Частное профессиональное образовательное учреждение Тюменского областного союза потребительских обществ

«Тюменский колледж экономики, управления и права» (ЧПОУ ТОСПО «ТюмКЭУП»)

ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ

ПМ. 02 ОСУЩЕСТВЛЕНИЕ ИНТЕГРАЦИИ ПРОГРАММНЫХ МОДУЛЕЙ

наименование профессионального модуля

Специальность 09.02.07 Информационные системы и программирование

Студента 3 курса ИП-22-14б группы

форма обучения очная

(очная, заочная)

Честных Владимира Алексеевича

(фамилия, имя, отчество)

Место практики ООО «КАЛЬДЕРА»

(наименование организации)

Срок практики с «06» мая 2025 г. по «26» мая 2025г.

Руководители практики

от организации

  Генеральный директор \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Ростовский

должность подпись ФИО

от образовательной организации

Преподаватель М.В. Тетерина

должность подпись ФИО

2025

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc199972008)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc199972009)

[2 ТРЕБОВАНИЯ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ VIDIFY 13](#_Toc199972010)

[3 РАЗРАБОТКА ТЕХНИЧЕСКОГО ЗАДАНИЯ 19](#_Toc199972011)

[4 РАЗРАБОТКА СТРУКТУРЫ ПРОЕКТА 25](#_Toc199972012)

[5 РАБОТА В СИСТЕМЕ КОНТРОЛЯ ВЕРСИЙ 30](#_Toc199972013)

[6 ВНЕШНЕЕ ПРОЕКТИРОВАНИЕ 44](#_Toc199972014)

[7 ВНУТРЕННЕЕ ПРОЕКТИРОВАНИЕ (РАЗРАБОТКА СХЕМ И ДИАГРАММ ПРОЕКТА) 49](#_Toc199972015)

[8 РАЗРАБОТКА МОДУЛЕЙ ПРОЕКТА И ИХ ЭЛЕМЕНТОВ 54](#_Toc199972016)

[9 ИНТЕГРАЦИЯ МОДУЛЕЙ В ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ 73](#_Toc199972017)

[10 МОДИФИКАЦИЯ МОДУЛЕЙ ПРОЕКТА 79](#_Toc199972018)

[11 ОТЛАДКА МОДУЛЕЙ ПРОГРАММНОГО ПРОЕКТА. ОРГАНИЗАЦИЯ ОБРАБОТКИ ИСКЛЮЧЕНИЙ 83](#_Toc199972019)

[12 ОТЛАДКА ПРОЕКТА. ИНСПЕКЦИЯ КОДА МОДУЛЕЙ ПРОЕКТА 86](#_Toc199972020)

[13 ТЕСТИРОВАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ СРЕДСТВАМИ ИНСТРУМЕНТАЛЬНОЙ СРЕДЫ РАЗРАБОТКИ. ВЫПОЛНЕНИЕ ФУНКЦИОНАЛЬНОГО ТЕСТИРОВАНИЯ 88](#_Toc199972021)

[ЗАКЛЮЧЕНИЕ 93](#_Toc199972022)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 94](#_Toc199972023)

# **ВВЕДЕНИЕ**

В ходе учебной практики я проходил стажировку в организации, деятельность которой связана с разработкой программного обеспечения для автоматизации бизнес-процессов и цифровой трансформации организаций. ООО «КАЛЬДЕРА» занимается созданием прикладных ИТ-решений для различных сфер, включая интеграцию, сопровождение и адаптацию программных продуктов под потребности заказчика. Предприятие реализует полный цикл разработки – от постановки задачи до внедрения и поддержки.

В рамках практики, в целях углубления профессиональных навыков и с согласия руководителя, я принял решение реализовать собственный проект – программное приложение Vidify. Данный проект стал базой для отчёта, так как позволил в полной мере применить полученные в колледже знания и продемонстрировать умения в области анализа, проектирования, программирования и тестирования программных систем.

Приложение Vidify представляет собой инструмент для автоматизированной обработки видеоконтента. Основная задача приложения – упрощение работы с видеороликами: их загрузка, конвертация и уникализация. Реализация проекта позволила мне пройти все ключевые этапы жизненного цикла программного обеспечения, от анализа предметной области до выпуска готового решения.

В процессе прохождения практики я выполнил следующие этапы работ в соответствии с индивидуальным заданием:

* анализ предметной области;
* определение требований проекта;
* разработка технического задания;
* разработка структуры проекта;
* работа в системе контроля версий;
* внешнее проектирование (разработка внешней спецификации);
* внутреннее проектирование (разработка схем и диаграмм проекта);
* разработка модулей проекта и их элементов;
* интеграция модулей в программное обеспечение;
* модификация модулей проекта;
* отладка модулей программного проекта;
* организация обработки исключений;
* отладка проекта, инспекция кода модулей проекта;
* тестирование интерфейса пользователя средствами инструментальной среды разработки, выполнение функционального тестирования.

ООО «КАЛЬДЕРА» предоставляет возможности для индивидуальной проектной деятельности, поощряя инициативу студентов в создании собственных решений. Это позволило мне реализовать проект с нуля, с учётом современных подходов к архитектуре, тестированию и контролю версий. Далее в отчёте представлена полная документация по проекту Vidify, включая техническое задание, проектные диаграммы и фрагменты исходного кода.

Для лучшего понимания структуры предприятия, на базе которого проходила практика, на рисунке 1 представлена организационная схема ООО «КАЛЬДЕРА».

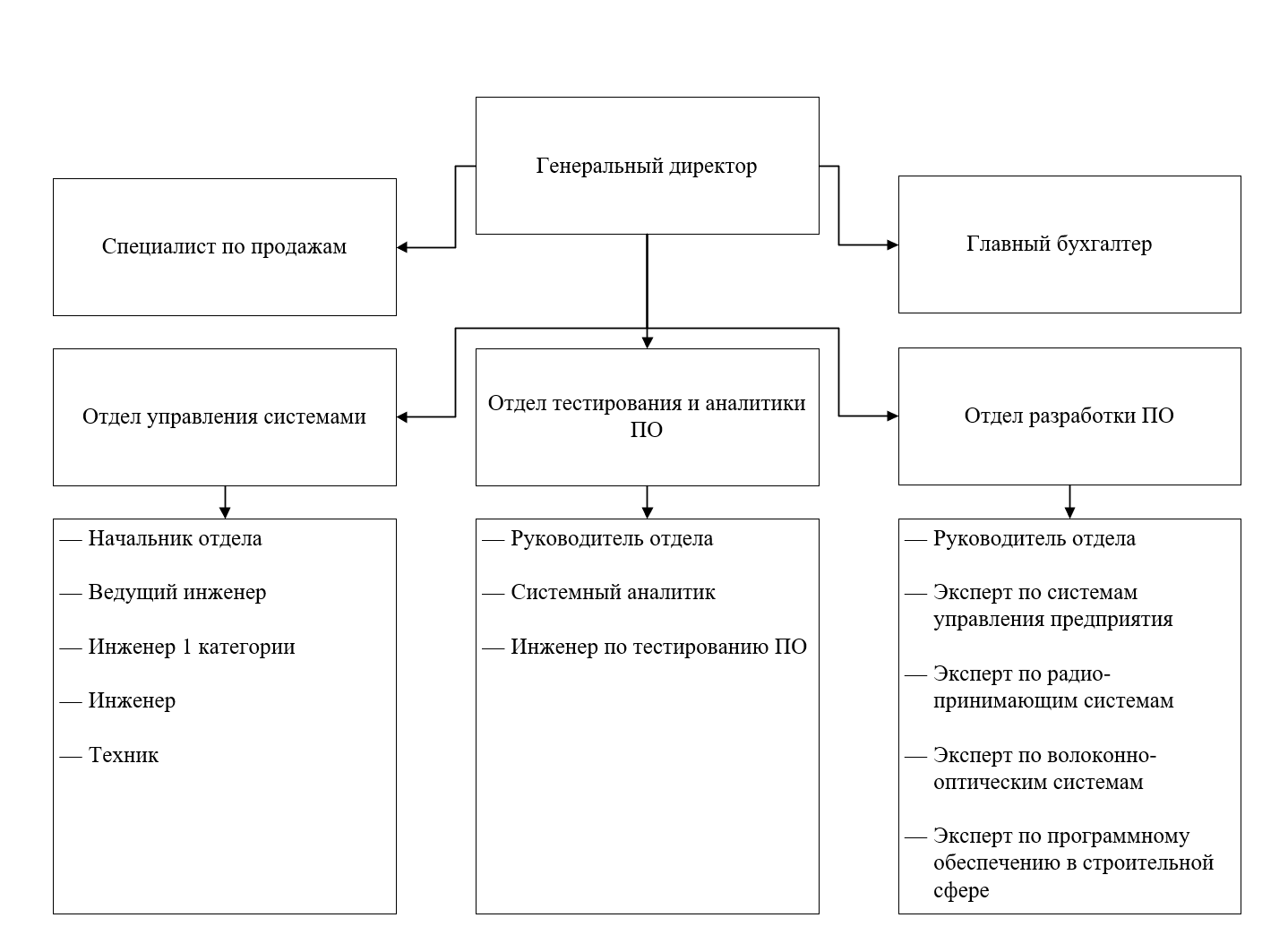


Рисунок 1 – Организационная структура ООО «КАЛЬДЕРА»

Организационная структура ООО «КАЛЬДЕРА» отражает иерархию управления и разделение функций между отделами. Каждый отдел включает специалистов, ответственных за отдельные направления, что позволяет эффективно реализовывать проекты различной сложности – от анализа требований до внедрения решений.

Центральное управление осуществляет генеральный директор, под руководством которого находятся административные и технические подразделения. Такая структура способствует распределению ответственности, чёткой специализации сотрудников и оперативному взаимодействию между отделами при реализации индивидуальных ИТ-проектов для заказчиков.

# **1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

Vidify – это программное обеспечение для автоматизации работы с видеоконтентом, предназначенное для упрощения процессов скачивания, конвертации и уникализации видео. Система ориентирована на создателей контента (блогеров, видеографов, стримеров), маркетологов и специалистов по работе с медиа, которым необходимо быстро и эффективно обрабатывать видеоматериалы для социальных сетей, рекламных кампаний или личных проектов. Приложение решает ключевые проблемы: сложность загрузки видео с платформ, необходимость адаптации форматов для разных устройств и платформ, а также потребность в уникализации контента для обхода ограничений авторских прав или создания оригинальных материалов. Vidify предоставляет интуитивный интерфейс на базе PyQt5, интеграцию с библиотеками yt-dlp и FFmpeg, а также поддержку управления файлами, что делает его комплексным инструментом для работы с видео.

Предметная область разделена на три основные подсистемы, каждая из которых выполняет специализированные функции и взаимодействует с другими для достижения общей цели.

Функции подсистемы скачивания видео:

* загрузка видео с популярных платформ (youtube, vimeo, vk) с использованием библиотеки yt-dlp;
* проверка валидности url и доступности контента;
* управление процессом загрузки (запуск, пауза, отмена);
* сохранение файлов в локальное хранилище и организация структуры папок;
* генерация превью-кадра для предпросмотра.

Входная информация:

* url-адрес видео, введенный пользователем;
* параметры загрузки (папка сохранения, формат файла).

Выходная информация:

* загруженный видеофайл в локальное хранилище (папка input);
* превью-кадр (изображение в формате jpeg);
* лог операции (успех/ошибка, время выполнения).

Функции подсистемы обработки видео:

* конвертация видео в форматы mp4, mkv, avi, mov, webm с поддержкой lossless-кодеков;
* уникализация видео (добавление рамки, обрезка, отзеркаливание, затемнение, размытие фона, наложение водяных знаков, интеграция фонового видео);
* генерация превью-кадра для предпросмотра эффектов;
* управление обработанными файлами (сохранение в папку output, организация);
* логирование операций обработки.

Входная информация:

* исходный видеофайл из локального хранилища;
* параметры обработки (выбранные эффекты, формат вывода, настройки рамки/водяного знака).

Выходная информация:

* обработанный видеофайл в папке output;
* превью-кадр с примененными эффектами;
* лог обработки (успех/ошибка, параметры, время выполнения).

Функции подсистемы пользовательского интерфейса:

* ввод данных пользователем (url, параметры загрузки/обработки);
* настройка эффектов и форматов через слайдеры, переключатели и поля ввода;
* отображение статуса операций (прогресс-бары, уведомления об ошибках);
* управление файлами (просмотр, удаление, перемещение, открытие папок);
* интеграция всех функций в единый интерфейс с вкладочной структурой.

Входная информация:

* пользовательские команды (запуск операций, выбор параметров);
* данные от подсистем downloader и video processor (логи, статусы).

Выходная информация:

* визуальное представление статуса операций (прогресс, ошибки, результаты);
* обновленное состояние интерфейса (списки файлов, превью).

Основные сценарии работы системы Vidify охватывают такие процессы, как скачивание видеоконтента, его последующая обработка и управление связанными файлами. Каждый из этих процессов представлен в виде последовательности логически выстроенных шагов, описывающих типовое взаимодействие пользователя с функциональными возможностями системы и её отдельными подсистемами.

Процесс скачивания видео реализуется подсистемой Downloader и включает следующие шаги:

1. Пользователь вводит URL видео в поле интерфейса.
2. UI передаёт URL в подсистему Downloader.
3. Downloader проверяет валидность URL с помощью VideoInfoFetcher.
4. Если URL валиден, отображается информация о видео (название, миниатюра).
5. Пользователь выбирает папку сохранения и нажимает "Скачать".
6. Downloader запускает загрузку через yt-dlp, отображая прогресс в UI.
7. При успехе файл сохраняется в папку Input, генерируется превью.
8. При ошибке (например, недоступный контент) отображается уведомление.

Обработка ошибок: Неверный URL или отсутствие интернета приводит к отображению сообщения об ошибке в статусной строке.

Процесс обработки видео выполняется подсистемой Video Processor и состоит из следующих этапов:

1. Пользователь выбирает видеофайл из папки Input через интерфейс.
2. UI открывает вкладку "Конвертер" или "Уникализация" для настройки параметров.
3. Пользователь задаёт параметры (формат, эффекты: рамка, водяной знак, обрезка).
4. Video Processor генерирует превью-кадр с помощью FFmpeg и отображает его в UI.
5. Пользователь подтверждает настройки и запускает обработку.
6. Video Processor применяет эффекты/конвертацию, сохраняя результат в папку Output.
7. UI отображает лог операции (время, успех/ошибка).

Обработка ошибок: Некорректные параметры или отсутствие FFmpeg приводят к уведомлению об ошибке.

Процесс управления файлами позволяет пользователю взаимодействовать с файлами в хранилище и включает следующие шаги:

1. Пользователь открывает список файлов в папках Input/Output через UI.
2. Выбирает действие (просмотр, удаление, открытие папки).
3. UI передаёт команду в соответствующую подсистему (Downloader/Video Processor).
4. Выполняется операция, и UI обновляет список файлов.

Система Vidify ориентирована на несколько основных категорий пользователей, для каждой из которых предусмотрен свой набор функций и возможностей при работе с приложением. Ниже приведён обзор этих пользовательских групп и соответствующих им доступных операций.

Целевая аудитория системы включает следующие группы пользователей, для которых Vidify предоставляет ключевые возможности:

* создатели контента (блогеры, видеографы, стримеры): используют уникализацию для создания оригинального контента и обхода ограничений платформ;
* маркетологи: конвертируют видео в разные форматы для рекламных кампаний и адаптации под соцсети;
* медиаспециалисты (видеоредакторы, контент-менеджеры): управляют большими объёмами видеоконтента, автоматизируя скачивание и обработку.

Система Vidify предлагает пользователям следующие возможности для работы с видеоконтентом:

* скачивание видео с youtube, vimeo, vk;
* конвертация видео в mp4, mkv, avi, mov, webm;
* уникализация (рамка, обрезка, отзеркаливание, водяные знаки, размытие фона);
* управление файлами (просмотр, удаление, организация);
* мониторинг статуса операций через прогресс-бары и логи.

Для более полного понимания внутреннего устройства системы Vidify целесообразно рассмотреть, из каких ключевых компонентов она состоит и каким образом организовано их взаимодействие.

На рисунке 1.1 представлена физическая диаграмма предметной области.

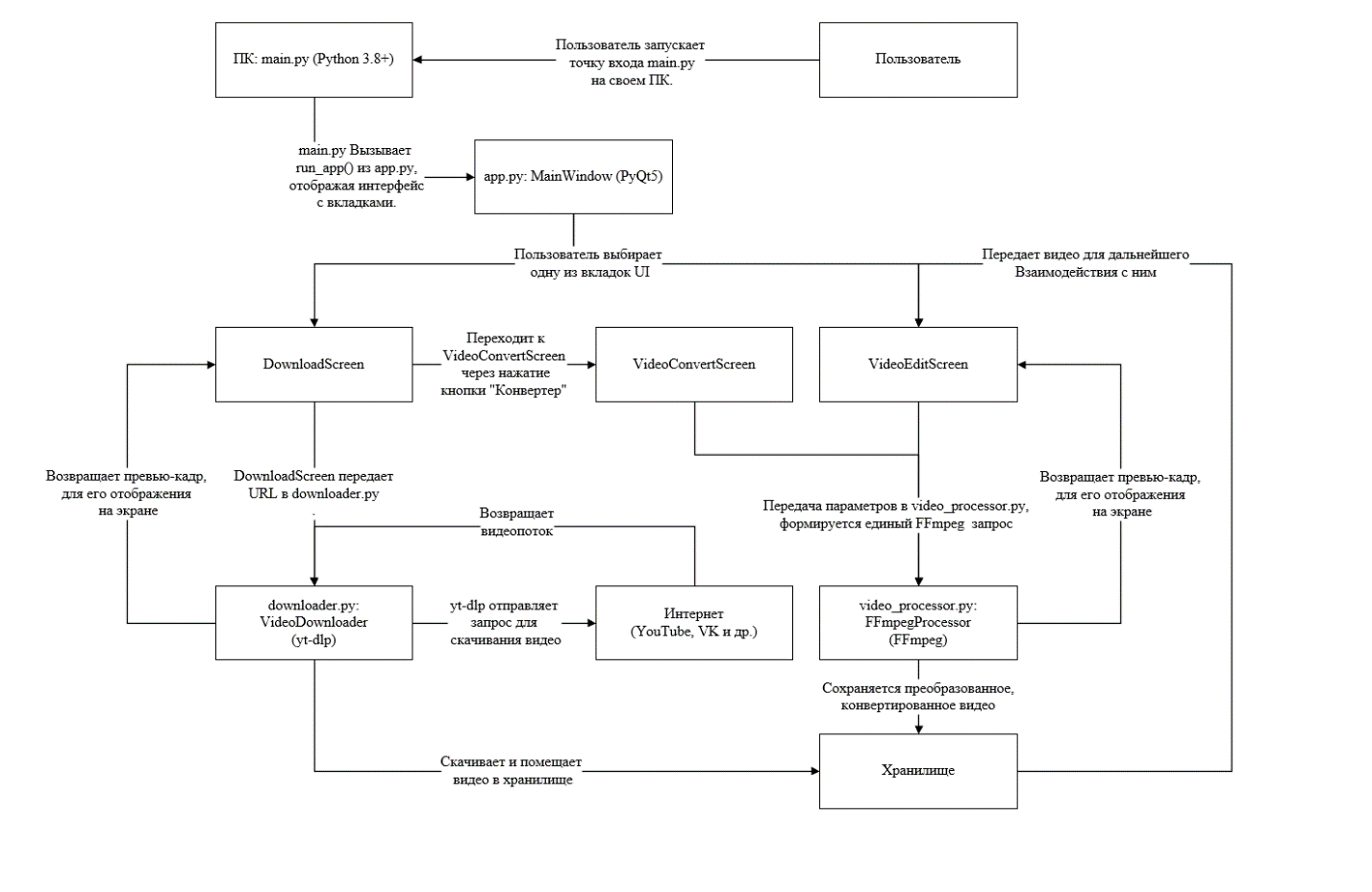


Рисунок 1.1 – Физическая диаграмма деятельности предметной области

Рассмотрев снимок экрана, мы увидели, что система состоит из следующих компонентов, взаимодействующих через ядро приложения:

* пользовательский интерфейс (pyqt5): обеспечивает ввод данных и отображение результатов;
* ядро приложения координирует работу подсистем, управляет потоками и логикой;
* downloader (yt-dlp) выполняет загрузку видео по url;
* локальное хранилище организовано в папки: input(исходные видеофайлы), output (обработанные файлы), temp(временные файлы для превью и обработки);

video processor (ffmpeg) обрабатывает видео (конвертация, уникализация).

# **2 ТРЕБОВАНИЯ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ VIDIFY**

Программное обеспечение Vidify предназначено для автоматизации процессов скачивания, конвертации и уникализации видеоконтента. Целевая аудитория — это создатели контента (блогеры, видеографы, стримеры), маркетологи и медиаспециалисты. Система предоставляет интуитивный графический интерфейс на базе библиотеки PyQt5, интеграцию с yt-dlp и FFmpeg, а также инструменты управления файлами для оптимизации работы с видео.

Функциональные требования разделяются по подсистемам: скачивания видео, обработки видео, пользовательского интерфейса и управления файлами.

Подсистема отвечает за загрузку видеоконтента с различных платформ по ссылке, предоставленной пользователем. Ниже перечислены основные требования:

1. Система должна обеспечивать скачивание видео с платформ YouTube, Vimeo и VK по введённому URL.
2. Система должна проверять корректность URL до начала загрузки и отображать сообщение об ошибке при недопустимом адресе или недоступном контенте.
3. Система должна позволять пользователю выбрать папку для сохранения загруженных файлов.
4. Интерфейс должен поддерживать управление загрузкой: запуск, приостановку и отмену процесса.
5. Для каждого загружаемого видео должно автоматически генерироваться превью-изображение (JPEG) и отображаться в интерфейсе.
6. Загруженные видеофайлы должны сохраняться в папке Input локального хранилища с соответствующей организацией файлов.
7. Должен вестись лог загрузок (успешные и ошибочные операции, время выполнения) с отображением в пользовательском интерфейсе.

Данная подсистема отвечает за конвертацию видеофайлов в другие форматы и применение эффектов уникализации. Требования к функциональности следующие:

1. Система должна обеспечивать конвертацию видео в форматы MP4, MKV, AVI, MOV и WEBM с использованием кодеков без потери качества.
2. Пользователь должен иметь доступ к функциям уникализации видео, таким как:
   * добавление рамки с настраиваемыми цветом и толщиной;
   * обрезка видео по заданному временному диапазону;
   * зеркалирование (горизонтальное или вертикальное);
   * затемнение или размытие фона;
   * наложение водяных знаков (текста или изображения);
   * добавление фонового видео.
3. До начала обработки система должна отображать превью-кадр с применёнными эффектами для подтверждения пользователем.
4. Обработанные файлы должны сохраняться в папке Output с упорядочиванием по дате и/или имени.
5. Все действия по обработке видео должны логироваться (статус, параметры, длительность) и отображаться в интерфейсе.
6. После завершения обработки система должна удалять временные файлы из папки Temp.

Интерфейс пользователя обеспечивает взаимодействие с функциональностью приложения. Ниже представлены основные требования к его структуре и элементам управления:

1. Интерфейс должен иметь вкладочную структуру, включающую разделы:
   * «скачивание» – для ввода ссылок и управления загрузкой;
   * «конвертер» – для задания параметров конвертации;
   * «уникализация» – для выбора эффектов обработки.
2. Элементы управления должны включать текстовые поля, слайдеры, переключатели и кнопки.
3. Система должна отображать индикаторы прогресса (progress bars) для процессов загрузки и обработки.
4. Интерфейс должен выводить уведомления о завершении операций или возникновении ошибок.
5. В интерфейсе должно быть реализовано управление файлами: просмотр, удаление и открытие папок Input и Output.
6. Файлы в папках Input и Output должны отображаться в виде списка с возможностью сортировки по имени, размеру и дате.

