интерфейсом в Python: библиотека tkinter. –** Хабр, 2022.https://habr.com/ru/articles/548210/
3. **Как скачать видео с YouTube с помощью Python и yt-dlp. –** Хабр, 2023.https://habr.com/ru/articles/722258/
4. **Универсальный парсер превью YouTube на Python. –** Хабр, 2021.https://habr.com/ru/articles/535708/
5. **Как организовать многопоточность в Python: потоки, очереди, таймеры. –** Хабр, 2020.  
   https://habr.com/ru/articles/501046/
6. **Создание кроссплатформенного GUI в Python с помощью Tkinter. –** Pythonist.ru,2022.  
   https://pythonist.ru/gui-v-python-s-pomoshchyu-tkinter/
7. **Requests – HTTP for Humans.** Документация библиотеки для запросов в интернете на Python. – Python Requests Project, 2024.  
   https://docs.python-requests.org/
8. **Pillow (PIL Fork) – работа с изображениями в Python. –** ReadTheDocs, 2023.https://pillow.readthedocs.io/en/stable/
9. **Visual Studio Code как среда разработки Python-проектов. –** Хабр, 2021.https://habr.com/ru/articles/678560/
10. **Git и GitHub для начинающих: как контролировать версии и публиковать проект.** – GitHub Docs, 2023.https://docs.github.com/ru/get-started/quickstart

Приложение А

**1. Исходный код**

**1.1. Код main.py:**

from tkinter import Tk

from ui\_sakura import build\_ui # Импорт красивого интерфейса

import os

import sys

def main():

root = Tk()

root.title("Sakura Downloader 🌸")

build\_ui(root)

root.mainloop()

if \_\_name\_\_ == "\_\_main\_\_":

main()

**1.2. Код ui\_sakura.py:**

# 📦 Импорт нужных классов и функций из tkinter (графический интерфейс)

from tkinter import (

Label, Entry, Button, StringVar, OptionMenu, filedialog,

Text, Scrollbar, Frame

)

# 📦 Импорт сторонних и стандартных библиотек

import yt\_dlp # Библиотека для скачивания видео с YouTube и других платформ

import os # Работа с путями и файловой системой

import threading # Для запуска функций в отдельном потоке (чтобы не тормозил интерфейс)

import requests # Для HTTP-запросов (например, загрузка превью)

from PIL import Image, ImageTk # Работа с изображениями (PIL = Pillow)

from io import BytesIO # Преобразование байтов в картинку

import re # Регулярные выражения (например, для проверки символов)

import random # Генерация случайных строк

import string # Символы для генерации имён

# Путь к ffmpeg (необходим для конвертации)

FFMPEG\_PATH = os.path.join(os.path.dirname(\_\_file\_\_), "ffmpeg", "bin", "ffmpeg.exe")

# 🔧 Функция генерации случайного имени файла

def generate\_random\_filename(length=10):

return ''.join(random.choices(string.ascii\_lowercase + string.digits, k=length))

# 🎯 Кастомный логгер для отображения логов yt\_dlp в GUI

class YTDLogger:

def \_\_init\_\_(self, log\_func):

self.log\_func = log\_func # Сюда передаём функцию, которая выводит сообщение в интерфейс

def debug(self, msg):

if msg.strip():

self.log\_func(msg)

def warning(self, msg):

self.log\_func(f"⚠️ {msg}") # Показываем предупреждения

def error(self, msg):

self.log\_func(f"❌ {msg}") # Показываем ошибки

# 🌸 Основной класс загрузчика

class SakuraDownloader:

def \_\_init\_\_(self, root):

self.root = root # Главное окно

# Переменные, привязанные к элементам интерфейса

self.url = StringVar()

self.quality = StringVar()

self.download\_type = StringVar(value="Видео + Аудио") # Значение по умолчанию

self.output\_path = StringVar()

self.video\_title = StringVar()

self.thumb\_url\_map = {} # Словарь с ссылками на превью

self.thumb\_resolution = StringVar(value="maxresdefault") # Разрешение превью

self.formats = [] # Список форматов видео

self.thumbnail = None # Здесь будет изображение превью

self.log\_output = None # Поле для логов

self.build\_ui() # Строим интерфейс

# 🧱 Функция создания графического интерфейса

def build\_ui(self):

self.root.configure(bg="#ffe6f0") # Цвет фона окна

# 📦 Основной контейнер с рамкой

frame = Frame(self.root, bg="#fff0f5", padx=15, pady=15, bd=3, relief="ridge")

frame.pack(padx=10, pady=10)

row = 0

# Заголовок

Label(frame, text="🌸 Сакура Загрузчик", bg="#fff0f5", fg="#d63384", font=("Helvetica", 16, "bold")).grid(row=row, column=0, columnspan=3, pady=10)

# 🎯 Поле ввода ссылки

row += 1

Label(frame, text="Ссылка на видео:", bg="#fff0f5").grid(row=row, column=0, sticky="w")

Entry(frame, textvariable=self.url, width=50).grid(row=row, column=1)

Button(frame, text="🔍", command=self.refresh\_video\_info, bg="#ffcce5").grid(row=row, column=2)

# 🖼️ Превью + название видео

row += 1

self.thumb\_label = Label(frame, bg="#fff0f5")

self.thumb\_label.grid(row=row, column=0, columnspan=1)

Label(frame, textvariable=self.video\_title, wraplength=400, bg="#fff0f5", fg="#cc3366").grid(row=row, column=1, columnspan=2)

# 🎞️ Качество

row += 1

Label(frame, text="Качество:", bg="#fff0f5").grid(row=row, column=0, sticky="w")

self.quality\_menu = OptionMenu(frame, self.quality, "") # Будет обновляться позже

self.quality\_menu.grid(row=row, column=1)

# 🎧 Тип загрузки

row += 1

Label(frame, text="Тип загрузки:", bg="#fff0f5").grid(row=row, column=0, sticky="w")

OptionMenu(frame, self.download\_type, "Видео", "Аудио", "Видео + Аудио").grid(row=row, column=1)

# 💾 Путь сохранения

row += 1

Label(frame, text="Папка сохранения:", bg="#fff0f5").grid(row=row, column=0, sticky="w")

Entry(frame, textvariable=self.output\_path, width=30).grid(row=row, column=1)

Button(frame, text="📂", command=self.choose\_output\_folder, bg="#ffcce5").grid(row=row, column=2)

# 🖼️ Выбор разрешения превью

row += 1

Label(frame, text="Превью разрешение:", bg="#fff0f5").grid(row=row, column=0, sticky="w")

OptionMenu(frame, self.thumb\_resolution, "maxresdefault", "sddefault", "hqdefault", "mqdefault", "default").grid(row=row, column=1)

Button(frame, text="💾 Сохранить превью", command=self.save\_thumbnail, bg="#ffb3d9").grid(row=row, column=2)

# 📃 Логи

row += 1

Label(frame, text="Лог:", bg="#fff0f5").grid(row=row, column=0, sticky="w")

row += 1

self.log\_output = Text(frame, height=8, wrap="word", bg="#fff0f5", fg="#800040")

self.log\_output.grid(row=row, column=0, columnspan=3)

scrollbar = Scrollbar(frame, command=self.log\_output.yview)

scrollbar.grid(row=row, column=3, sticky="ns")

self.log\_output.config(yscrollcommand=scrollbar.set)

# Кнопка СКАЧАТЬ

row += 1

Button(frame, text="⬇️ Скачать", command=self.download\_video, bg="#ff99cc").grid(row=row, column=1, pady=10)

# 📁 Открытие проводника для выбора папки

def choose\_output\_folder(self):

folder = filedialog.askdirectory()

if folder:

self.output\_path.set(folder)