Управление файлами – это вспомогательная функциональность, обеспечивающая удобную навигацию и организацию хранимых данных. К ней предъявляются следующие требования:

1. Система должна отображать список файлов в папках Input и Output с указанием имени, размера и даты создания.
2. Пользователь должен иметь возможность удалять отдельные файлы из указанных папок через интерфейс.
3. Интерфейс должен позволять открытие папок Input и Output в стандартном проводнике операционной системы.

Нефункциональные требования же включают в себя характеристики производительности, надежности, удобства использования, совместимости и безопасности.

Система должна обеспечивать приемлемую скорость работы на среднестатистическом пользовательском оборудовании:

1. Загрузка видео в разрешении до 4K должна занимать от 1 до 5 минут при скорости интернет-соединения 10 Мбит/с.
2. Конвертация видео длительностью 5 минут (1080p) должна выполняться за 1–3 минуты на процессоре уровня Intel Core i5 10-го поколения.
3. Уникализация аналогичного видео (например, наложение рамки или водяного знака) должна занимать 1–2 минуты.

Программа должна быть устойчива к сбоям и корректно реагировать на ошибки:

1. При возникновении ошибок (неверный URL, отсутствие интернета, некорректные параметры) система должна выдавать информативные сообщения.
2. При сбоях (например, прерывании загрузки) система должна сохранять целостность файлов.

Интерфейс должен быть доступным и понятным:

1. Интерфейс должен быть интуитивным, поддерживать тёмную тему и адаптацию к разрешениям экрана от 1280×720 до 4K.
2. Все элементы управления должны быть легко распознаваемыми, без необходимости чтения документации.

Система должна быть кроссплатформенной и использовать актуальные технологии:

1. Совместимость обеспечивается с Windows 10/11, macOS 12 и выше, Linux (Ubuntu 20.04+).
2. Используемые библиотеки (yt-dlp, FFmpeg) должны быть последними стабильными версиями на момент разработки.
3. Обработка видеофайлов должна поддерживаться для размеров до 10 ГБ.

Особое внимание уделяется сохранности данных пользователя:

1. Система не должна сохранять пользовательские данные (URL, параметры) после завершения работы.
2. Вся работа с данными должна выполняться локально, без передачи на внешние серверы.

В процессе эксплуатации системы необходимо учитывать ряд ограничений:

1. Программа зависит от наличия установленных библиотек yt-dlp и FFmpeg.
2. Для загрузки видео требуется стабильное подключение к интернету.
3. Максимальный объём обрабатываемого файла ограничен объёмом доступной оперативной памяти (рекомендуется минимум 8 ГБ).

В рамках разработки системы Vidify были сформулированы некоторые базовые предположения, касающиеся предполагаемых условий её эксплуатации. Эти допущения отражают обстановку, в которой программа, как ожидается, будет использоваться, и служат ориентиром при проектировании функциональности и структуры приложения:

1. Пользователи обладают базовыми компьютерными навыками (ввод URL, выбор папки и т. д.).
2. Устройство пользователя соответствует минимальным требованиям: 4 ГБ оперативной памяти, 2 ГБ свободного места на диске.

# **3 РАЗРАБОТКА ТЕХНИЧЕСКОГО ЗАДАНИЯ**

В современном цифровом пространстве обработка видеоконтента становится одной из ключевых задач для создателей медиа. С развитием социальных сетей и видеохостингов возросла потребность в быстрой, автоматизированной и удобной уникализации видеофайлов.

Данное техническое задание направлено на разработку программного продукта Vidify, предназначенного для обработки и уникализации видео. Приложение позволяет загружать видеоролики, применять к ним преобразования и сохранять в нужном формате. В рамках разработки рассматриваются вопросы проектирования функциональных модулей, построения интерфейса и определения целевой аудитории.

Предметная область охватывает все процессы, связанные с автоматизированной обработкой видеоконтента, в том числе:

* + 1. Загрузка видеофайлов:
* получение видео по ссылке (YouTube, VK);
* проверка корректности введённого URL;
* управление загрузкой.
  + 1. Обработка видео:
* добавление рамки;
* зеркалирование изображения;
* наложение водяного знака;
* эффект затемнения;
* генерация превью.
  + 1. Управление файлами:
* организация входных и выходных файлов;
* сохранение и удаление результатов;
* управление временными файлами.
  + 1. Интерфейс пользователя:
* настройка параметров обработки;
* отображение прогресса;
* интуитивно понятная навигация.

Программное обеспечение может использоваться как в личных целях (блогерами и видеографами), так и в профессиональной среде – для массовой обработки и стандартизации видеоматериалов.

Программный продукт ориентирован на следующие категории пользователей:

* + 1. Основная аудитория:
  + создатели контента (блогеры, стримеры, монтажёры):  
    возраст 18–35 лет, уверенные навыки работы с видео, регулярная необходимость в уникализации контента.
  + маркетологи: возраст 25–45 лет, опыт работы с медиа, заинтересованность в быстром преобразовании и оптимизации видео под разные платформы.
    1. Дополнительная аудитория:
* видеографы-любители, студенты профильных специальностей, малый бизнес.
  + 1. Техническая аудитория:
  + разработчики, системные администраторы, специалисты по автоматизации рабочих процессов с видеофайлами.

Модель пользователя – Создатель контента:

* + 1. Социально-демографические характеристики:
* возраст: 25–30 лет;
* образование: высшее техническое;
* опыт работы с видео: 2–3 года.
  + 1. Уровень компьютерной грамотности:
* уверенный пользователь;
* владение программами обработки видео (adobe premiere, davinci resolve);
* знание основ командной строки и ffmpeg.
  + 1. Цели и задачи:
* быстрая уникализация видеоконтента;
* оптимизация рабочего процесса;
* массовая обработка однотипных роликов.
  + 1. Рабочее окружение:
* домашний офис;
* операционная система: windows 10;
* мощный персональный компьютер.
  + 1. Требования к системе:
* простой и понятный интерфейс;
* минимальное количество ручных действий;
* высокая скорость и качество обработки.

Сейчас мы рассмотри предполагаемые сценарии использования нашего приложения:

Сценарий 1 – Загрузка видео:

1. Пользователь запускает приложение;
2. Вводит ссылку на источник видео;
3. Выбирает папку для сохранения;
4. Инициирует загрузку;
5. Получает уведомление об успешном завершении.

Сценарий 2 – Уникализация видео:

1. Пользователь выбирает файл из списка;
2. Настраивает параметры (эффекты, водяные знаки);
3. Запускает обработку;
4. Просматривает результат в превью;
5. Сохраняет итоговый файл.

Сценарий 3 – Управление файлами:

1. Открывает список всех загруженных видео;
2. Сортирует и выбирает нужные;
3. Удаляет, перемещает или открывает в проводнике;
4. Контролирует свободное пространство и версионность.

Структурно-функциональная схема интерфейса системы Vidify показывает основные компоненты и связи между ними.

Рассмотрим рисунок 3.1, на представлена схема пользовательского интерфейса приложения Vidify, отображающая структуру экранов и переходов между ними.

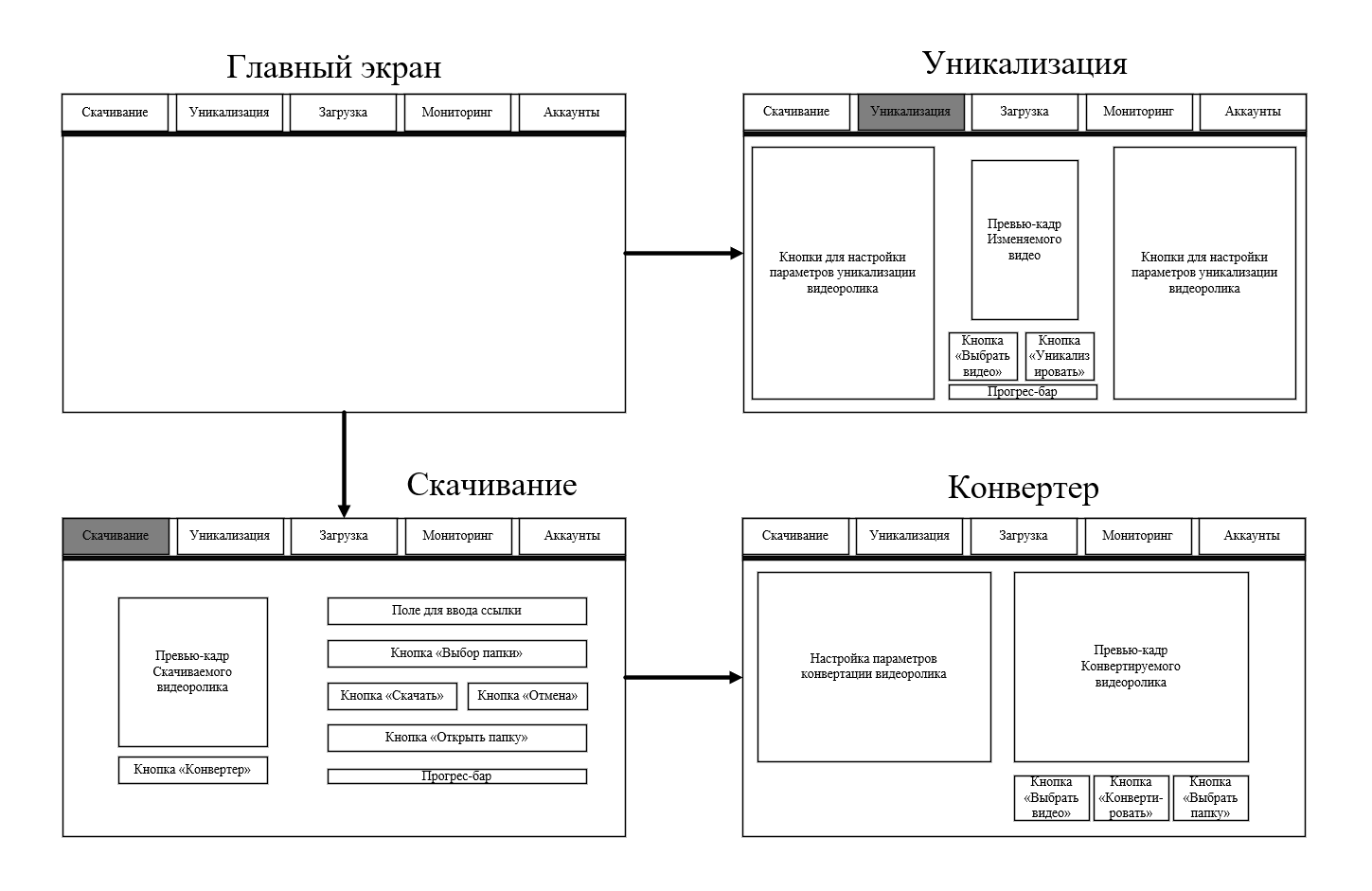


Рисунок 3.1 – Структура интерфейса и взаимодействие экранов приложения Vidify

Интерфейс включает пять основных вкладок, представленных в виде верхнего меню: Скачивание, Уникализация, Загрузка, Мониторинг, Аккаунты. Каждая вкладка содержит функциональные блоки:

* скачивание (ввод ссылки, выбор папки, кнопки загрузки и просмотра, прогресс-бар);
* уникализация (кнопки настройки эффектов, превью результата, запуск обработки);
* конвертер (параметры конвертации, выбор формата, итоговое сохранение);
* нереализованные вкладки (загрузка, мониторинг, аккаунты).

Такое разбиение обеспечивает логичную навигацию, повышает удобство работы и минимизирует вероятность ошибок пользователя.

Разработка программного обеспечения Vidify позволяет автоматизировать процессы уникализации и обработки видеоконтента для различных категорий пользователей. Анализ предметной области, определение целевой аудитории, построение пользовательских сценариев и проектирование интерфейса обеспечивают комплексный подход к созданию системы.

В результате использования Vidify пользователи получают удобный, гибкий и производительный инструмент, способствующий улучшению качества контента и сокращению временных затрат на его обработку.

# **4 РАЗРАБОТКА СТРУКТУРЫ ПРОЕКТА**

Разрабатываемое программное обеспечение Vidify предназначено для автоматизации работы с видеоконтентом, включая загрузку, конвертацию и уникализацию видеофайлов. Ниже представлены основные предложения по созданию программного обеспечения:

* 1. Модульная архитектура – проект будет разделён на функциональные подсистемы: Downloader, Video Processor, User Interface, что обеспечит гибкость разработки и удобство в сопровождении;
  2. Технологический стек:
* язык программирования: python (для кроссплатформенности и широкой поддержки библиотек);
* библиотеки: pyqt5 (разработка интерфейса), yt-dlp (загрузка видео), ffmpeg (обработка видео);
* система контроля версий: git с использованием tortoisegit и репозитория на github.
  1. Пользовательский интерфейс – вкладочная структура с разделами «Скачивание», «Конвертер», «Уникализация», что обеспечивает интуитивное взаимодействие пользователя с программой;
  2. Хранилище данных – использование локальных папок Input, Output, Temp для управления файлами на каждом этапе обработки;
  3. Методология разработки – итеративный (agile-подобный) подход с короткими циклами обратной связи.

Разработка программного продукта Vidify планируется в рамках учебной практики и рассчитана на восемь недель. Каждый этап имеет определённые цели и задачи:

Неделя 1. Анализ и планирование (5 часов):

* изучение предметной области;
* формулировка требований;
* согласование технического задания.

Неделя 2. Проектирование (5 часов):

* разработка модульной архитектуры;
* создание прототипа интерфейса с использованием pyqt5;
* настройка репозитория github с помощью tortoisegit.

Недели 3–4. Разработка подсистемы Downloader (10 часов):

* интеграция библиотеки yt-dlp;
* реализация проверки url, загрузки видео, генерации превью;
* тестирование загрузки с популярных платформ (youtube, vk и др.).

Недели 5–6. Разработка подсистемы Video Processor (10 часов):

* интеграция ffmpeg;
* реализация конвертации в различные форматы (mp4, mkv, avi и др.);
* реализация эффектов уникализации (рамки, водяные знаки, зеркалирование);
* тестирование обработки видео.

Неделя 7. Разработка пользовательского интерфейса и интеграция (5 часов):

* создание интерфейсных компонентов: вкладок, прогресс-баров, списка файлов;
* интеграция всех модулей в единую систему;
* проведение функционального тестирования.

Неделя 8. Финальное тестирование и подготовка документации (5 часов):

* проведение системного тестирования;
* устранение выявленных дефектов;
* подготовка пользовательской документации и итогового отчета.

Общая трудоёмкость проекта составляет 40 часов.

В рамках учебного проекта оценка стоимости производится условно, исходя из затраченного времени и минимальных ресурсов:

* 1. Зарплатные издержки:
* программист (1 чел.): 40 часов × 500 руб./час = 20 000 руб.;
* тестировщик (0,5 ставки): 10 часов × 400 руб./час = 4 000 руб.
  1. Программные и технические ресурсы:
* используемые инструменты: python, pyqt5, yt-dlp, ffmpeg (свободно распространяемые);
* хостинг репозитория: github (бесплатный план);
* оборудование: предполагается наличие пк (intel core i5, 8 гб ram).

Итого: 24 000 рублей (без учёта стоимости оборудования).

Для реализации проекта требуется минимальный состав участников. В учебном контексте часть ролей совмещается:

* + 1. Программист:
* навыки: python, pyqt5, работа с yt-dlp и ffmpeg, опыт работы с git;
* задачи: реализация всех программных модулей, интеграция, настройка репозитория.
  + 1. Тестировщик (на полставки):
* навыки: базовое тестирование программного обеспечения;
* задачи: составление сценариев тестирования, проверка корректности работы.
  1. Куратор проекта (преподаватель):
* функции: контроль выполнения этапов, консультирование по гост и техническому заданию.

Контроль осуществляется через несколько каналов:

* + 1. Еженедельные отчёты:
* программист предоставляет краткое описание проделанной работы и демонстрирует результаты.
  + 1. Использование системы контроля версий (Git):
* ведение репозитория на github;
* применение tortoisegit для управления версиями;
* регулярные коммиты (не менее 2–3 в неделю);
* проверка журналов изменений через интерфейс github.
  + 1. Этапное тестирование:
  + после завершения каждого этапа проводится частичное тестирование (например, проверка модуля загрузки после недели 4);
  + финальное тестирование осуществляется на 8-й неделе.
  1. Контрольные точки проекта:
  + неделя 2 – утверждение технического задания и прототипа;
  + неделя 4 – демонстрация работающего модуля загрузки;
  + неделя 6 – презентация модуля обработки видео;
  + неделя 8 – защита проекта и итогового отчета.

Для ведения разработки используется система контроля версий Git. Основные действия по организации работы включают:

* + 1. Первичная настройка:
  + установка git и tortoisegit на пк;
  + создание и настройка удалённого репозитория github;
  + связывание локального и удалённого репозиториев.
    1. Рабочий процесс:
  + создание отдельной ветки dev для разработки;
  + ведение коммитов после завершения каждой задачи;
  + отправка изменений в удалённый репозиторий (git push);
  + создание pull request'ов (для имитации командной работы и проверки кода).
    1. Примеры операций в Git:
  + git commit – фиксация изменений (например, добавление обработчика загрузки);
  + git push – отправка изменений в репозиторий github;
  + git pull – получение обновлений;
  + удаление временных файлов – через интерфейс tortoisegit.

## **5 РАБОТА В СИСТЕМЕ КОНТРОЛЯ ВЕРСИЙ**

Для работы с системой контроля версий используется Git с графическим интерфейсом TortoiseGit и хостингом репозитория на GitHub. Репозиторий создается с нуля для проекта Vidify.

* + 1. Установка инструментов:
  + установлен git (https://git-scm.com/downloads);
  + установлен tortoisegit (https://tortoisegit.org/download/);
  + зарегистрирован аккаунт на github (https://github.com/sorryharam).
    1. Создание репозитория:
  + на github создан новый репозиторий с названием vidify (публичный, с файлом readme.md);
  + url репозитория: https://github.com/sorryharam/vidify.

Рассмотрим процесс создания репозитория на GitHub.

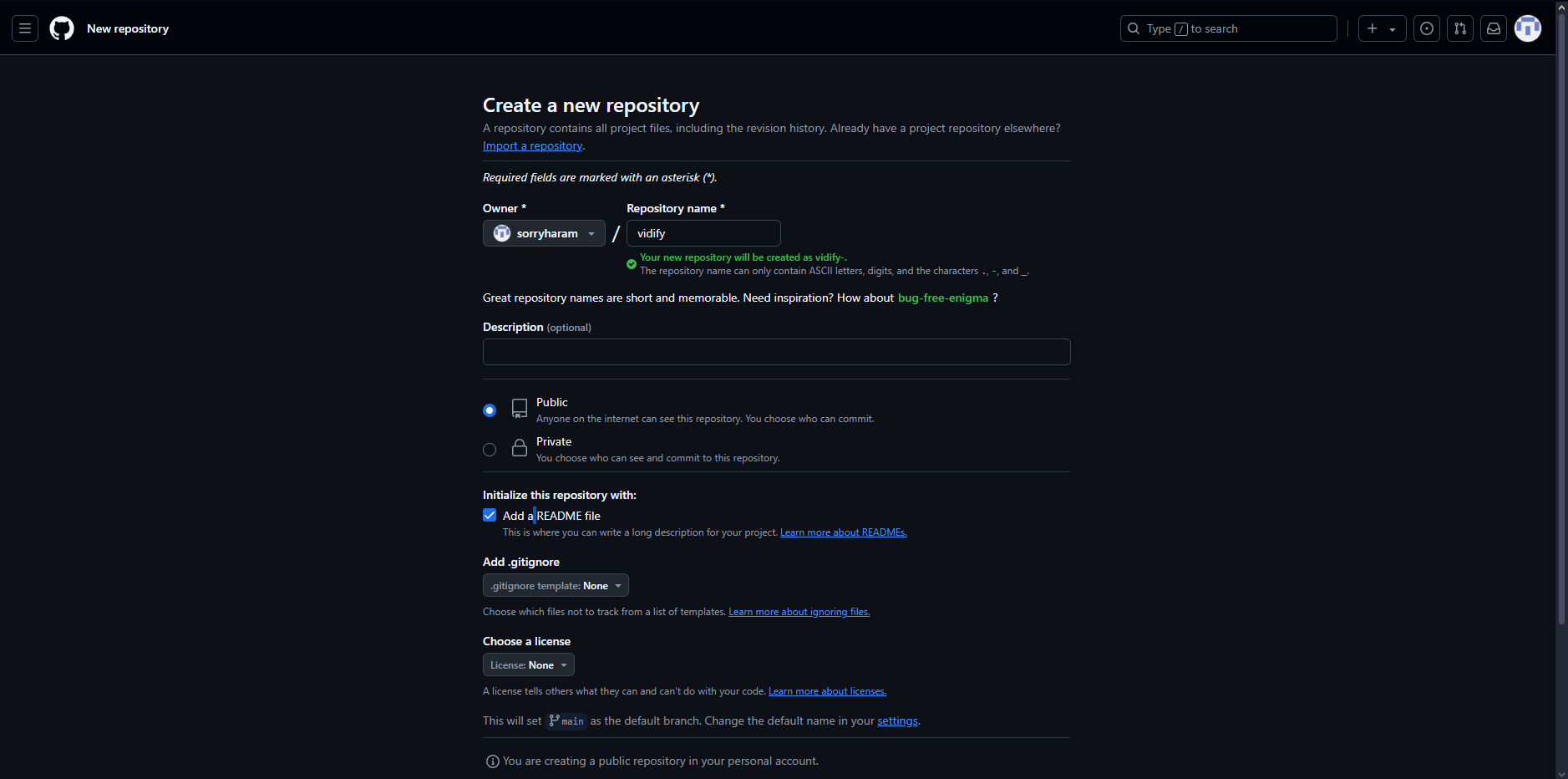


Рисунок 5.1 – Создание репозитория на GitHub

На изображении показано окно GitHub с созданием репозитория vidify, включенным файлом README.md и публичным доступом.

Инициализация локального репозитория:

* 1. Открыта папка проекта vidify/ в проводнике.
  2. Выполнен щелчок правой кнопкой мыши, выбран пункт «git create repository here» в меню tortoisegit.
  3. Подтверждено создание .git.

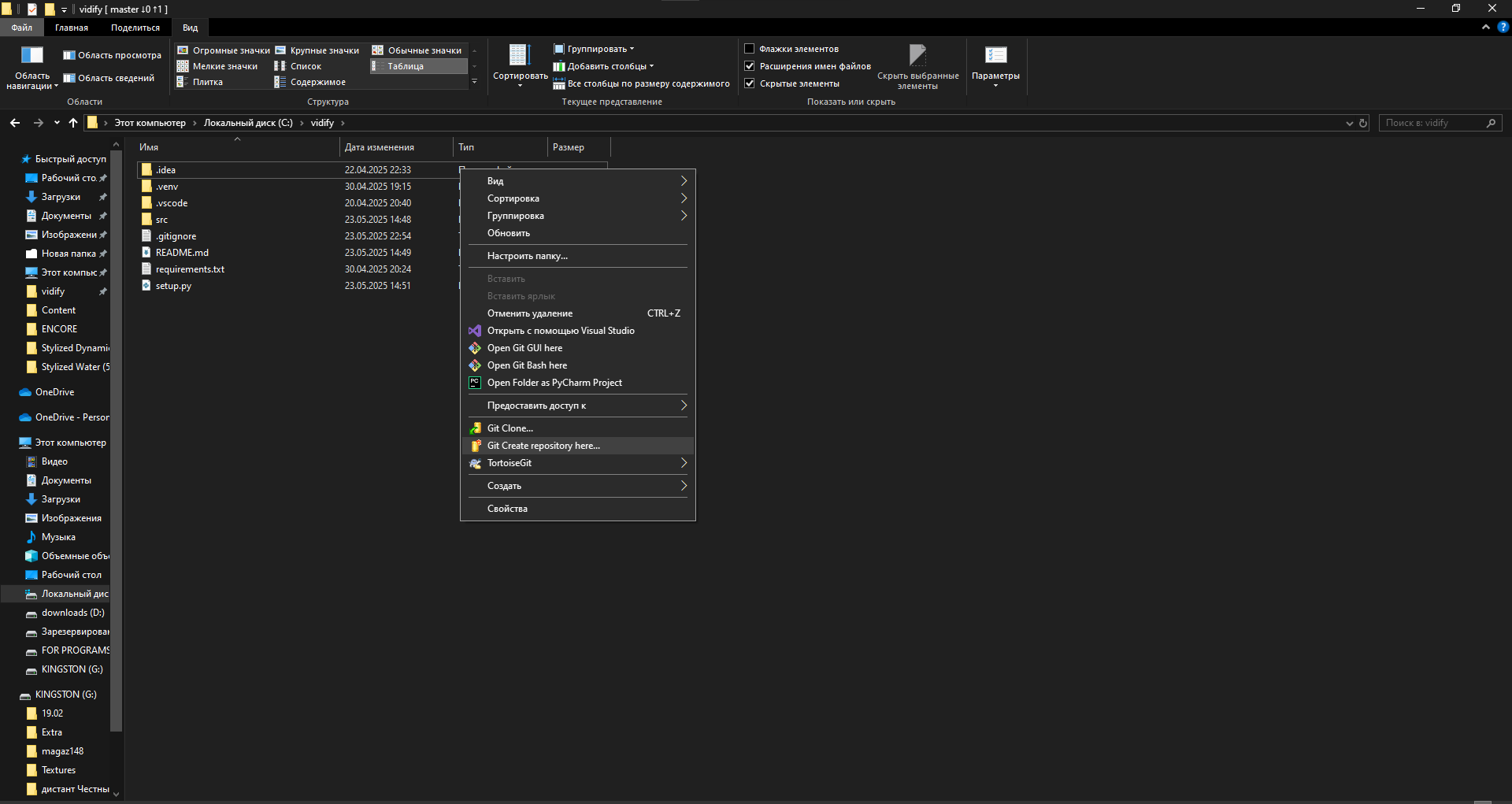
Рассмотрим рисунок 5.2, где показана локальная папка нашего проекта.

Рисунок 5.2 – Инициализация локального репозитория через TortoiseGit

На изображении показано контекстное меню TortoiseGit с выбранным пунктом «Git Create repository here» в папке проекта vidify.

Подключение к удаленному репозиторию:

* 1. В папке vidify/ выполнен щелчок правой кнопкой мыши, выбран пункт «TortoiseGit/Settings/Git/Remote».
  2. Добавлен удаленный репозиторий:
     + имя: origin;
     + url: https://github.com/sorryharam/vidify.
  3. Настройки сохранены.

Рассмотрим рисунок 5.3 на котором изображена настройка удаленного репозитория.

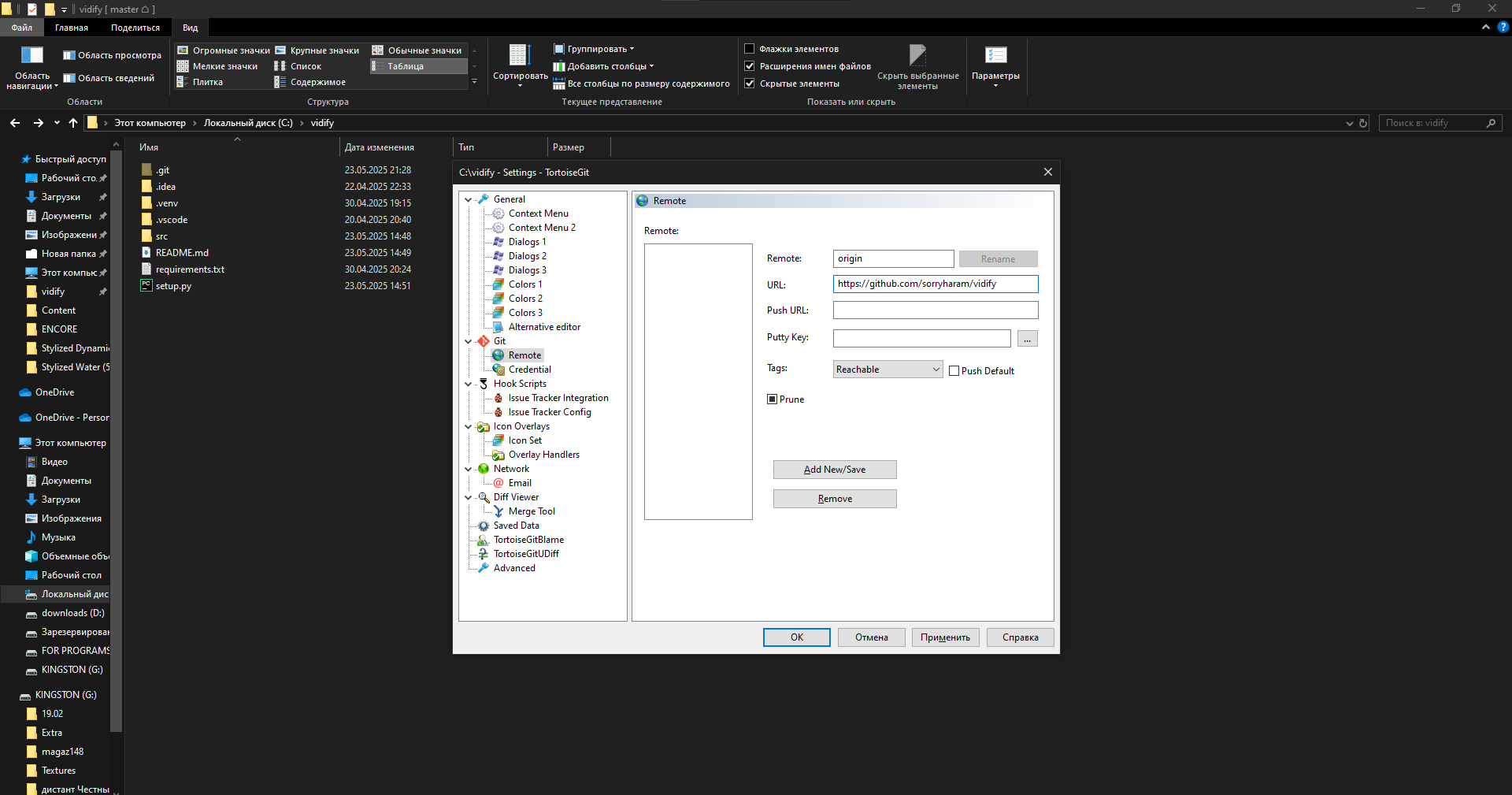


Рисунок 5.3 – Настройка удаленного репозитория origin

На изображении показано окно TortoiseGit в разделе «Settings/Git/Remote», где добавлен репозиторий origin с URL https://github.com/sorryharam/vidify.

Теперь мы рассмотрим отправка всех локальных файлов в репозиторий. Локальные файлы проекта Vidify добавляются в репозиторий, чтобы он не был пустым:

1. Подготовка файлов:
   * проверена структура проекта в папке vidify.
2. Добавление файлов:
   * в папке vidify/ выполнен щелчок правой кнопкой мыши, выбран пункт «tortoisegit/add»;
   * выбраны файлы: .gitignore, readme.md, requirements.txt, setup.py, src/.
3. Коммит файлов:
   * выполнен щелчок правой кнопкой, выбран пункт «tortoisegit/commit/master»;
   * введено сообщение: «initial commit with project files»;
   * подтвержден коммит.

Рассмотрим рисунок 5.4 на котором продемонстрирован коммит файлов.

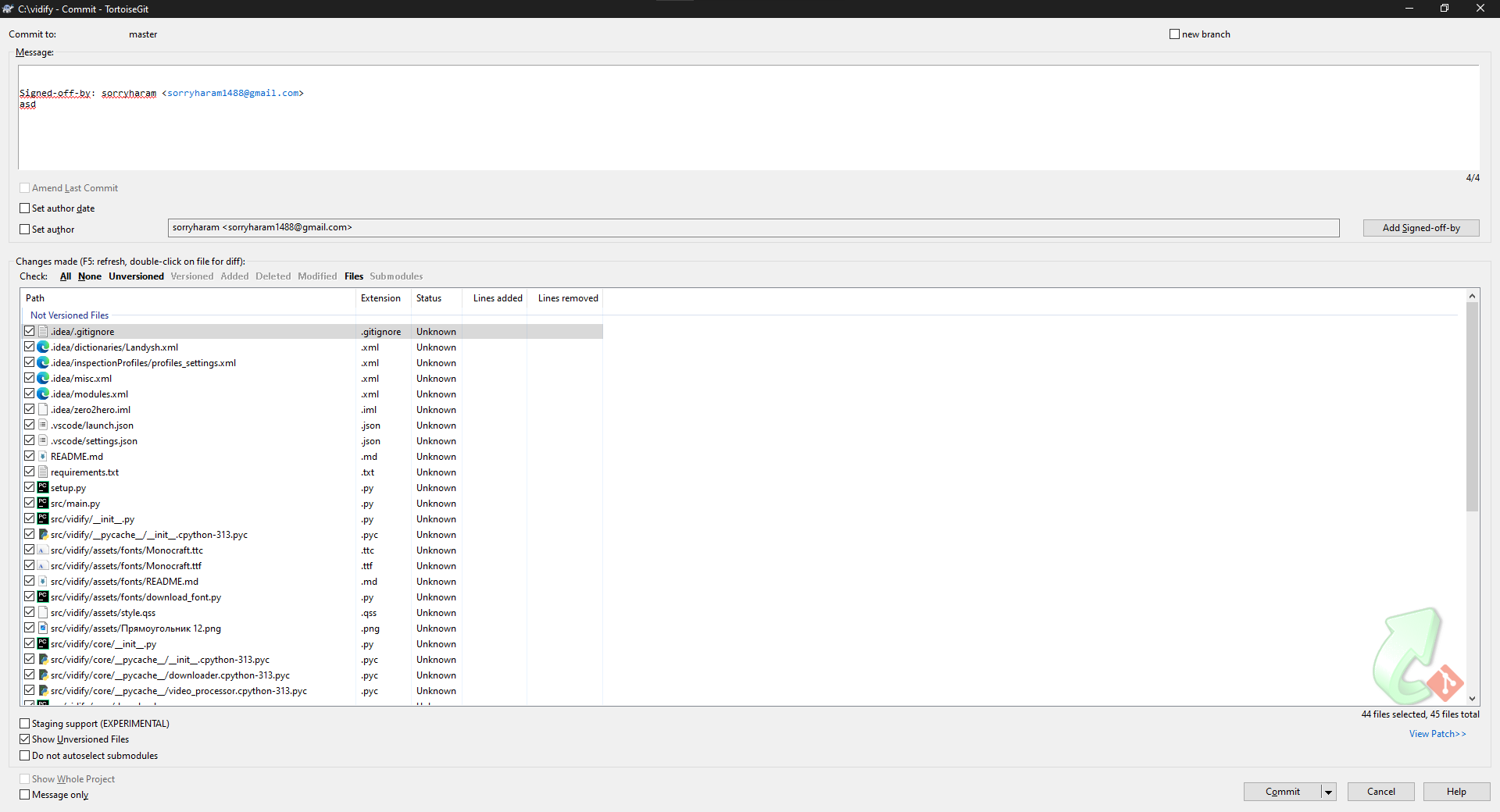


Рисунок 5.4 – Коммит файлов в ветку master

На изображении показано окно TortoiseGit с выбранными файлами для коммита в ветку master и сообщением «Initial commit with project files».

Отправка в репозиторий:

* + выполнен щелчок правой кнопкой, выбран пункт «tortoisegit/sync»;
  + выбрана ветка master, удаленная ветка main, установлен флаг «force»;
  + нажата кнопка «push».

Рассмотрим рисунок 5.5 на котором изброжена отправка изменений.

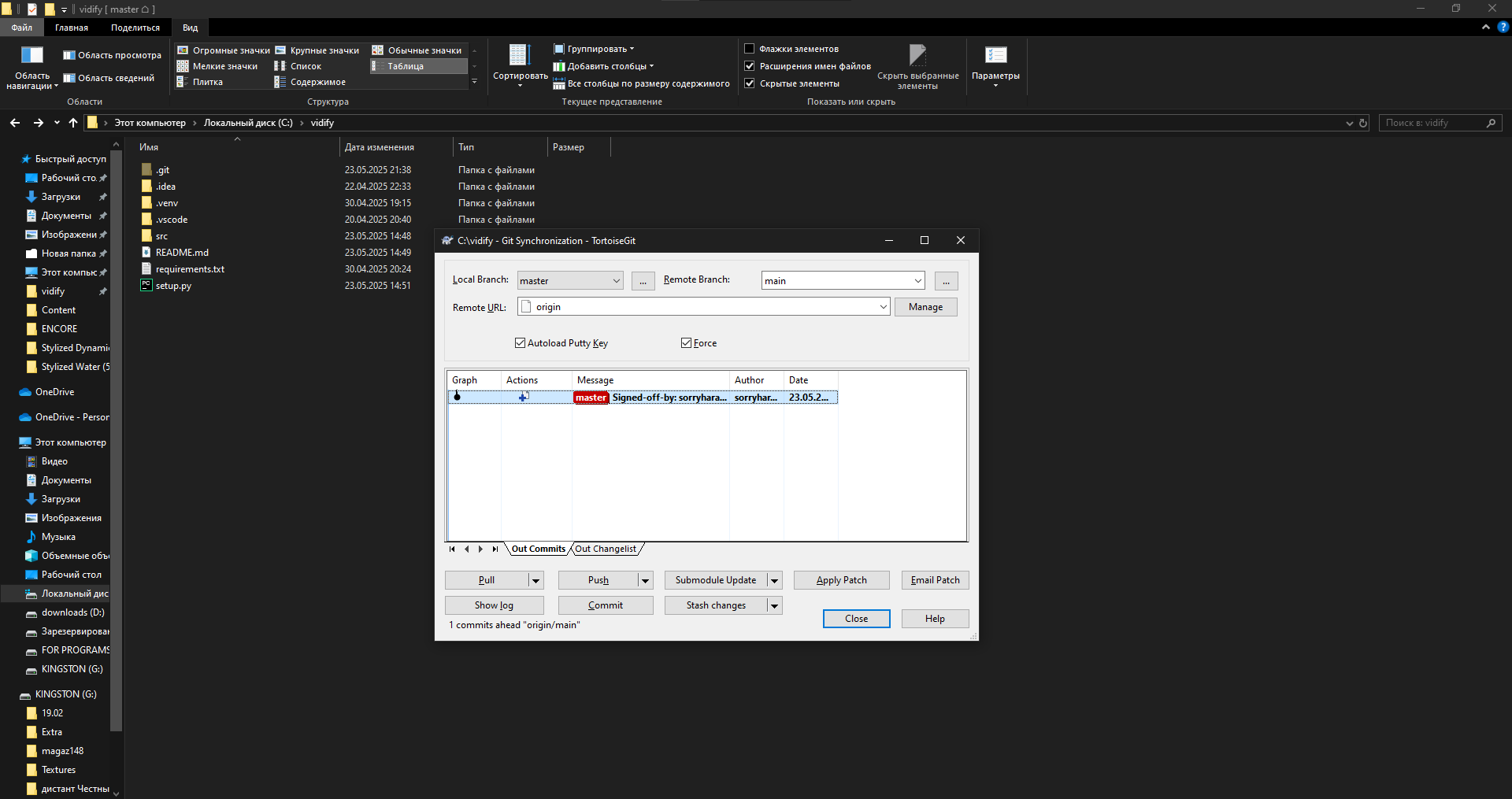


Рисунок 5.5 – Синхронизация и push в репозиторий

На изображении показано окно TortoiseGit Sync с выбранной веткой master, удаленной веткой main, установленным флагом «Force» и нажатой кнопкой «Push».

Рассмотрим рисунок 5.6 на котором показано содержимое репозитория.

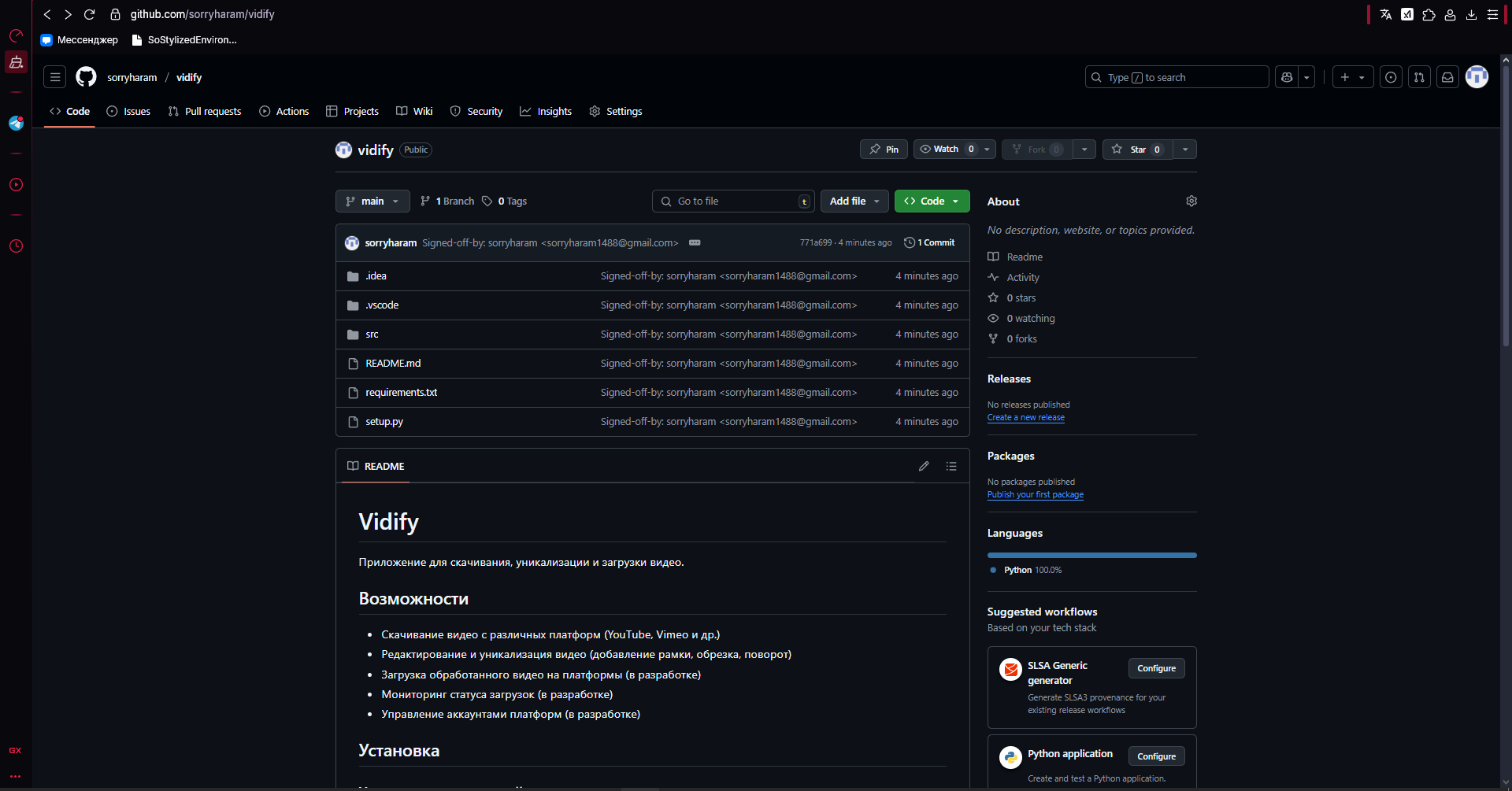


Рисунок 5.6 – Просмотр файлов в ветке main на GitHub

На изображении показана страница GitHub с файлами проекта vidify в ветке main, включая .gitignore, src/, requirements.txt и другие.

Перейдем к скачиванию репозитория. Проект клонируется на локальный компьютер для проверки содержимого. Логика выполнения аналогична действиям по отправке файлов в пункт 2, ранее выполненным.

Клонирование репозитория:

* + создана новая папка (например, c:\projects\vidify-clone);
  + выполнен щелчок правой кнопкой, выбран «git clone» в меню tortoisegit;
  + введен url: https://github.com/sorryharam/vidify;
  + подтверждено клонирование, восстановлена структура проекта (.gitignore, src/, requirements.txt).

Рассмотрим рисунок 5.7 на котором показано клонирование репозитория.

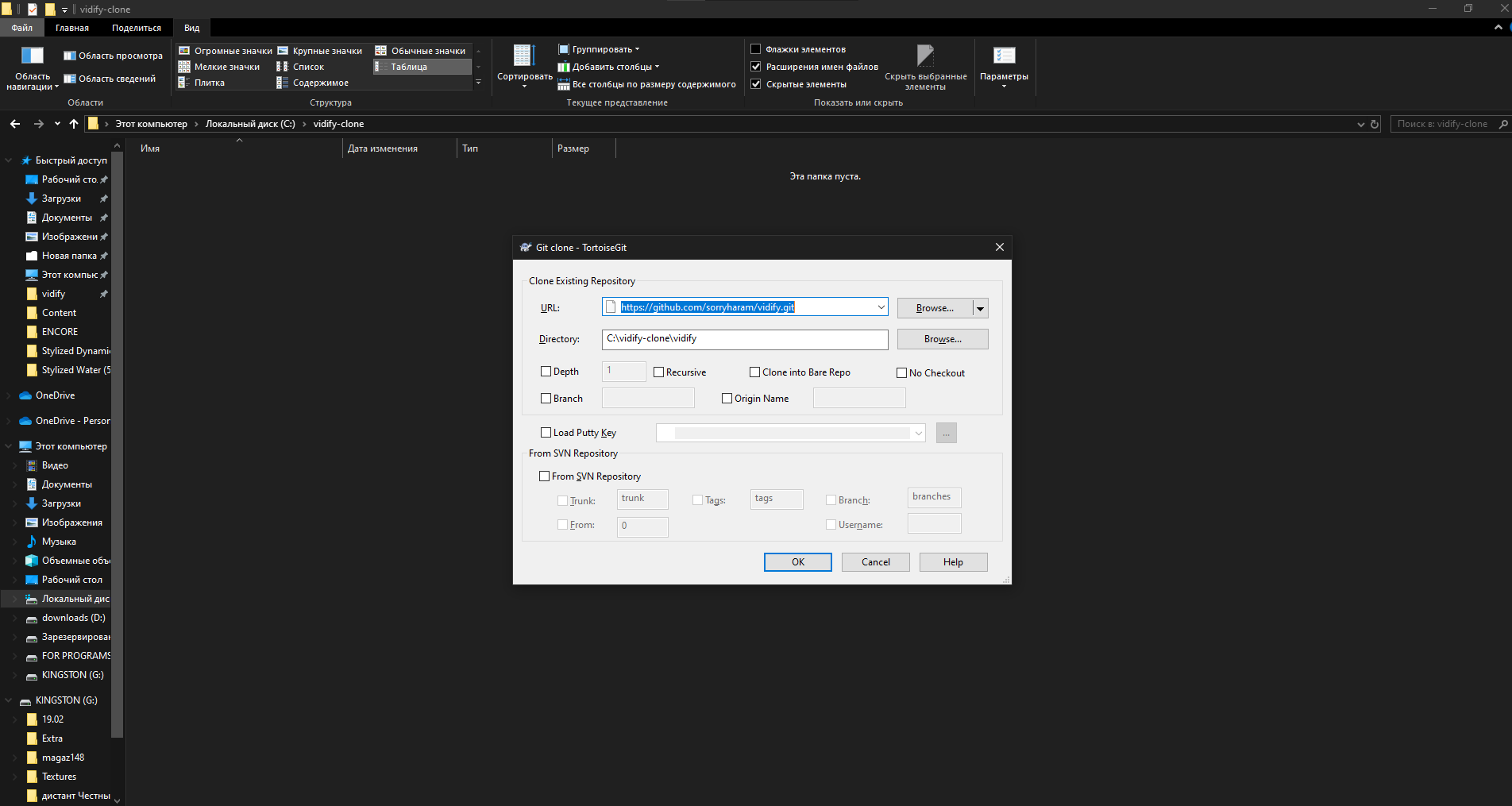


Рисунок 5.7 – Клонирование репозитория через TortoiseGit

На изображении показано окно TortoiseGit с выбранным пунктом «Git Clone» и введенным URL https://github.com/sorryharam/vidify.