# 🔍 Получение инфо о видео и превью

def refresh\_video\_info(self):

url = self.url.get().strip()

if not url:

return

def task(): # Запускается в потоке

self.log("Получение информации о видео...")

try:

ydl\_opts = {"quiet": True, "skip\_download": True}

with yt\_dlp.YoutubeDL(ydl\_opts) as ydl:

info = ydl.extract\_info(url, download=False)

self.video\_title.set(info.get("title", ""))

self.formats = info.get("formats", [])

# Обновляем меню выбора качества

self.quality\_menu['menu'].delete(0, 'end')

for fmt in self.formats:

label = f"{fmt.get('format\_id')} - {fmt.get('format\_note', '')} - {fmt.get('ext', '')}"

self.quality\_menu['menu'].add\_command(label=label,

command=lambda v=fmt.get("format\_id"): self.quality.set(v))

if self.formats:

self.quality.set(self.formats[0]['format\_id'])

# Загружаем превью

video\_id = info.get("id")

if video\_id:

resolutions = ["maxresdefault", "sddefault", "hqdefault", "mqdefault", "default"]

self.thumb\_url\_map = {res: f"https://img.youtube.com/vi/{video\_id}/{res}.jpg" for res in resolutions}

chosen = self.thumb\_url\_map[self.thumb\_resolution.get()]

response = requests.get(chosen)

img = Image.open(BytesIO(response.content)).resize((120, 90))

self.thumbnail = ImageTk.PhotoImage(img)

self.thumb\_label.configure(image=self.thumbnail)

except Exception as e:

self.log(f"Ошибка: {e}")

threading.Thread(target=task).start()

# 💾 Сохранение превью

def save\_thumbnail(self):

if not self.thumb\_url\_map:

self.log("Нет превью для сохранения.")

return

url = self.thumb\_url\_map.get(self.thumb\_resolution.get())

if not url:

return

folder = filedialog.askdirectory()

if not folder:

return

try:

response = requests.get(url)

title = self.video\_title.get()

if re.search(r'[\\/\*?:"<>|]', title): # Проверка на недопустимые символы

filename = generate\_random\_filename()

self.log("Недопустимые символы. Название заменено.")

else:

filename = title.replace(" ", "\_")

filepath = os.path.join(folder, f"{filename}\_{self.thumb\_resolution.get()}.jpg")

with open(filepath, "wb") as f:

f.write(response.content)

self.log(f"Превью сохранено: {filepath}")

except Exception as e:

self.log(f"Ошибка: {e}")

# ⬇️ Функция загрузки видео/аудио

def download\_video(self):

def task():

path = self.output\_path.get() or "output"

os.makedirs(path, exist\_ok=True)

url = self.url.get().strip()

mode = self.download\_type.get()

try:

format\_id = self.quality.get()

ydl\_opts = {

"format": format\_id if mode != "Видео + Аудио" else "bestvideo+bestaudio",

"outtmpl": os.path.join(path, "%(title)s.%(ext)s"),

"ffmpeg\_location": FFMPEG\_PATH,

"postprocessors": [],

"logger": YTDLogger(self.log) # 👈 Передаём логгер

}

if mode == "Аудио":

ydl\_opts["postprocessors"].append({

'key': 'FFmpegExtractAudio',

'preferredcodec': 'mp3',

'preferredquality': '192',

})

with yt\_dlp.YoutubeDL(ydl\_opts) as ydl:

ydl.download([url])

self.log("✅ Загрузка завершена.")

except Exception as e:

self.log(f"❌ Ошибка при загрузке: {e}")

threading.Thread(target=task).start()

# 📢 Вывод логов в текстовое поле

def log(self, msg):

self.log\_output.insert("end", msg + "\n")

self.log\_output.see("end")

# 🏗️ Запуск интерфейса

def build\_ui(root):

SakuraDownloader(root)