Добавим новый класс и подробнее рассмотрим изменения, внесённые на данном этапе по сравнению с предыдущей версией проекта.

Добавлен новый файл utils/helper.py с классом Helper.

1. Создание файла:
   * в папке src/vidify/utils/ создан файл helper.py;
   * класс содержит метод для очистки имен файлов, полезный для подсистемы downloader.
2. Добавление в репозиторий:
   * в папке vidify/ выполнен щелчок правой кнопкой, выбран «tortoisegit/add»;
   * выбран файл src/vidify/utils/helper.py;
   * выполнен коммит.

Рассмотрим рисунок 5.8 на котором изображен коммит класса Helper.

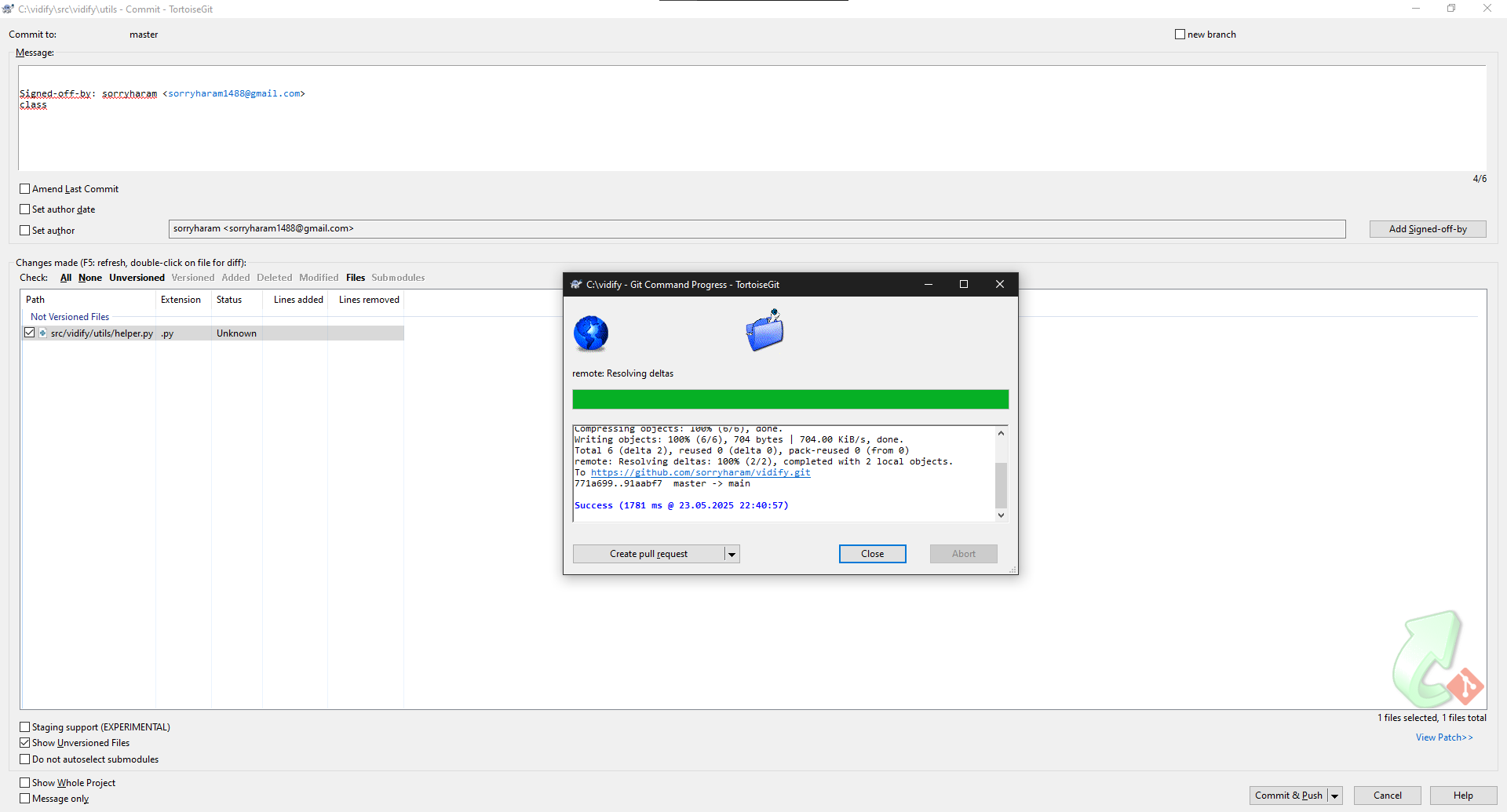


Рисунок 5.8 – Коммит класса Helper в ветку master

На изображении показано окно TortoiseGit с коммитом файла helper.py в ветку master и сообщением «Added Helper class». Изменения отправлены через «TortoiseGit/Push» в ветку main.

Рассмотрим рисунок 5.9 на котором показана страница GitHub.

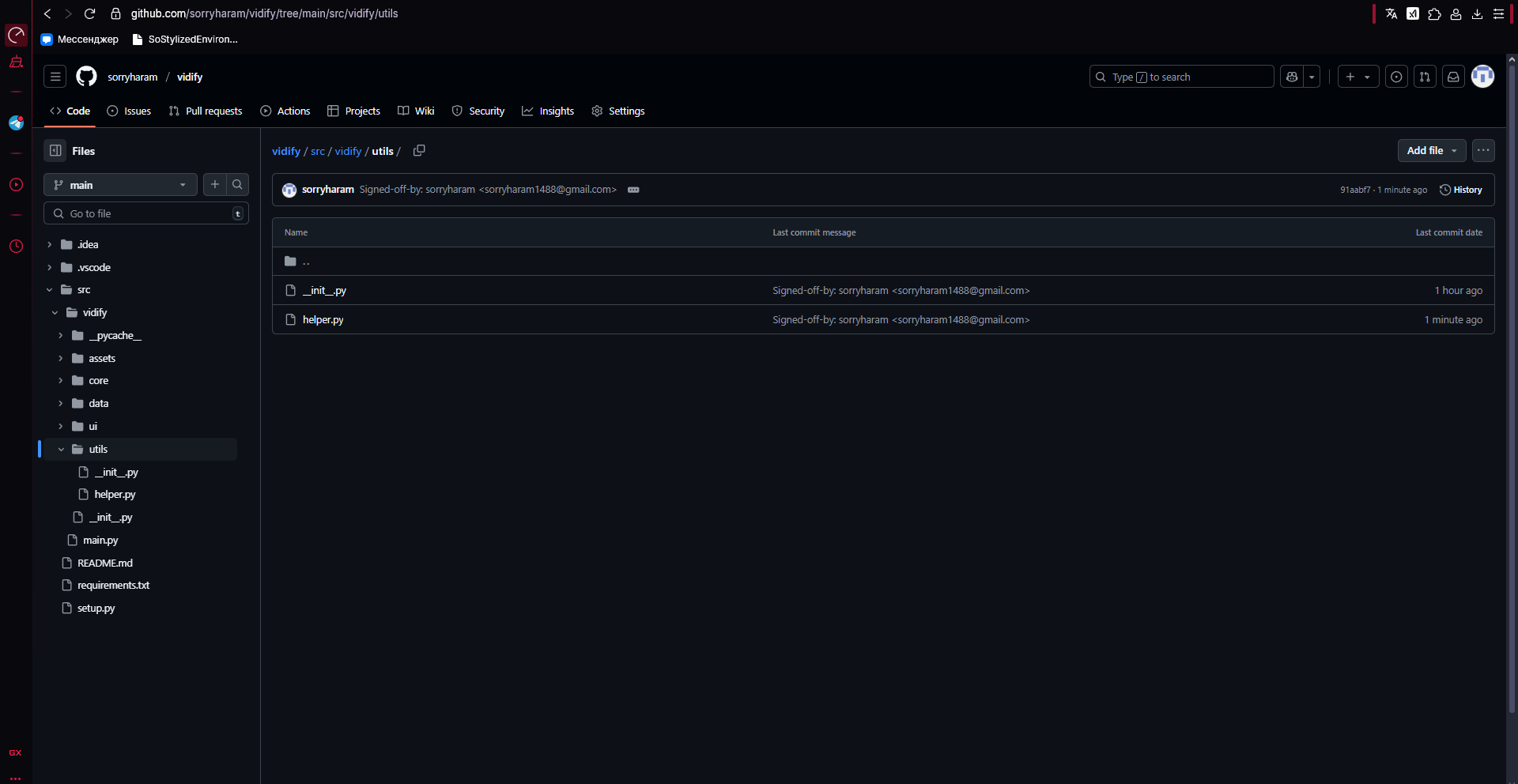


Рисунок 5.9 – Просмотр класса Helper на GitHub

На изображении показана страница GitHub с открытым файлом src/vidify/utils/helper.py, содержащим класс Helper с методом validate\_filename.

Приступим к внесению изменений в класс. Добавим новый метод Helper.

Изменение файла:

* 1. Открыт файл src/vidify/utils/helper.py в IDE Cursor.
  2. Добавлен метод: @staticmethod def get\_file\_extension(filename: str) -> str: """Возвращает расширение файла.""" return filename.split('.')[-1] if '.' in filename else ''
  3. Файл сохранен.

Рассмотрим рисунок 5.10, где продемонстрированы изменения в коде.

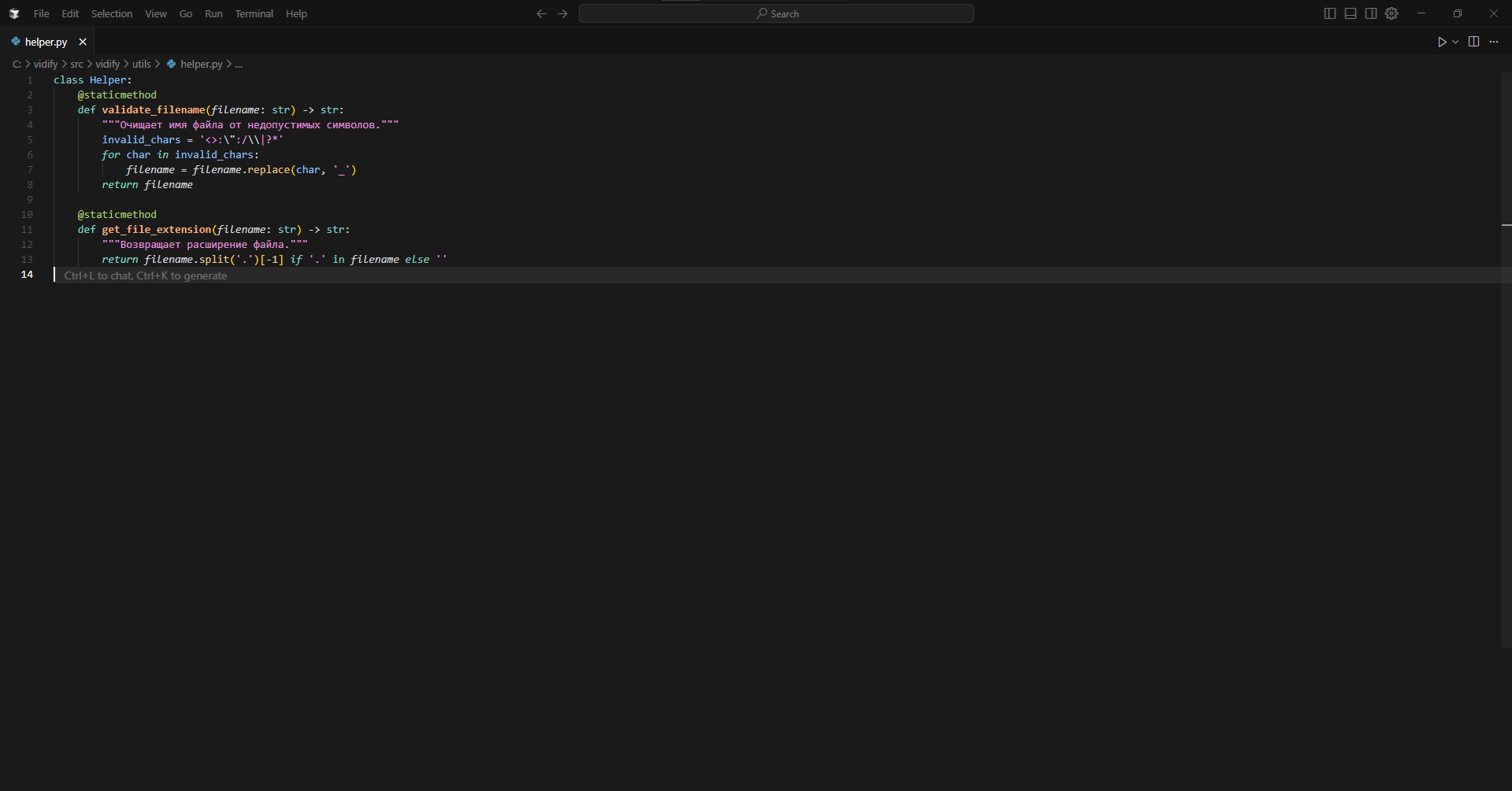


Рисунок 5.10 – Изменения в классе Helper в IDE Cursor

На изображении показано окно IDE Cursor с открытым файлом helper.py, содержащим новый метод get\_file\_extension.

Коммит изменений:

* 1. В папке vidify/ выполнен щелчок правой кнопкой, выбран «TortoiseGit/Commit/master».
  2. Введено сообщение: «Added get\_file\_extension method to Helper class».
  3. Подтвержден коммит.
  4. Изменения отправлены через «TortoiseGit/Push» в ветку main.

Обновим класс в репозитории. Изменения отправлены в репозиторий на шаге 5. Для проверки:

1. Открыт репозиторий на GitHub: https://github.com/sorryharam/vidify.
2. Подтверждено, что файл src/vidify/utils/helper.py содержит методы validate\_filename и get\_file\_extension.

Рассмотрим рисунок 5.11 на котором изображен обновленный класс на GitHub.

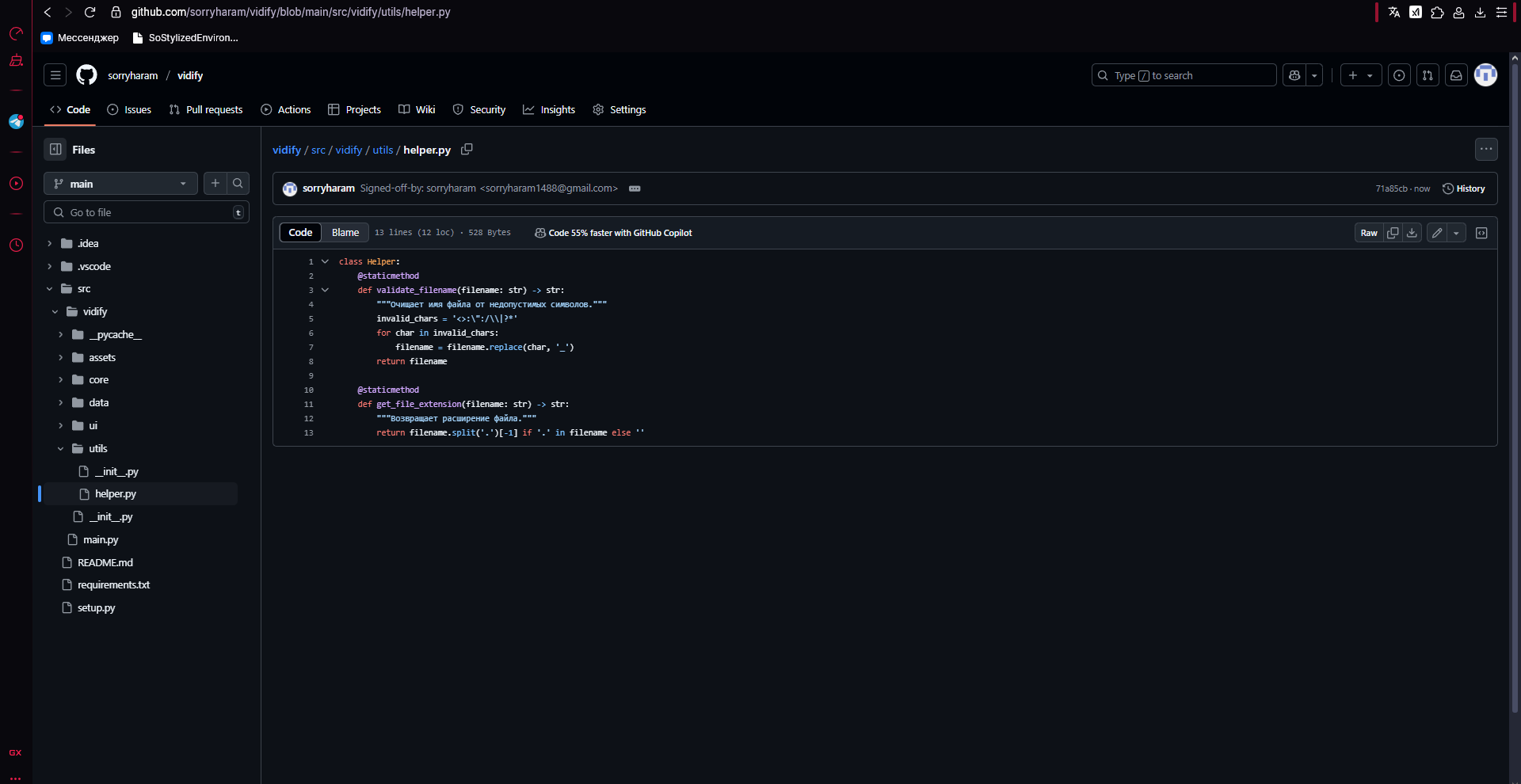


Рисунок 5.11 – Подтверждение изменений в классе Helper на GitHub

На изображении показана страница GitHub с открытым файлом src/vidify/utils/helper.py, содержащим методы validate\_filename и get\_file\_extension.

Рассмотрим удаление локальных файлов и скачивание проекта из репозитория

Для демонстрации восстановления проекта:

1. Удаление локальных файлов:
   * закрыта ide cursor;
   * удалена папка vidify/ через проводник.
2. Клонирование репозитория:
   * создана новая папка (например, c:\projects\vidify-reclone);
   * выполнен щелчок правой кнопкой, выбран «git clone»;
   * введен url: https://github.com/sorryharam/vidify;
   * подтверждено клонирование, структура проекта восстановлена, включая helper.py.

Добавление и удаление «лишнего» файла в репозитории. Добавление лишнего файла:

* + В папке src/vidify/ создан файл temp.txt с текстом: «Временный файл».

Рассмотрим рисунок 5.12, где показано создание файла temp.txt.

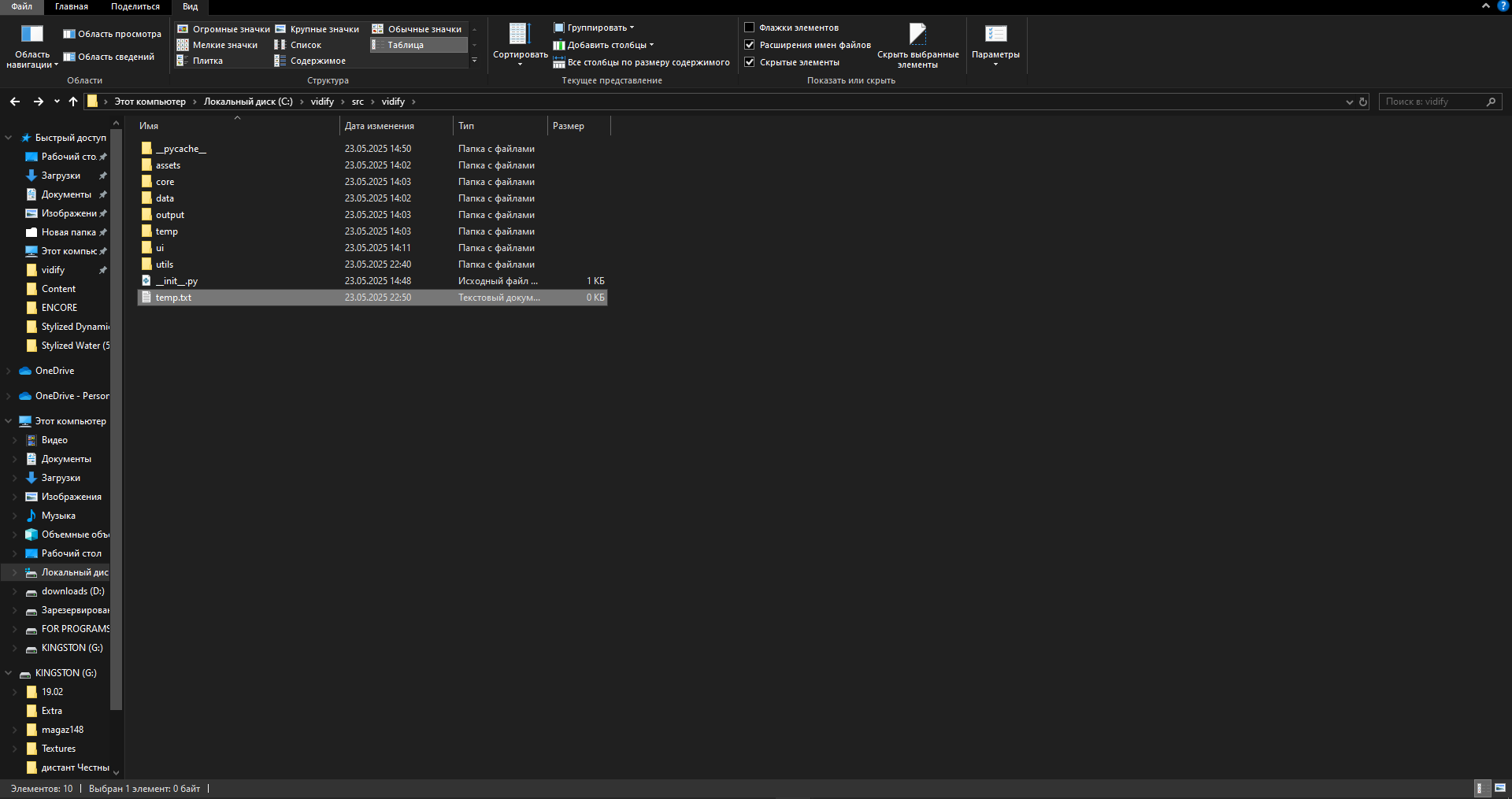


Рисунок 5.12 – Создание файла temp.txt

На изображении показан файл temp.txt, созданный в директории src/vidify/ с содержимым «Временный файл».

* + файл добавлен через «tortoisegit/add»;
  + выполнен коммит с сообщением: «added temporary file».

Рассмотрим рисунок 5.13, на котором показано добавление файла temp.txt в репозиторий.

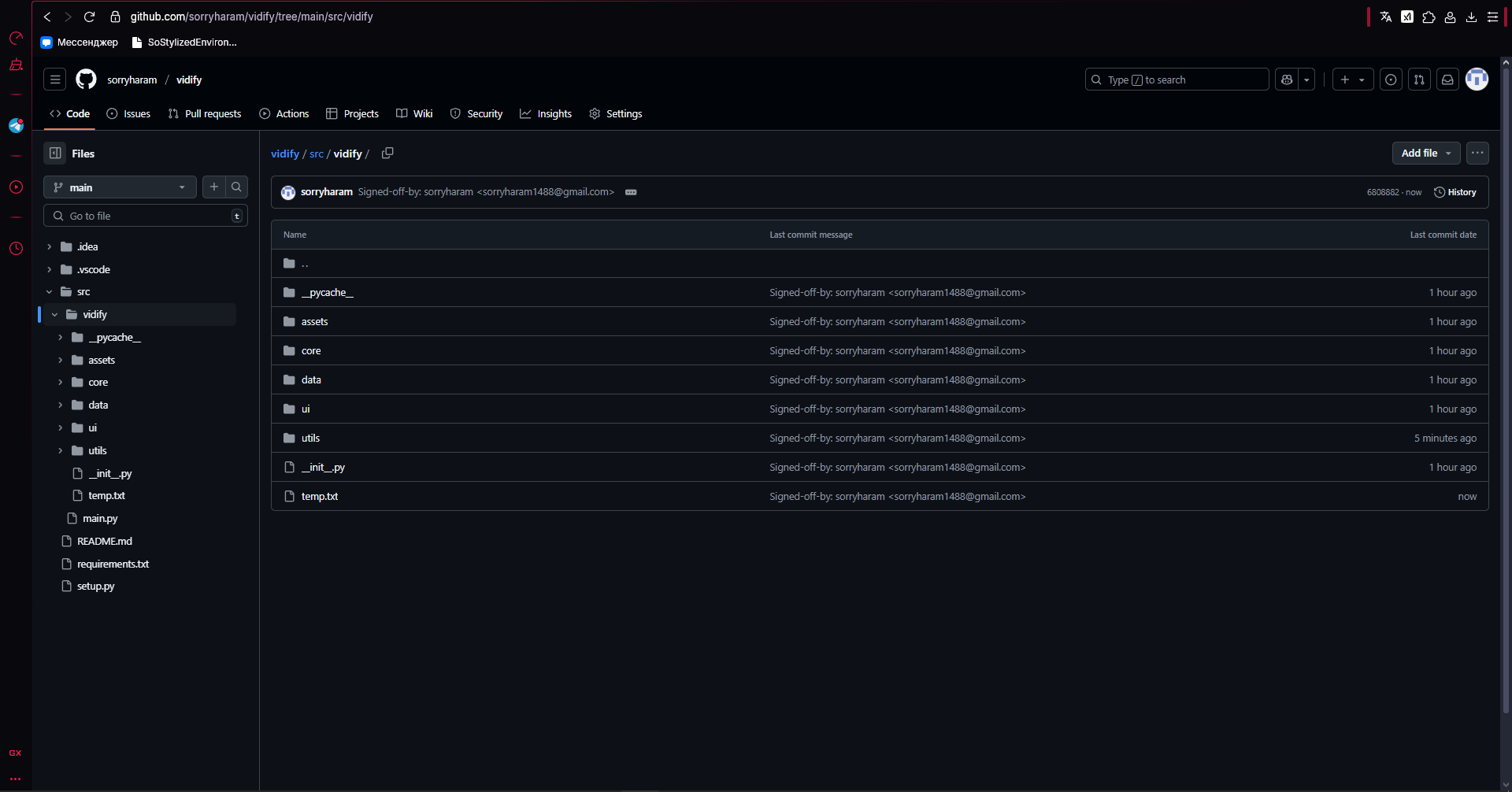


Рисунок 5.13 – Коммит файла temp.txt

На изображении показано окно TortoiseGit с коммитом файла temp.txt в ветку master и сообщением «Added temporary file». Изменения отправлены через «TortoiseGit/Push».

Удаление файла:

* + щелчок правой кнопкой на temp.txt, выбран «tortoisegit/delete and add to ignore list»;
  + выполнен коммит с сообщением: «removed temporary file»;
  + изменения отправлены через «tortoisegit/ push».

Рассмотрим рисунок 5.14, где продемонстрированно удаление файла temp.txt.

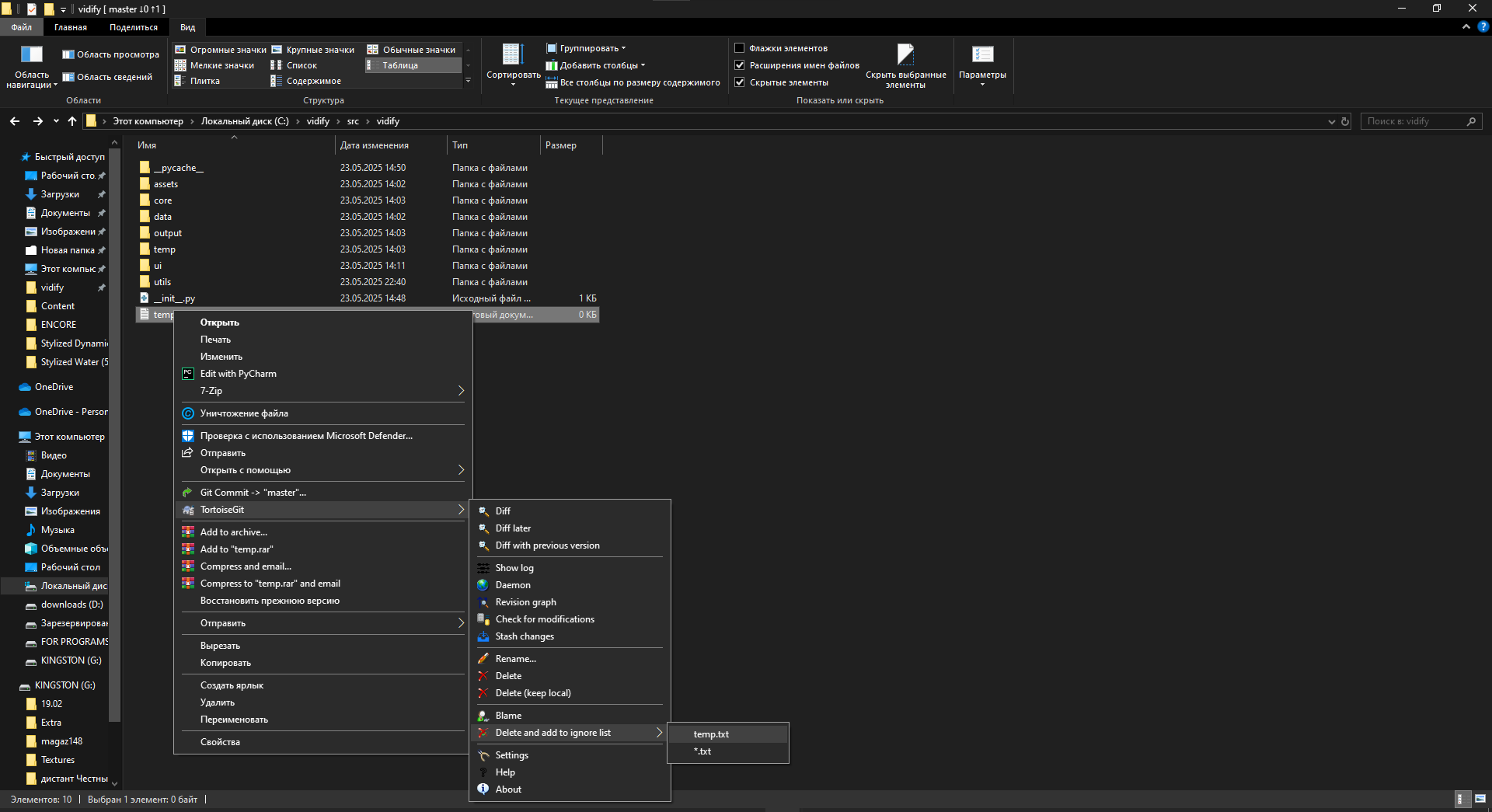


Рисунок 5.14 – Удаление файла temp.txt и коммит

На изображении показано окно TortoiseGit с удалением файла temp.txt, добавлением его в .gitignore и коммитом с сообщением «Removed temporary file».

Рассмотрим рисунок 5.15, где показан репозиторий после удаления файла.

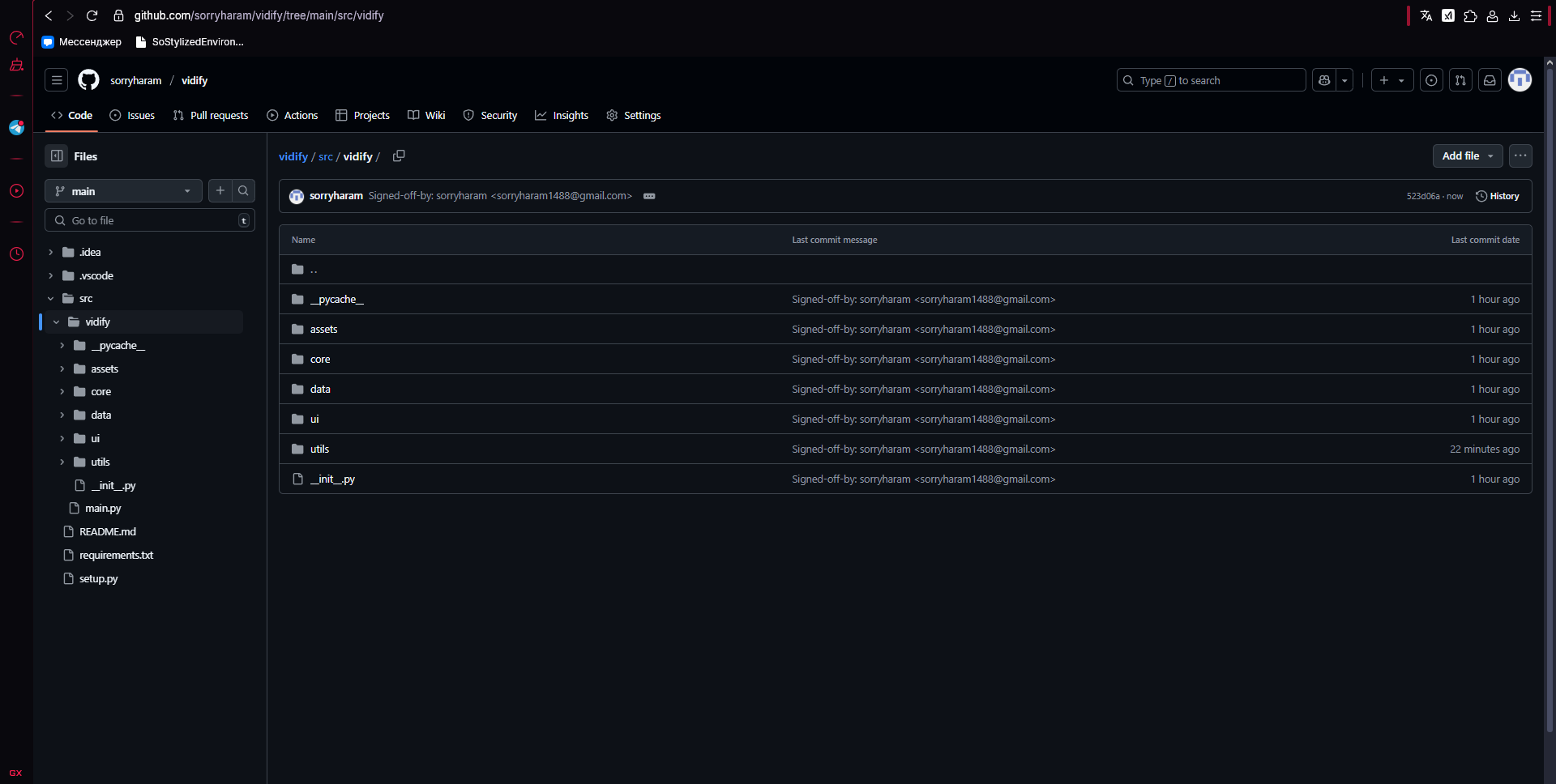


Рисунок 5.15 – Подтверждение удаления файла temp.txt на GitHub

На изображении показана страница GitHub с веткой main, где файл temp.txt отсутствует в директории src/vidify/.

Приступим к изучению журнала изменений файлов. Просмотр логов локально:

* 1. В папке vidify/ выполнен щелчок правой кнопкой, выбран «TortoiseGit/Show log».
  2. Просмотрена история коммитов:
     + «initial commit with project files»;
     + «added helper class»;
     + «added get\_file\_extension method to helper class»;
     + «added temporary file»;
     + «removed temporary file».
  3. Анализ на GitHub:
* открыт репозиторий, раздел «commits» показал те же изменения.

Итогом стало:

* настроенный репозиторий vidify на github (https://github.com/sorryharam/vidify);
* локальный репозиторий, связанный с удаленным через tortoisegit;
* отправлены все локальные файлы проекта в репозиторий;
* добавлен класс helper (utils/helper.py) с двумя методами;
* выполнены изменения, добавление/удаление файлов, изучен журнал изменений;
* подготовлены пятнадцать снимков экрана, подтверждающих выполнение операций;
* отчет с описанием шагов и изображениями.

# **6 ВНЕШНЕЕ ПРОЕКТИРОВАНИЕ**

Vidify – это настольное приложение, предназначенное для скачивания, конвертации и уникализации видеоконтента. Программа реализована с использованием фреймворка PyQt5 и включает несколько экранов (вкладок): DownloadScreen, VideoConvertScreen, VideoEditScreen. Основной пользовательский сценарий – ввод URL, выбор параметров и запуск процесса через основную кнопку. Скачивание осуществляется с помощью yt\_dlp, преобразование и уникализация – с помощью FFmpeg. Для визуализации прогресса используются сигналы PyQt5. Все ошибки записываются в лог-файл.

Перейдем к рассмотрению IPO-диаграммы. Рассмотрим рисунок 6.1, на котором представлена IPO-диаграмма, отражающая общий процесс функционирования приложения Vidify.

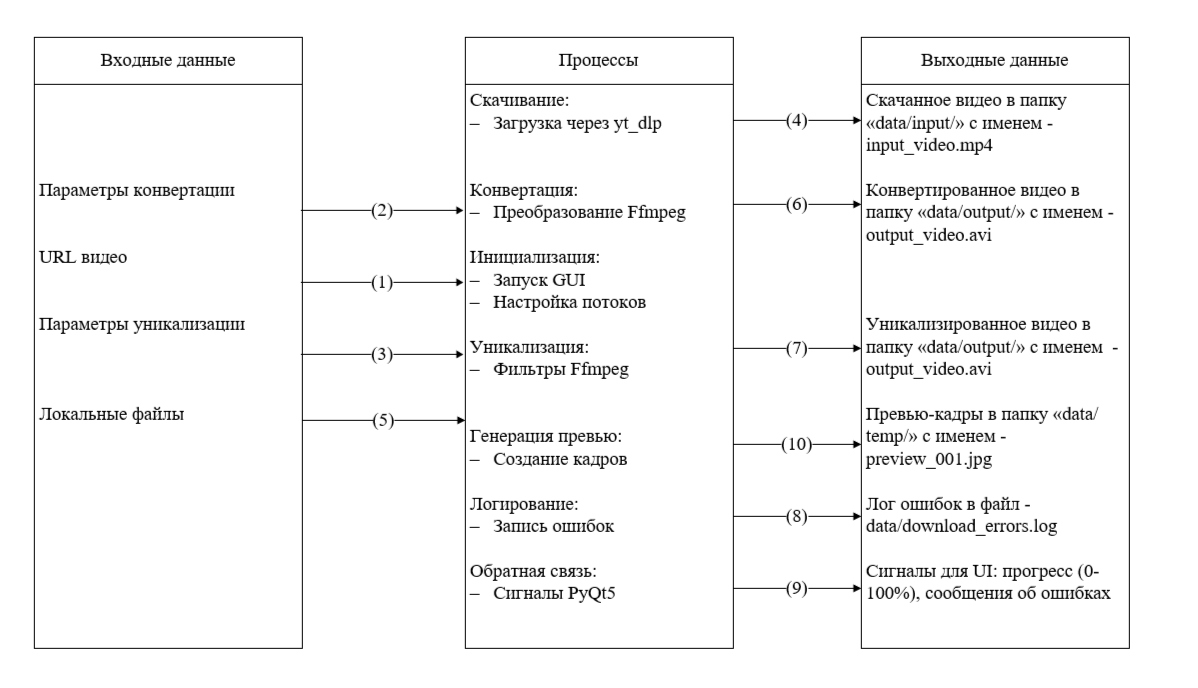


Рисунок 6.1 – IPO-диаграмма программы Vidify

На изображении показано, какие входные данные получает система, какие процессы выполняются внутри, и какие выходные данные формируются в результате. Входными параметрами являются URL, параметры преобразования, параметры уникализации, а также локальные видеофайлы. На выходе – итоговые видеофайлы, лог ошибок и сигналы для обновления пользовательского интерфейса.

В процессе обработки данных в системе происходит несколько логических переходов, каждый из которых приводит к формированию новых объектов или вызывает соответствующие действия.

1) Сначала URL видео передаётся в процесс инициализации.

1. Параметры конвертации поступают в модуль преобразования видео.
2. Параметры уникализации передаются в фильтрующую систему FFmpeg.
3. После скачивания видео сохраняется во входную директорию data/input/.
4. Локальные файлы также используются для генерации предварительных кадров.
5. После завершения процесса конвертации файл сохраняется в папку data/output/.
6. При включении уникализации создаётся дополнительный выходной файл с эффектами.
7. Все ошибки записываются в файл data/download\_errors.log.
8. Сигналы об ошибках и прогрессе передаются в UI.
9. Кадры для предварительного просмотра сохраняются в папке data/temp/.

Во время работы приложения обрабатываются несколько типов входных данных. Каждый из них имеет определённые требования к формату и источнику.

Во-первых, пользователь вводит URL видеофайла через графический интерфейс на вкладке загрузки. Этот URL должен начинаться с http:// или https:// и проходить проверку с использованием регулярных выражений. Поддерживаются все ресурсы, совместимые с yt\_dlp, включая YouTube.

Во-вторых, задаются параметры конвертации. Обычно они представляют собой структуру с указанием формата (например, AVI), битрейта, видеокодека и других технических параметров, совместимых с FFmpeg.

Также пользователь может ввести параметры уникализации. Это могут быть фильтры, такие как crop, scale, eq, применяемые в строковом формате, например: crop=1280:720, scale=1920:1080.

Наконец, пользователь может выбрать локальные файлы для обработки. Такие файлы должны располагаться в директории data/input/, иметь совместимый формат (например, .mp4 или .webm) и размер не превышать 2 ГБ.

Результатом работы программы являются несколько видов выходных данных. Они генерируются в зависимости от выбранных пользователем параметров.

Скачанный видеоролик сохраняется в директории data/input/ с расширением .mp4 или .webm. После этого производится его преобразование: создаётся файл в выбранном формате (например, .avi), который сохраняется в директории data/output/.

Если активирована функция уникализации, программа накладывает на видео выбранные фильтры FFmpeg. В результате получается дополнительный уникализированный файл, также размещённый в data/output/.

Для отображения видео на экране редактирования из видео извлекаются кадры-превью, которые сохраняются в формате .jpg в папке data/temp/.

Все возникающие в процессе ошибки (например, отсутствие соединения, сбой FFmpeg) записываются в файл data/download\_errors.log. Также система генерирует сигналы прогресса и ошибок, которые передаются в интерфейс пользователя и отображаются в соответствующих виджетах.

Программа Vidify выполняет последовательность операций, в каждой из которых происходит преобразование исходных данных в выходной результат.

* + 1. На первом этапе происходит инициализация пользовательского интерфейса и запуск необходимых потоков обработки.
    2. Затем осуществляется скачивание видео с помощью yt\_dlp. Результат сохраняется в папке data/input/.

1. После этого начинается конвертация. Исходя из заданных параметров, вызывается FFmpeg, и создаётся файл с другим расширением, помещаемый в data/output/.
2. При необходимости запускается уникализация. К видеопотоку применяются фильтры, и создаётся дополнительный файл с модификациями.
3. На основе локальных видео или скачанного файла извлекаются отдельные кадры, которые отображаются в графическом интерфейсе.
4. Все действия сопровождаются логированием. При возникновении ошибок информация об этом записывается, а пользователь получает уведомление.
5. Также во время выполнения операций система отправляет сигналы интерфейсу, отображая прогресс на экране.

Программа Vidify учитывает потенциальные сбои и реализует несколько механизмов обеспечения надёжности.

Во-первых, реализована обработка ошибок при скачивании. Если сервер возвращает ошибку (например, 403 или 404), yt\_dlp сообщает об этом, и информация заносится в лог. Пользователь при этом видит сообщение об ошибке в всплывающем окне.

Во-вторых, FFmpeg также фиксирует ошибки кодирования. При сбое видеофайл не создаётся, и программа уведомляет пользователя. Потоки, в которых выполняются операции, изолированы от основного интерфейса, что предотвращает зависание интерфейса при больших нагрузках.

В-третьих, предусмотрено восстановление после сбоев. Частично загруженные файлы сохраняются с префиксом partial\_ и могут быть повторно обработаны после перезапуска. Все временные данные очищаются при завершении работы программы.

Программа демонстрирует достаточную производительность и разумное потребление ресурсов.

При использовании yt\_dlp скорость скачивания достигает 10 МБ/с. Конвертация осуществляется с использованием FFmpeg с параметром -preset fast, при наличии видеокарты возможен переход на NVENC для дополнительного ускорения.

Работа программы организована с помощью потоков QThread, что позволяет выполнять операции параллельно, не блокируя интерфейс.

По ресурсам: для корректной работы требуется минимум 2 ядра процессора и 4 ГБ оперативной памяти. При обработке большого количества видео желательно иметь 8 ГБ ОЗУ и 50 ГБ свободного места на диске.

Для оптимизации предусмотрено использование низкого качества видео при загрузке по умолчанию и копирование потока (-c:v copy) при отсутствии необходимости перекодирования.

В процессе анализа были выявлены области, которые могут быть улучшены в будущих версиях программы.

* + 1. Необходимо реализовать проверку корректности фильтров FFmpeg с использованием регулярных выражений.
    2. Можно перейти на использование asyncio для асинхронной загрузки нескольких видеофайлов.
    3. Имеет смысл добавить кэширование метаданных, создавая отдельную директорию data/cache/.
  1. Рекомендуется расширить логирование, добавив отдельный лог-файл для FFmpeg и разделив уровни логов (INFO, ERROR).
  2. Следует реализовать функциональность кнопки "Отмена", которая будет прерывать выполнение текущего потока.
  3. Для большей кроссплатформенности стоит использовать pathlib вместо os.path.
  4. Также необходимо реализовать проверку наличия FFmpeg в системе с помощью shutil.which("ffmpeg").

# **7 ВНУТРЕННЕЕ ПРОЕКТИРОВАНИЕ (РАЗРАБОТКА СХЕМ И ДИАГРАММ ПРОЕКТА)**

В рамках выполнения задания 7 "Внутреннее проектирование (разработка схем и диаграмм проекта)" были разработаны модели, описывающие структуру и процессы программы Vidify. Моделирование выполнено в соответствии с методологиями IDEF0, IDEF3 и DFD (диаграммы потоков данных) для обеспечения полноты представления системы. Данные модели отражают функциональные аспекты, последовательность процессов и потоки данных в рамках проектирования программного обеспечения.

Модель IDEF0 (Функциональное моделирование) представлена на уровне A-0 и предназначена для описания основной функции системы Vidify – обработки видеоконтента.

Рассмотрим рисунок 7.1, на котором изображена модель IDEF0 уровня A-0, демонстрирующая общий контекст работы системы.

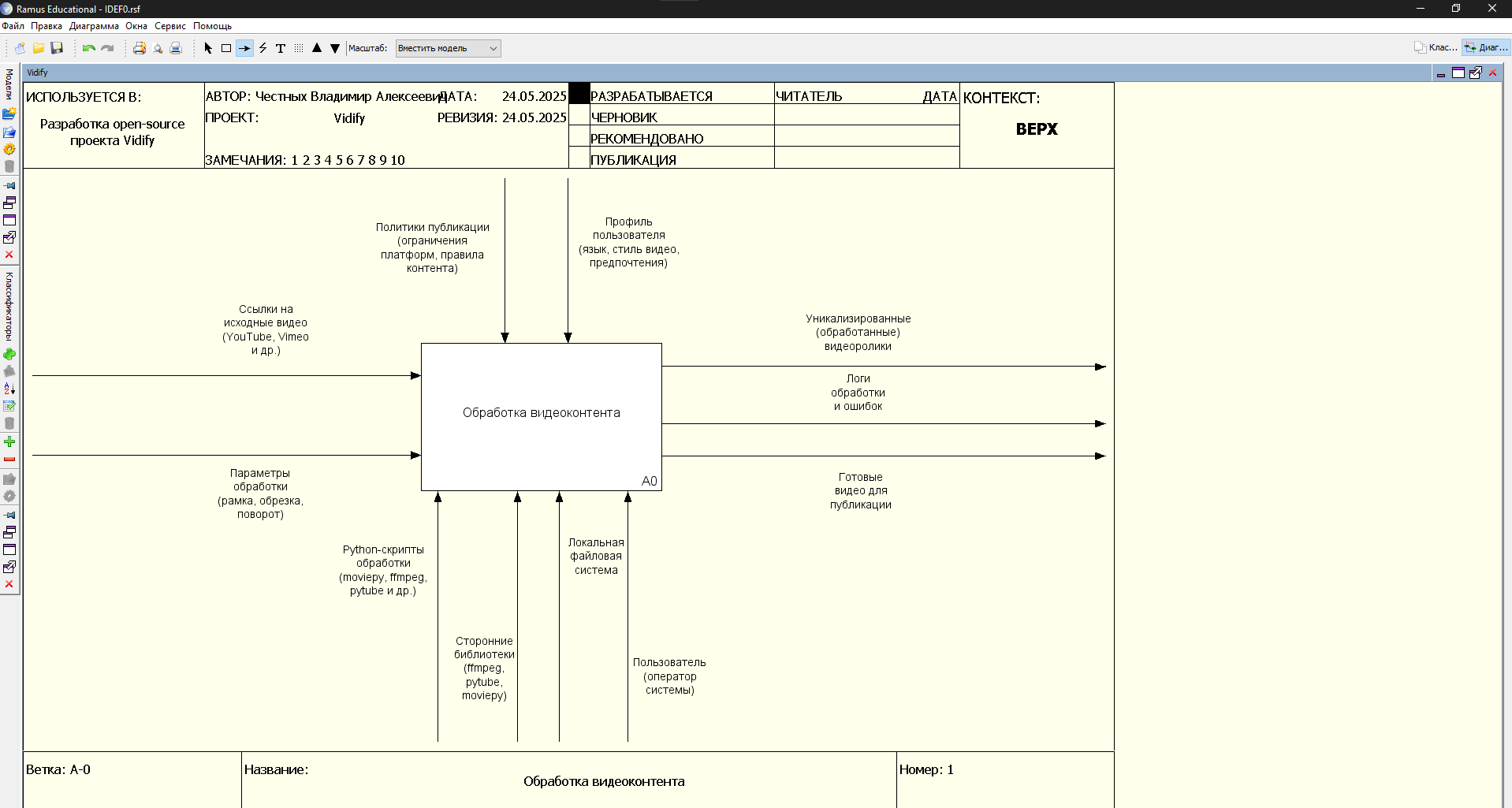


Рисунок 7.1 – Модель IDEF0

Мы рассмотрели Рисунок 7.1 и увидели на нём функциональную схему обработки видеоконтента.

Описание модели, рассмотренной на рисунке 7.1:

1. Главная функция: "Обработка видеоконтента".
2. Входы: URL видео, параметры конвертации, параметры уникализации.
3. Управление: Правила FFmpeg, настройки yt\_dlp.
4. Механизмы: Пользователь, FFmpeg, yt\_dlp, PyQt5.
5. Выходы: Скачанное видео, конвертированное видео, уникализированное видео, превью, лог ошибок.

Модель демонстрирует взаимодействие системы с внешними сущностями и внутренними механизмами, обеспечивая общее представление о процессе.

Модель IDEF3 (Моделирование процессов) описывает последовательность этапов выполнения задач в системе Vidify.

Рассмотрим рисунок 7.2, на котором представлена модель IDEF3, отражающая последовательность процессов обработки видео.

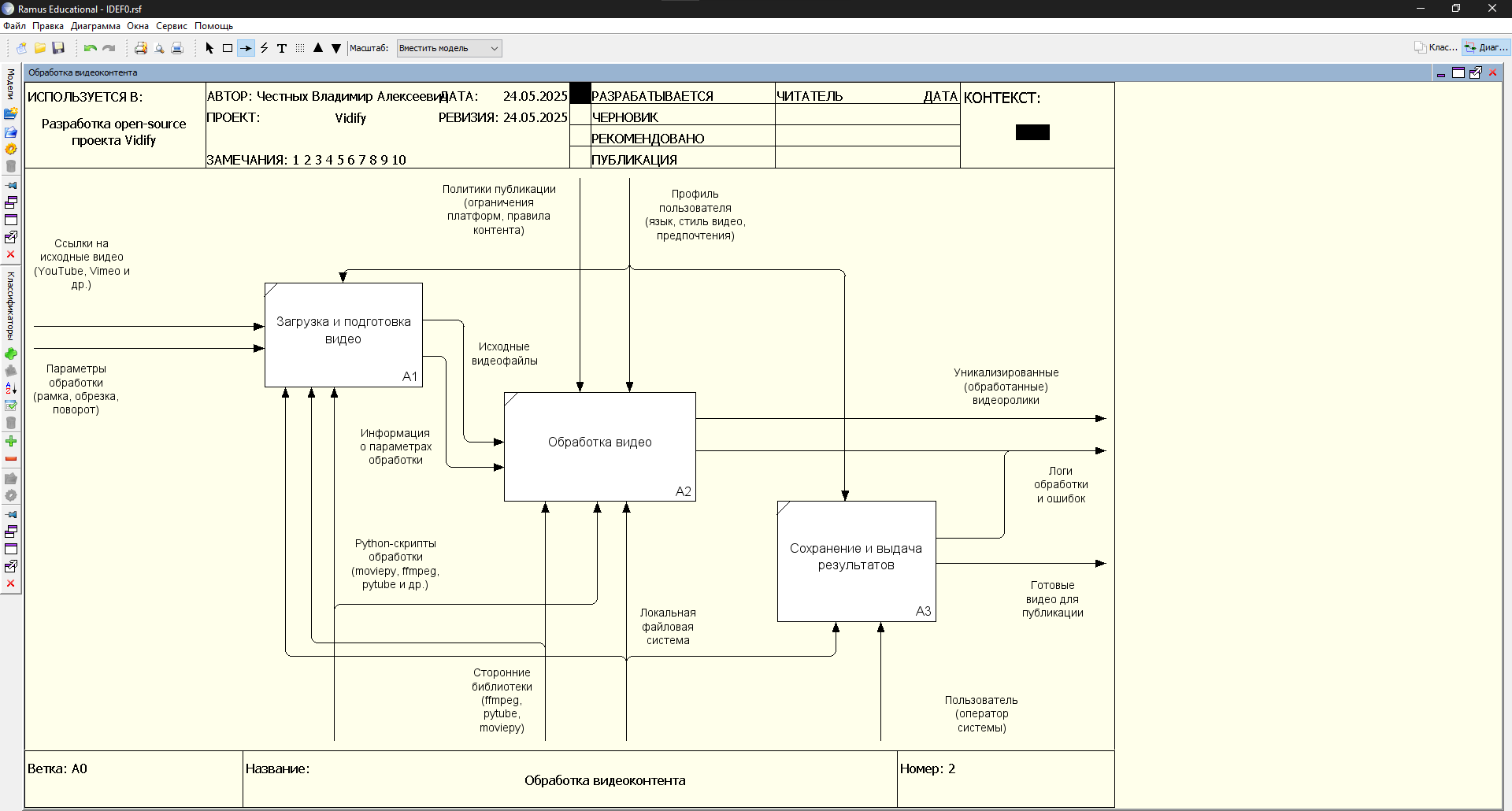


Рисунок 7.2 – Модель IDEF3

Мы рассмотрели Рисунок 7.2 и увидели на нём этапы работы системы Vidify.

Описание этапов, рассмотренных на рисунке 7.2:

1. Инициализация (запуск GUI и настройка).
2. Скачивание видео (по URL через yt\_dlp).
3. Конвертация (применение параметров для изменения формата).
4. Уникализация (применение фильтров FFmpeg).
5. Генерация превью (создание кадров).
6. Логирование и обратная связь (сохранение логов и обновление UI).

Модель отражает временную последовательность операций, обеспечивая основу для детализации процессов в DFD.

Модель DFD (Диаграмма потоков данных) уровня 1 иллюстрирует движение данных между процессами, хранилищами и внешними сущностями в системе Vidify.

Рассмотрим рисунок 7.3, на котором изображена модель DFD уровня 1, показывающая потоки данных в системе.

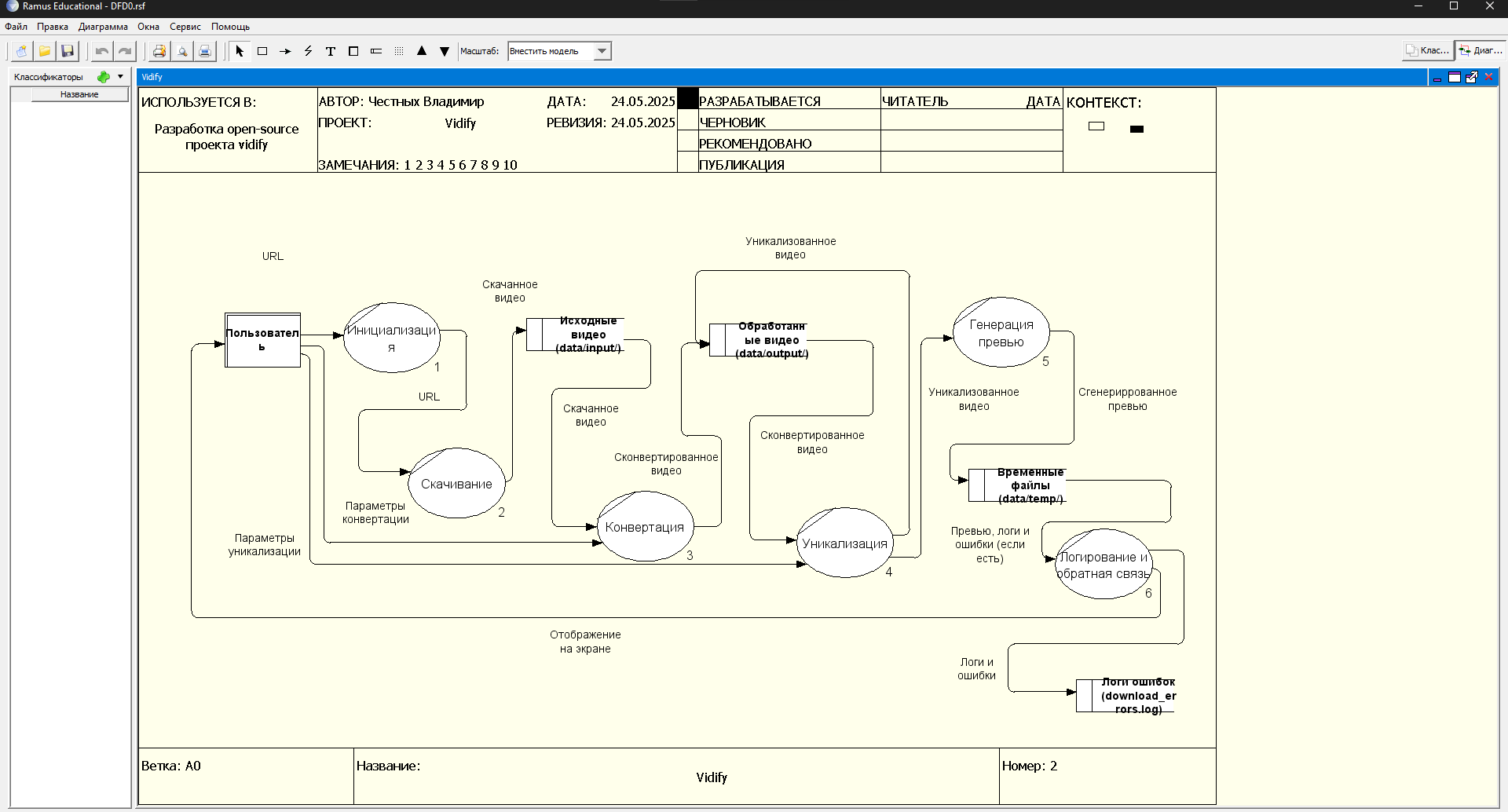


Рисунок 7.3 – Модель DFD

Мы рассмотрели Рисунок 7.3 и увидели на нём, как данные перемещаются между процессами и хранилищами.

Подробное описание модели DFD рассмотренной на рисунке 7.3:

* + 1. Внешняя сущность:
* пользователь.
  + 1. Процессы:
* инициализация;
* скачивание;
* конвертация;
* уникализация;
* генерация превью;
* логирование и обратная связь.
  + 1. Хранилища данных:
* d1: data/input/ (скачанные видео);
* d2: data/output/ (обработанные видео);
* d3: data/temp/ (превью);
* d4: download\_errors.log (логи).
  + 1. Потоки данных:
* пользователь передаёт url и параметры в процесс "инициализация";
* процесс "инициализация" передаёт url и подтверждение готовности в процесс "скачивание";
* процесс "скачивание" передаёт скачанное видео в хранилище d1 (data/input/);
* хранилище d1 передаёт скачанное видео в процесс "конвертация";
* процесс "конвертация" передаёт видео в хранилище d2 (data/output/);
* хранилище d2 передаёт видео в процесс "уникализация";
* процесс "уникализация" передаёт уникализированное видео в хранилище d2;
* хранилище d1 передаёт скачанное видео в процесс "генерация превью";
* хранилище d2 передаёт обработанное видео в процесс "генерация превью";
* процесс "генерация превью" передаёт превью-кадры в хранилище d3 (data/temp/);
* хранилище d3 передаёт превью-кадры в процесс "логирование";
* процесс "скачивание" передаёт ошибки в процесс "логирование";
* процесс "конвертация" передаёт ошибки в процесс "логирование";
* процесс "уникализация" передаёт ошибки в процесс "логирование";
* процесс "логирование" передаёт лог ошибок в хранилище d4 (download\_errors.log);
* процесс "логирование" передаёт прогресс, сообщения об ошибках и превью пользователю;
* хранилище d2 передаёт уникализированное видео пользователю.

Модель обеспечивает визуализацию данных, циркулирующих в системе, и их обработки.

Разработанные модели (IDEF0, IDEF3, DFD) в совокупности обеспечивают полное представление внутренней структуры и функционирования программы Vidify.

## **8 РАЗРАБОТКА МОДУЛЕЙ ПРОЕКТА И ИХ ЭЛЕМЕНТОВ**

Программа Vidify предназначена для скачивания видеоконтента с интернет-платформ, конвертации форматов и уникализации видео путем редактирования, обрезки и наложения эффектов. Программа реализована на языке Python с использованием библиотек yt\_dlp (скачивание), FFmpeg (обработка и конвертация видео) и PyQt5 (графический интерфейс). Архитектура модульная, включает подсистемы Downloader, VideoProcessor, UI и Data.

Сейчас мы рассмотрим внешнюю спецификацию модулей, описывающих функциональность программы и её компонентов.

Модуль main.py является точкой входа программы, отвечает за запуск приложения и инициализацию графического интерфейса.

Рассмотрим рисунок 8.1, на котором представлен блок импортов модуля main.py.

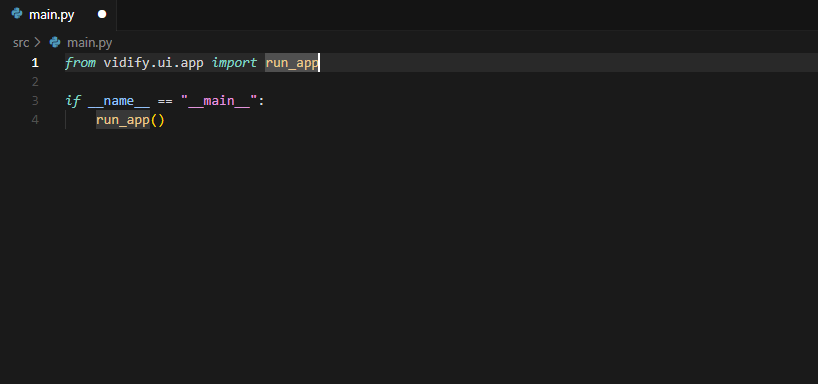


Рисунок 8.1 – Импорты модуля main.py

Мы рассмотрели рисунок 8.1, на нём изображено подключение модуля. Следующие элементы используются для подключения:

* импорты: vidify.ui.app.run\_app;
* зависимости: pyqt5;
* вызывает функцию run\_app из ui/app.py.

Входные данные: Аргументы командной строки (не используются).

Выходные данные: Запущенное приложение с графическим интерфейсом.

Модуль содержит единственную функцию run\_app(), вызываемую при запуске.

Рассмотрим рисунок 8.2, на котором представлен исходный код модуля main.py.

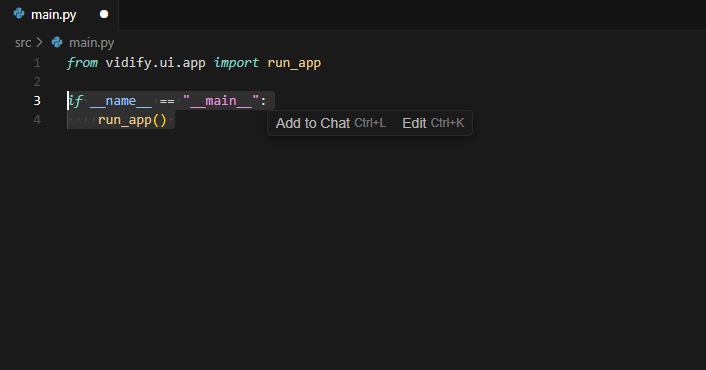


Рисунок 8.2 – Исходный код модуля main.py

Мы рассмотрели рисунок 8.2, на нём изображено определение функции run\_app() и её вызов через условие if \_\_name\_\_ == "\_\_main\_\_".

Средства защиты информации:

* проверка условия if \_\_name\_\_ == "\_\_main\_\_", предотвращающая выполнение при импорте;
* делегирование обработки ошибок модулю ui/app.py.

Модуль core/downloader.py реализует подсистему скачивания видеоконтента по URL с различных платформ (YouTube и другие), включая получение информации о видео.

Рассмотрим рисунок 8.3, на котором представлен блок импортов модуля core/downloader.py.

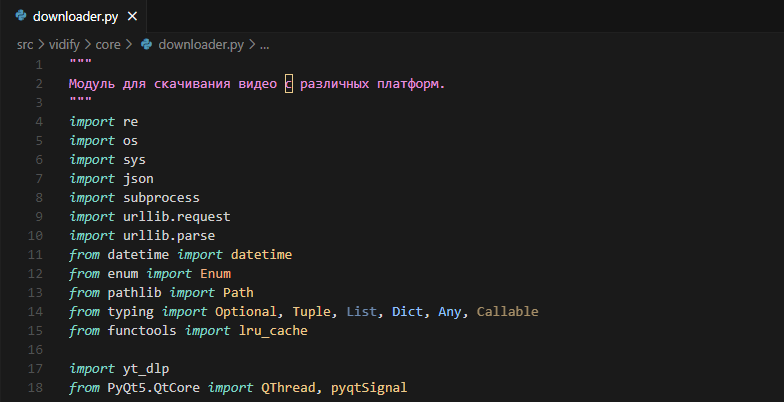


Рисунок 8.3 – Импорты модуля core/downloader.py

Мы рассмотрели рисунок 8.3, на нём изображено подключение модулей. Следующие элементы используются для подключения:

* импорты: re, os, sys, json, subprocess, urllib.request, urllib.parse, datetime, enum, pathlib, typing, functools, yt\_dlp, pyqt5.qtcore;
* зависимости: yt\_dlp, pyqt5. используется модулем ui/app.py через экран downloadscreen.

Входные данные:

* url видео (строка, формат: http:// или https://);
* путь для сохранения (строка или path);
* опционально: format\_id (строка, формат видео), log\_file (путь к логу).

Выходные данные:

* путь к загруженному файлу (строка);
* лог ошибок (data/download\_errors.log);
* информация о видео (словарь с ключами title, uploader, thumbnail, id);
* сигналы pyqt5: прогресс (int), статус (str), ошибка (str), завершение.

Модуль включает перечисление DownloadStatus для статусов скачивания и константу VIDEO\_EXTS с поддерживаемыми расширениями видео. Также реализованы следующие функции и классы:

* + 1. Функция is\_valid\_url(url: str) -> bool проверяет валидность URL.

Рассмотрим рисунок 8.4, на котором представлен исходный код функции is\_valid\_url из модуля core/downloader.py.

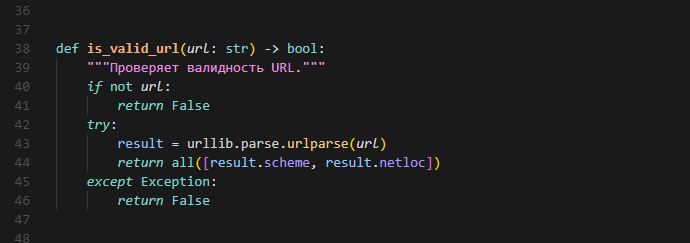


Рисунок 8.4 – Исходный код функции is\_valid\_url

Мы рассмотрели рисунок 8.4, на нём изображено использование urllib.parse для проверки схемы URL и валидации строки.

* + 1. Функция setup\_paths(base\_dir\_name: str) -> Tuple[Path, Path, Path] создаёт директории input, output, temp.

Рассмотрим рисунок 8.5, на котором представлен исходный код функции setup\_paths.

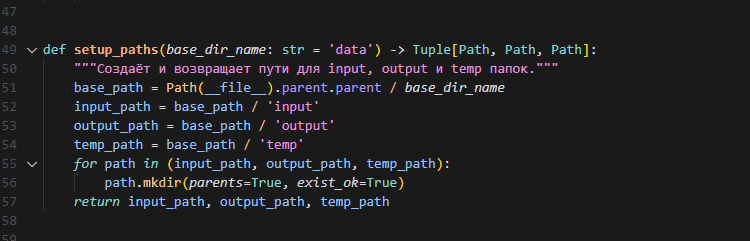


Рисунок 8.5 – Исходный код функции setup\_paths

Мы рассмотрели рисунок 8.5, на нём изображено создание трёх директорий с использованием pathlib.Path и метода mkdir.

* + 1. Функция open\_folder(folder\_path: str) -> None открывает папку в проводнике.

Рассмотрим рисунок 8.6, на котором представлен исходный код функции open\_folder.

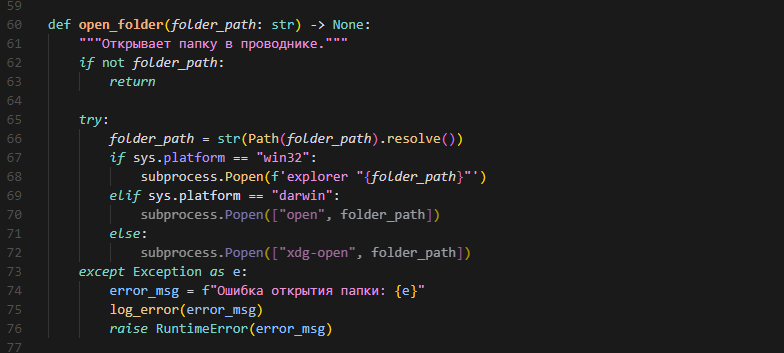


Рисунок 8.6 – Исходный код функции open\_folder

Мы рассмотрели рисунок 8.6, на нём изображено использование os.startfile или subprocess.run для открытия папки в зависимости от ОС.

* + 1. Функция log\_error(error\_msg: str, url: str = None) -> None логирует ошибки.

Рассмотрим рисунок 8.7, на котором представлен исходный код функции log\_error.

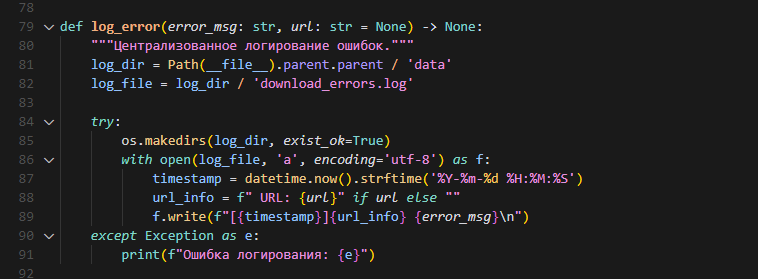


Рисунок 8.7 – Исходный код функции log\_error

Мы рассмотрели рисунок 8.7, на нём изображено добавление сообщения об ошибке с временной меткой в файл data/download\_errors.log.

* + 1. Функция extract\_video\_id(url: str) -> Optional[str] извлекает ID видео YouTube (кэшируется).

Рассмотрим рисунок 8.8, на котором представлен исходный код функции extract\_video\_id.

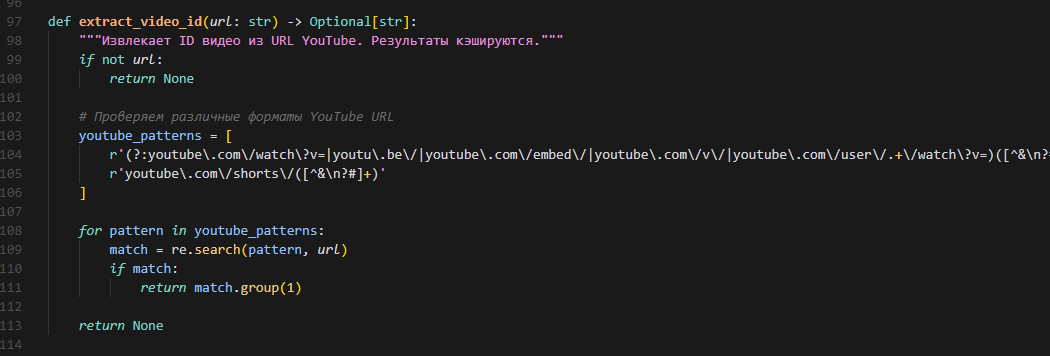


Рисунок 8.8 – Исходный код функции extract\_video\_id

Мы рассмотрели рисунок 8.8, на нём изображено использование регулярного выражения и декоратора @lru\_cache для кэширования ID видео.

* + 1. Класс VideoInfoFetcher(QThread) содержит методы \_\_init\_\_, abort, run, \_get\_youtube\_info, \_get\_info\_via\_ytdlp и сигналы info\_ready(dict), error(str).

Рассмотрим рисунок 8.9, на котором представлен исходный код класса VideoInfoFetcher.

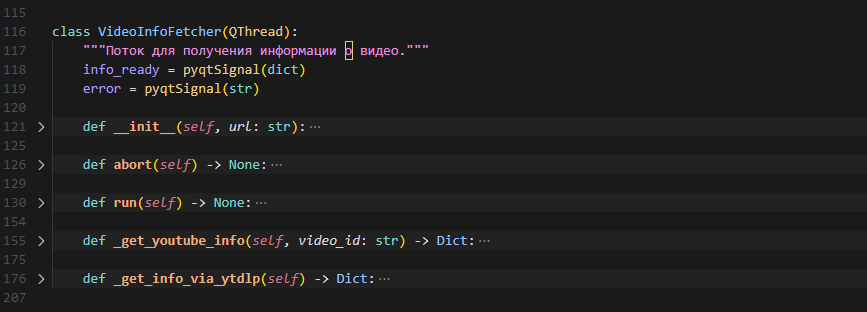


Рисунок 8.9 – Исходный код класса VideoInfoFetcher

Мы рассмотрели рисунок 8.9, на нём изображены методы класса, включая запуск потока для получения информации о видео через yt\_dlp и отправку сигналов.

* + 1. Класс VideoDownloader(QThread) содержит методы \_\_init\_\_, abort, ydl\_hook, run и сигналы update\_progress(int), update\_status(str), finished\_with\_error(str), download\_complete().

Рассмотрим рисунок 8.10, на котором представлен исходный код класса VideoDownloader.



Рисунок 8.10 – Исходный код класса VideoDownloader

Мы рассмотрели рисунок 8.10, на нём изображены методы класса, включая обработку прогресса скачивания через ydl\_hook и отправку сигналов об успехе или ошибке.

Средства защиты информации:

* валидация url через is\_valid\_url с использованием urllib.parse;
* проверка существования директорий перед сохранением (save\_path.exists());
* логирование ошибок в защищённый файл data/download\_errors.log с временными метками;
* обработка исключений в videoinfofetcher и videodownloader с отправкой ошибок через сигналы;
* кэширование id видео через lru\_cache для предотвращения повторных запросов;
* троттлинг обновлений ui в ydl\_hook для оптимизации производительности;
* проверка размера скачанного файла (candidate.stat().st\_size > 0);
* ограничение доступа к директориям data/\* через права ос.

Модуль core/video\_processor.py реализует подсистему обработки видеофайлов с использованием FFmpeg, включая конвертацию форматов (например, MP4 в AVI, WebM, MKV), обрезку, применение фильтров и создание превью.

Рассмотрим рисунок 8.11, на котором представлен блок импортов модуля core/video\_processor.py.

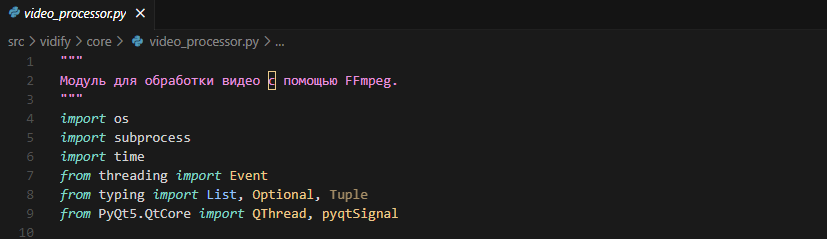


Рисунок 8.11 – Импорты модуля core/video\_processor.py

Мы рассмотрели рисунок 8.11, на нём изображено подключение модулей. Следующие элементы используются для подключения:

* импорты: os, subprocess, time, threading, typing, pyqt5.qtcore;
* зависимости: ffmpeg (через subprocess), pyqt5;
* используется модулем ui/app.py через экраны videoeditscreen и videoconvertscreen.

Входные данные:

* команда ffmpeg (список строк, например, ['ffmpeg', '-i', 'input.mp4', '-c:v', 'libx264', 'output.avi'] для конвертации);
* путь к выходному файлу (строка, например, data/output/video.avi);
* флаг parse\_progress (булево, для отслеживания прогресса конвертации/обработки);
* общее количество кадров (total\_frames, опционально, для расчёта прогресса).

Выходные данные:

* путь к обработанному/сконвертированному файлу (строка, например, data/output/video.avi) или статус "ok";
* сигналы pyqt5: прогресс (int, процент завершения), завершение (str, путь или статус), ошибка (str, текст ошибки).

Модуль включает следующий функционал:

1. Класс FFmpegProcessor(QThread) содержит методы \_\_init\_\_, run, stop и сигналы finished(str), error(str), progress(int).

Рассмотрим рисунок 8.12, на котором представлен исходный код класса FFmpegProcessor.

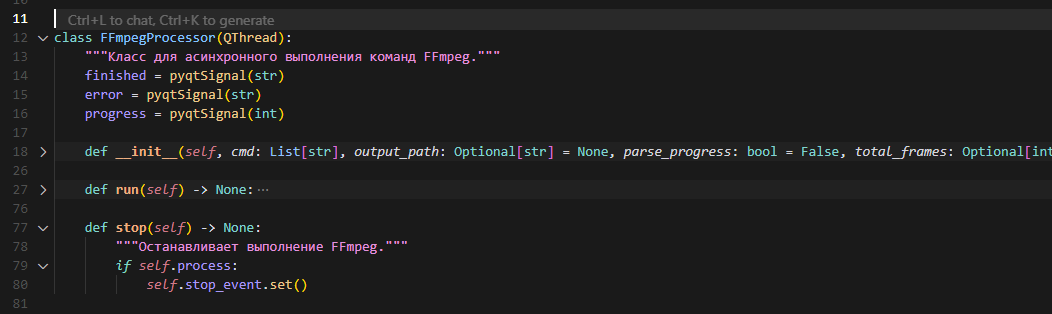


Рисунок 8.12 – Исходный код класса FFmpegProcessor

Мы рассмотрели рисунок 8.12, на нём изображены методы класса, включая запуск команды FFmpeg в потоке и отправку сигналов о прогрессе и ошибках.

1. Функция create\_ffmpeg\_command(input\_path: str, output\_path: str, filter\_str: Optional[str], is\_preview: bool, frame\_time: str) -> List[str] создаёт команду FFmpeg для конвертации, обрезки или фильтров.

Рассмотрим рисунок 8.13, на котором представлен исходный код функции create\_ffmpeg\_command.

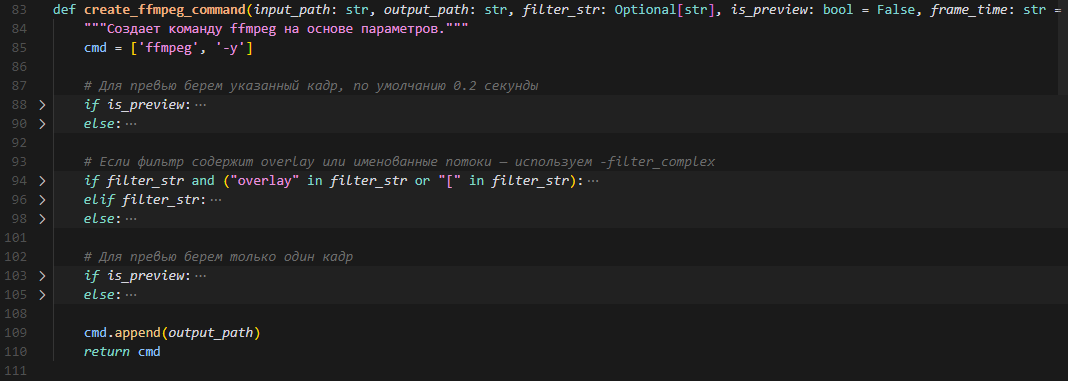


Рисунок 8.13 – Исходный код функции create\_ffmpeg\_command

Мы рассмотрели рисунок 8.13, на нём изображено формирование команды FFmpeg с учётом фильтров, флага превью и времени кадра.

1. Функция check\_ffmpeg\_available() -> bool проверяет наличие FFmpeg.

Рассмотрим рисунок 8.14, на котором представлен исходный код функции check\_ffmpeg\_available.

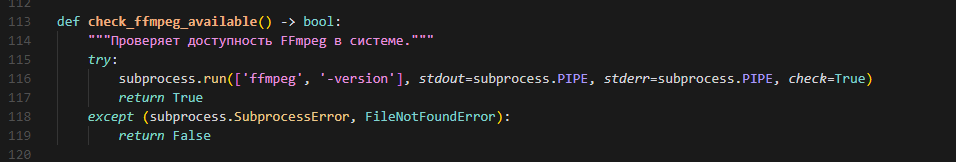


Рисунок 8.14 – Исходный код функции check\_ffmpeg\_available

Мы рассмотрели рисунок 8.14, на нём изображено выполнение команды ffmpeg -version через subprocess для проверки доступности FFmpeg.

1. Функция cleanup\_temp\_files(temp\_dir: str, max\_age\_hours: int) -> None очищает временные файлы.

Рассмотрим рисунок 8.15, на котором представлен исходный код функции cleanup\_temp\_files.

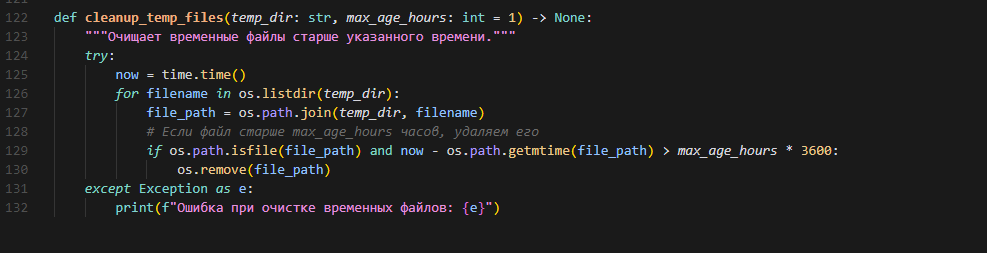


Рисунок 8.15 – Исходный код функции cleanup\_temp\_files

Мы рассмотрели рисунок 8.15, на нём изображено удаление файлов в директории temp\_dir, старше max\_age\_hours, с использованием os и time.

Средства защиты информации:

* проверка наличия ffmpeg перед выполнением команд;
* валидация входных параметров (input\_path, output\_path, filter\_str);
* обработка исключений в ffmpegprocessor с отправкой ошибок через сигналы;
* проверка существования и возраста временных файлов перед удалением;
* использование subprocess с безопасными параметрами (check=true);
* ограничение доступа к директориям data/\* через права ос.

Модуль ui/app.py реализует главное окно приложения с интерфейсом, включающим вкладки для скачивания, уникализации и конвертации видео. Вкладка "Конвертер" доступна через кнопку на вкладке "Скачивание", позволяя пользователю конвертировать видео в различные форматы.

Рассмотрим рисунок 8.16, на котором представлен блок импортов модуля ui/app.py.

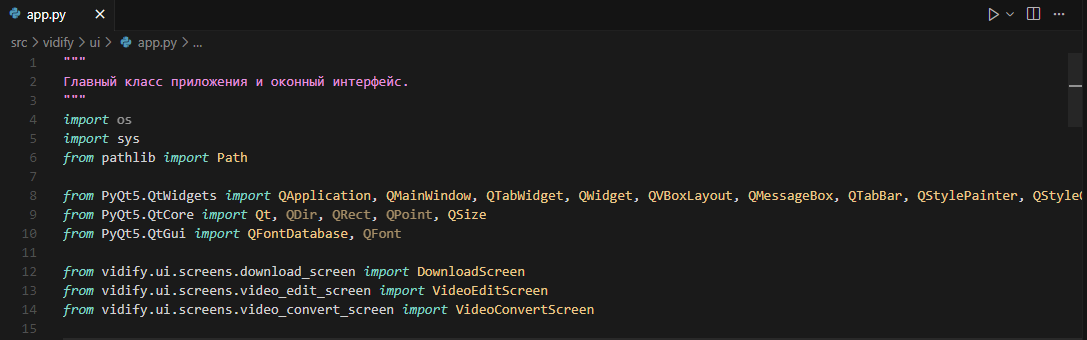


Рисунок 8.16 – Импорты модуля ui/app.py

Мы рассмотрели рисунок 8.16, на нём изображено подключение модулей. Следующие элементы используются для подключения:

* импорты: os, sys, pathlib, pyqt5.qtwidgets, pyqt5.qtcore, pyqt5.qtgui, vidify.ui.screens.download\_screen.downloadscreen, vidify.ui.screens.video\_edit\_screen.videoeditscreen, vidify.ui.screens.video\_convert\_screen.videoconvertscreen;
* зависимости: pyqt5;
* используется модулем main.py.

Входные данные:

* события интерфейса (переключение вкладок, нажатие кнопки "конвертер" на downloadscreen, выбор формата);
* данные от экранов (downloadscreen, videoeditscreen, videoconvertscreen), включая путь к файлу и параметры конвертации (например, целевой формат: avi, webm).

Выходные данные:

* отображение интерфейса с вкладками и переходом к конвертеру через кнопку;
* передача команд на экраны, включая запуск конвертации через videoconvertscreen.

Модуль включает следующий функционал:

1. Класс CustomTabBar(QTabBar) содержит методы \_\_init\_\_, setTabVisible, isTabVisible, tabSizeHint, paintEvent и управляет отображением вкладок.  
   Рассмотрим рисунок 8.17, на котором представлен исходный код класса CustomTabBar.

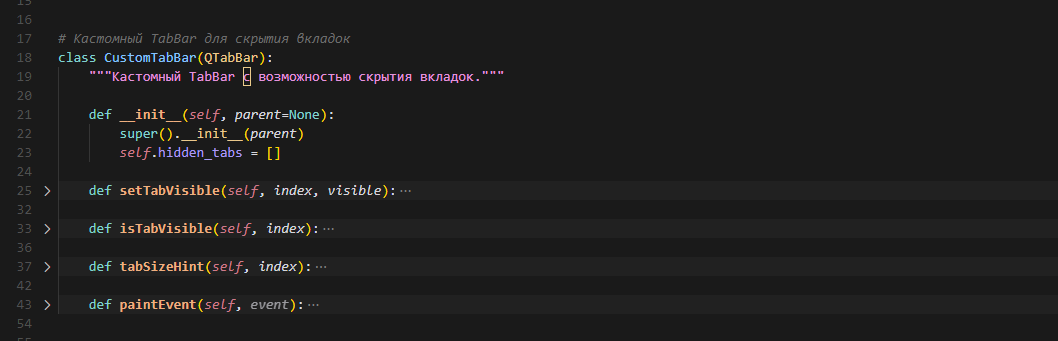


Рисунок 8.17 – Исходный код класса CustomTabBar

Мы рассмотрели рисунок 8.17, на нём изображены методы для управления видимостью и отображением вкладок с переопределением paintEvent.

1. Класс CustomTabWidget(QTabWidget) содержит методы \_\_init\_\_, setTabVisible, isTabVisible и использует CustomTabBar для управления вкладками.

Рассмотрим рисунок 8.18, на котором представлен исходный код класса CustomTabWidget.

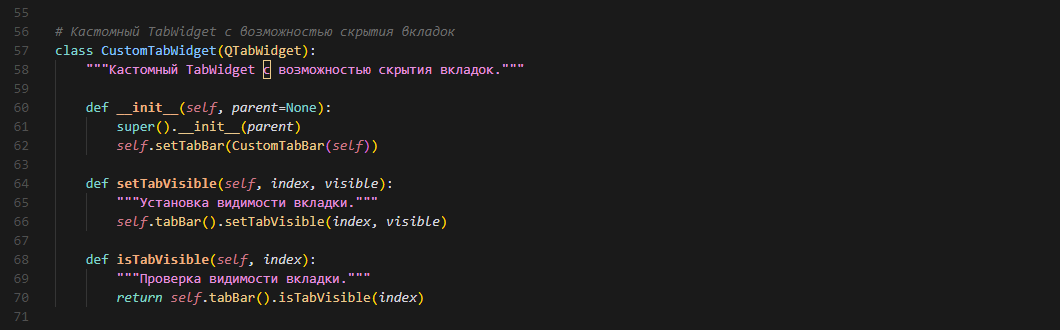


Рисунок 8.18 – Исходный код класса CustomTabWidget

Мы рассмотрели рисунок 8.18, на нём изображено создание вкладочного виджета с использованием CustomTabBar и методы для управления видимостью вкладок.

1. Класс MainWindow(QMainWindow) содержит методы \_\_init\_\_, \_get\_fallback\_font, \_load\_fonts, \_setup\_window, \_setup\_tab\_styles, \_update\_tab\_style, \_create\_tabs, \_create\_stub\_tab и создаёт окно с вкладками, включая VideoConvertScreen, доступную через DownloadScreen.

Рассмотрим рисунок 8.19, на котором представлен исходный код класса MainWindow.

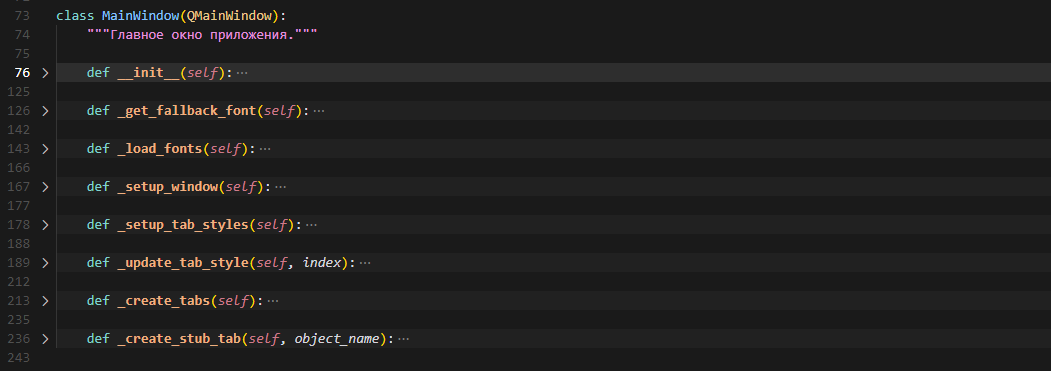


Рисунок 8.19 – Исходный код класса MainWindow

Мы рассмотрели рисунок 8.19, на нём изображены методы для инициализации окна, настройки шрифтов, стилей и создания вкладок.

1. Функция run\_app() запускает приложение.

Рассмотрим рисунок 8.20, на котором представлен исходный код функции run\_app из модуля ui/app.py.



Рисунок 8.20 – Исходный код функции run\_app

Мы рассмотрели рисунок 8.20, на нём изображено создание экземпляра QApplication и запуск MainWindow.

Средства защиты информации:

* проверка наличия шрифтов и файлов стилей с fallback на системные шрифты;
* обработка исключений при загрузке шрифтов и стилей;
* предупреждение пользователю при отсутствии шрифта monocraft;
* валидация путей к файлам (style.qss, шрифты);
* валидация параметров конвертации в videoconvertscreen перед передачей в video\_processor.py;
* ограничение размеров окна (setfixedsize) для предотвращения некорректного отображения.

Постараемся представить структурную схему работы программы. Вот её текстовое описание:

1. Пользователь запускает программу через main.py, который вызывает run\_app() из ui/app.py.
2. Модуль ui/app.py создаёт MainWindow, инициализирует интерфейс с вкладками (DownloadScreen, VideoEditScreen, VideoConvertScreen).
3. На вкладке "Скачивание" (DownloadScreen):
   * пользователь вводит url и выбирает папку;
   * downloadscreen создаёт videoinfofetcher для получения информации о видео;
   * после получения информации videodownloader скачивает видео в data/temp/, отправляя сигналы о прогрессе и статусе;
   * ошибки логируются в data/download\_errors.log;
   * пользователь нажимает кнопку "конвертер", переходя к videoconvertscreen.
4. На вкладке "Конвертер" (VideoConvertScreen):
   * пользователь выбирает скачанный файл и целевой формат (например, avi, webm);
   * videoconvertscreen создаёт команду ffmpeg через create\_ffmpeg\_command в video\_processor.py;
   * ffmpegprocessor выполняет конвертацию, сохраняя результат в data/output/, и отправляет сигналы о прогрессе.
5. На вкладке "Уникализация" (VideoEditScreen):

* пользователь задаёт параметры обработки (фильтры, обрезка);
* videoeditscreen создаёт команду ffmpeg через create\_ffmpeg\_command;
* ffmpegprocessor выполняет обработку, сохраняя результат в data/output/.

1. Статус операций отображается в интерфейсе через сигналы PyQt5.
2. Временные файлы очищаются через cleanup\_temp\_files.

Осуществим инкрементную интеграцию модулей

Подход Bottom-up - модули downloader.py и video\_processor.py независимы, их можно протестировать отдельно, затем интегрировать с ui/app.py через экраны и main.py.

Этапы интеграции:

1. Тестирование downloader.py:
   * тест videoinfofetcher: создать экземпляр с url (https://youtube.com/watch?v=example), проверить сигнал info\_ready с данными (title, uploader, etc.);
   * тест videodownloader: запустить скачивание с url и save\_path="data/temp", проверить создание файла и сигнал download\_complete;
   * проверка: лог ошибок в data/download\_errors.log, файлы в data/temp/.
2. Тестирование video\_processor.py:
   * тест create\_ffmpeg\_command: создать команду для конвертации (input\_path="data/temp/video.mp4", output\_path="data/output/video.avi", filter\_str=none), проверить корректность команды;
   * тест ffmpegprocessor: выполнить команду, проверить сигнал finished и создание файла в data/output/;
   * проверка: ошибки через сигнал error, удаление старых файлов через cleanup\_temp\_files.
3. Интеграция с ui/app.py:
   * подключение downloadscreen к videodownloader и videoinfofetcher для скачивания;
   * подключение downloadscreen к videoconvertscreen через кнопку "конвертер";
   * подключение videoconvertscreen и videoeditscreen к ffmpegprocessor для конвертации и обработки;
   * тест: запуск интерфейса, ввод url, скачивание, переход к конвертеру через кнопку, выбор формата, проверка результата.
4. Интеграция с main.py:
   * вызов run\_app() из main.py, запуск mainwindow
   * тест: полный цикл (запуск, скачивание, конвертация, обработка, отображение).

Средства тестирования:

* модульные тесты с unittest для downloader.py (проверка is\_valid\_url, extract\_video\_id) и video\_processor.py (проверка create\_ffmpeg\_command);
* ручное тестирование интерфейса через ui/app.py с реальными url, файлами и переходом к конвертеру.

Итогом внутреннего проектирования стало:

1. Разработаны модули main.py, downloader.py, video\_processor.py, app.py на основе предоставленного кода.
2. Оформлена внешняя спецификация модулей (описание, подключение, вход/выход, структура, защита).
3. Описана структурная схема работы программы с учётом конвертера.
4. Выполнена инкрементная интеграция модулей с подходом bottom-up.
5. Документ соответствует требованиям ГОСТ 19.201-78 и архитектуре Vidify.

# **9 ИНТЕГРАЦИЯ МОДУЛЕЙ В ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ**

После разработки и тестирования отдельных модулей системы Vidify была выполнена их интеграция в единое программное обеспечение. Интеграция осуществлялась по подходу инкрементной интеграции снизу вверх (Bottom-up), что позволило поэтапно подключать и проверять компоненты в процессе сборки.

Все модули разрабатывались независимо, имели чёткие интерфейсы взаимодействия и передавали данные между собой с помощью сигналов (в случае PyQt5) и вызовов функций.

Процесс интеграции осуществлялся последовательно, начиная с базовых модулей. Ниже приведены этапы интеграции основных компонентов системы Vidify.

На первом этапе был интегрирован модуль VideoInfoFetcher в экран загрузки DownloadScreen. Это позволило пользователю вводить URL, получать информацию о видео (название, обложка, автор) и отображать её до начала загрузки.

Рассмотрим рисунок 9.1, на котором показано подключение класса VideoInfoFetcher к DownloadScreen.

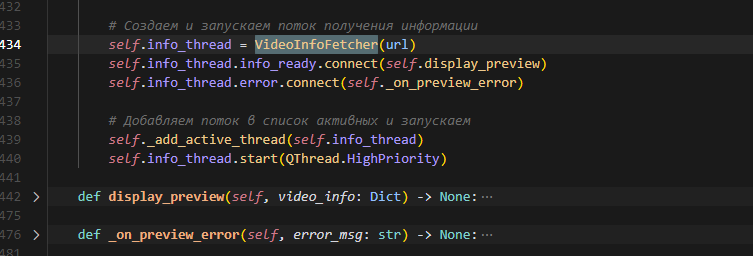


Рисунок 9.1 – Подключение VideoInfoFetcher к DownloadScreen

На изображении представлен код, реализующий создание экземпляра VideoInfoFetcher, подключение сигналов info\_ready и error, а также запуск получения информации по URL.

Следующим шагом стало добавление класса VideoDownloader в DownloadScreen, что обеспечило скачивание видео с помощью библиотеки yt\_dlp. Перед этим осуществляется валидация URL и проверка наличия соединения.

Рассмотрим рисунок 9.2, демонстрирующий интеграцию загрузчика видео.

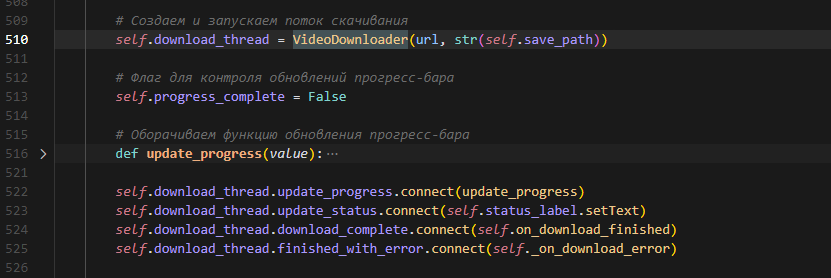


Рисунок 9.2 – Интеграция VideoDownloader в DownloadScreen

На изображении показано создание экземпляра VideoDownloader, подключение к сигналам download\_complete, update\_progress и finished\_with\_error, запуск процесса загрузки и отображение прогресса пользователю.

После завершения загрузки пользователь может перейти к конвертации видео. Для этого была реализована кнопка, открывающая экран VideoConvertScreen, передавая путь к файлу и необходимые параметры.

Рассмотрим рисунок 9.3, на котором изображён переход между экранами.

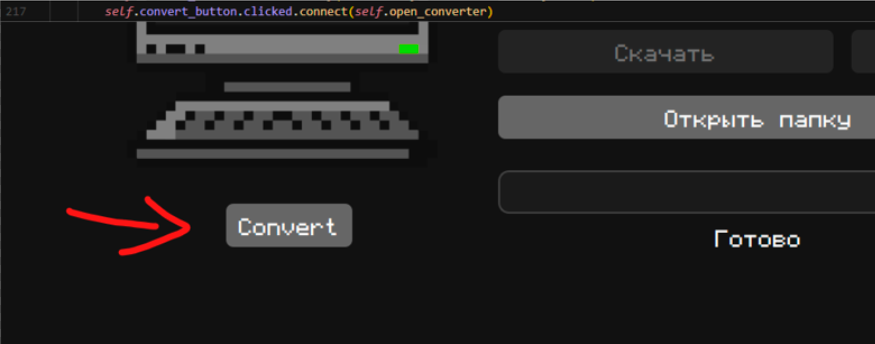


Рисунок 9.3 – Переход от DownloadScreen к VideoConvertScreen

На изображении отображён участок интерфейса с кнопкой "Конвертер" и соответствующий код, реализующий передачу информации между экранами.

Для выполнения конвертации видео интегрирован модуль FFmpegProcessor, запускающий subprocess с командой FFmpeg и отслеживающий прогресс.

Рассмотрим рисунок 9.4, демонстрирующий работу FFmpegProcessor с экраном VideoConvertScreen.

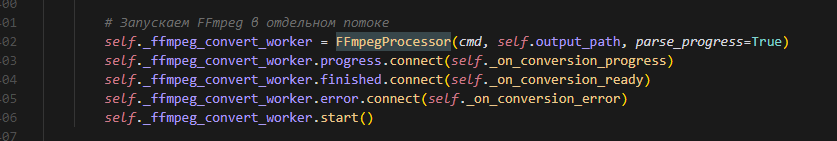


Рисунок 9.4 – Интеграция FFmpegProcessor в VideoConvertScreen

На изображении представлена схема вызова метода create\_ffmpeg\_command, передача его в FFmpegProcessor, запуск обработки и обновление прогресс-бара.

После создания всех экранов (DownloadScreen, VideoConvertScreen, VideoEditScreen) они были объединены в единое окно MainWindow с вкладочной навигацией.

Рассмотрим рисунок 9.5, на котором показано формирование главного окна приложения.

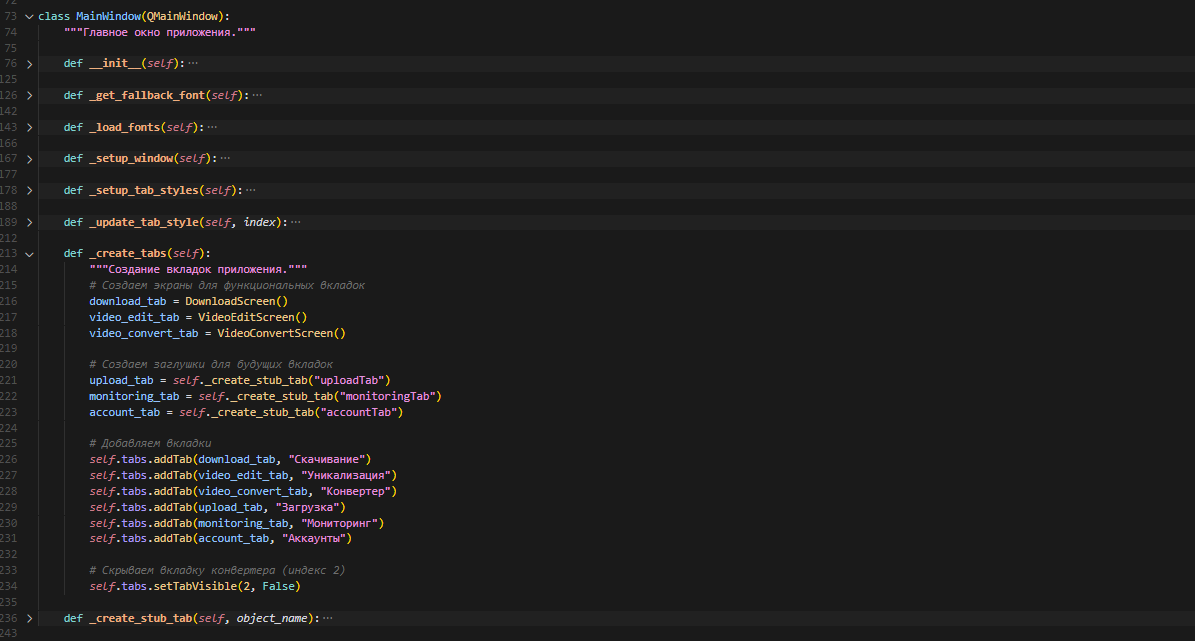


Рисунок 9.5 – Инициализация главного окна приложения

На изображении продемонстрировано создание вкладок в QTabWidget, добавление экранов, установка шрифтов и стилей, а также логика переключения между вкладками.

После полной сборки приложения выполнено системное тестирование, в ходе которого проверялись все возможные сценарии: ввод URL, скачивание, переход к конвертеру, настройка параметров и получение результата.

Рассмотрим рисунок 9.6, на котором представлено окно приложения во время системного тестирования.

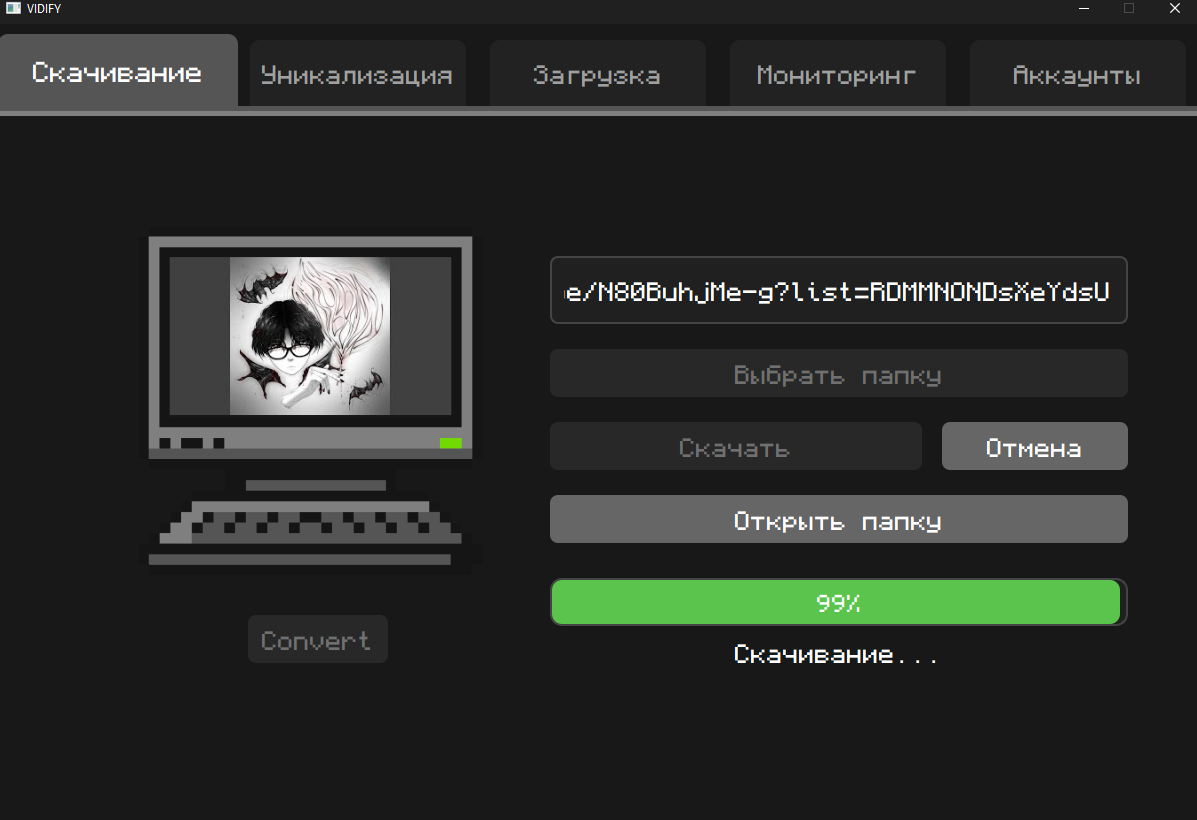


Рисунок 9.6 – Системное тестирование приложения Vidify

На изображении показан интерфейс Vidify с загруженным видео, активной конвертацией и отображением хода выполнения, подтверждающим корректную интеграцию всех модулей.

В результате выполнения задания была завершена интеграция всех модулей проекта Vidify в единое приложение. Интеграция выполнена итеративно (инкрементно), что обеспечило контроль над каждым этапом сборки и возможность оперативного выявления ошибок.

Основные результаты:

* модули videoinfofetcher, videodownloader, ffmpegprocessor и интерфейс mainwindow интегрированы в единую систему;
* все модули обмениваются данными с помощью сигналов и функций;
* интерфейс адаптирован под многовкладочный режим и обеспечивает интуитивную навигацию;
* системное тестирование подтвердило стабильную работу всей системы.

# **10 МОДИФИКАЦИЯ МОДУЛЕЙ ПРОЕКТА**

В ходе разработки проекта Vidify и проведения модульного тестирования были выявлены направления для улучшения функциональности отдельных компонентов. На этом этапе реализована модификация кода в модулях video\_processor.py и video\_edit\_screen.py. Цель изменений – обеспечить управление активными процессами FFmpeg, улучшить логирование и повысить стабильность работы при взаимодействии с пользователем.

На данном этапе были изменены два ключевых модуля, отвечающих за обработку видео и взаимодействие с пользователем через графический интерфейс. Изменения направлены на повышение удобства использования и контроля выполнения процессов. Описание модификаций приведено ниже:

1) Модуль video\_processor.py. Была добавлена возможность остановки процесса обработки видео. Это реализовано с помощью метода stop() в классе FFmpegProcessor. При вызове метода происходит завершение потока и остановка FFmpeg.

2) Модуль video\_edit\_screen.py В интерфейс добавлена кнопка «Отмена», которая позволяет пользователю вручную завершить процесс обработки видео. Сигнал от этой кнопки вызывает метод stop() из video\_processor.py.

Перейдем к разбору примера модификации: добавление метода stop(). Рассмотрим рисунок 10.1, на котором изображён код метода stop(), реализованного в классе FFmpegProcessor.

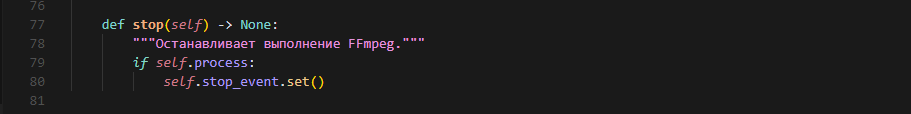


Рисунок 10.1 – Метод stop() в классе FFmpegProcessor

На изображении представлен фрагмент кода, позволяющий остановить активный процесс FFmpeg по команде из интерфейса пользователя.

Метод реализует проверку на наличие активного процесса и завершает его с помощью terminate(). Также устанавливается флаг \_stopped для контроля в основном потоке.

Для удобства отладки был доработан механизм логирования. Добавлена поддержка уровней логов (INFO, ERROR), а также сохранение временных меток.

Рассмотрим рисунок 10.2, на котором изображена функция логирования ошибок в файл ffmpeg\_log.log.

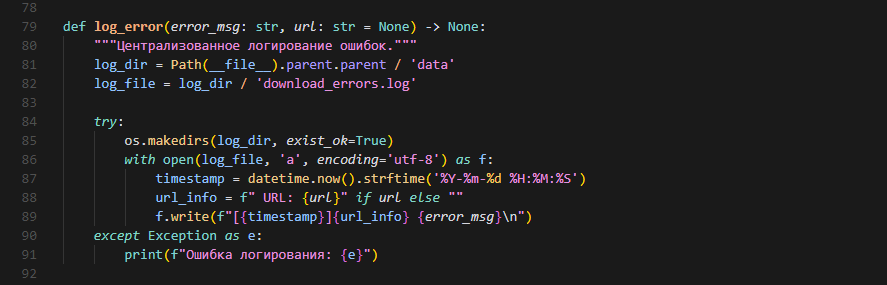


Рисунок 10.2 – Обновлённая функция логирования ошибок

На изображении показано, как формируется строка лога с отметкой времени и уровнем события. Это упрощает диагностику возникающих проблем.

Приступим к расширению интерфейса и добавлению кнопки «Отмена». Рассмотрим рисунок 10.3, на котором показан интерфейс с добавленной кнопкой отмены обработки видео.

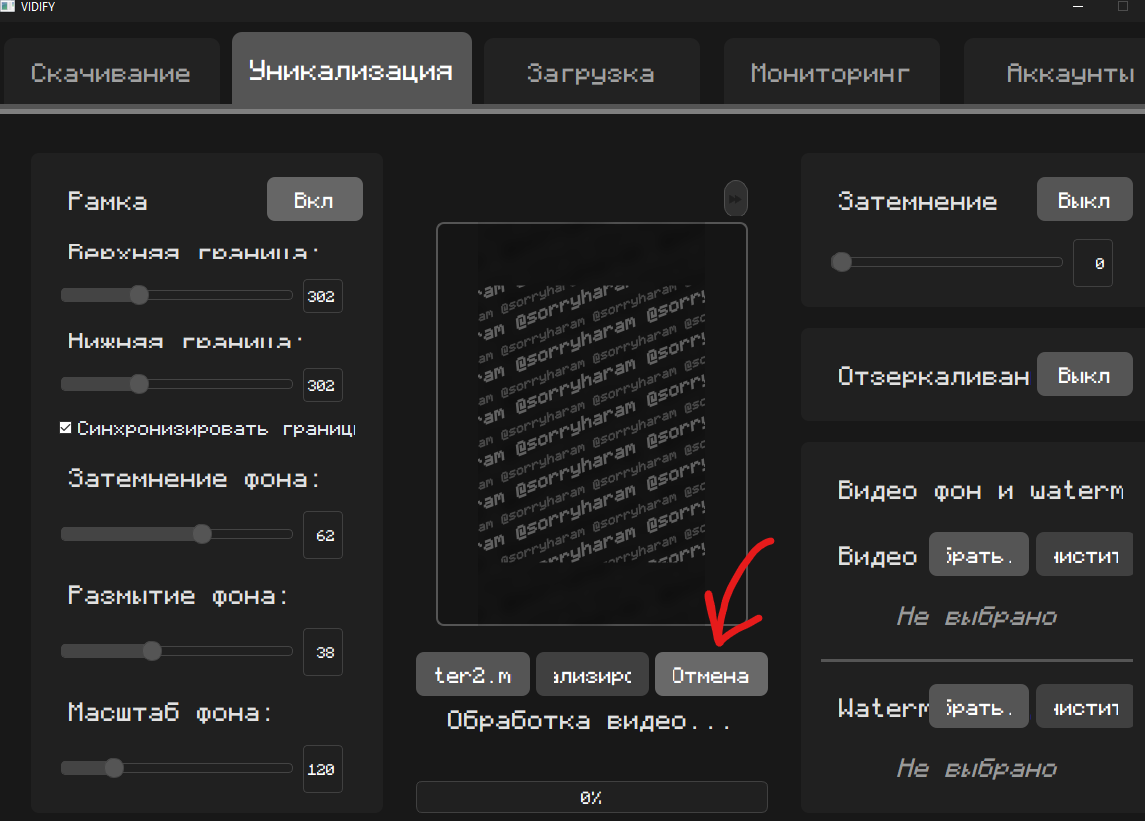


Рисунок 10.3 – Кнопка «Отмена» в окне редактирования видео

На изображении видно, что по нажатию кнопки отправляется сигнал на вызов метода stop() в процессе обработки. Это позволяет пользователю прерывать долгие или нежелательные операции.

Также реализовано отображение статуса отмены – в случае завершения процесса по команде отображается сообщение об остановке в UI.

После модификации компонентов было проведено ручное тестирование функционала:

* метод stop() успешно завершает активный процесс ffmpeg;
* кнопка «отмена» коррект но передаёт команду и завершает обработку;
* файл ffmpeg\_log.log содержит подробные записи событий;
* ui адекватно отображает статус отмены и ошибки.

Тестирование показало стабильную работу новых функций.

Модификация модулей позволила повысить управляемость процессами внутри приложения Vidify. Благодаря внедрению функции остановки, улучшенному логированию и элементам UI, взаимодействие пользователя с системой стало более гибким и удобным. Все изменения выполнены в рамках архитектуры проекта, протестированы и интегрированы без нарушения основной логики приложения.

# **11 ОТЛАДКА МОДУЛЕЙ ПРОГРАММНОГО ПРОЕКТА. ОРГАНИЗАЦИЯ ОБРАБОТКИ ИСКЛЮЧЕНИЙ**

В процессе разработки программного обеспечения Vidify важным этапом является отладка и организация обработки ошибок. Это позволяет обеспечить устойчивость системы к некорректным данным и неожиданным сбоям, а также повысить надёжность приложения в условиях пользовательских ошибок.

Для выполнения задания был выбран модуль video\_processor.py, отвечающий за выполнение ресурсоёмких операций – конвертацию и обработку видео с использованием FFmpeg. В частности, была проведена отладка класса FFmpegProcessor, который отвечает за запуск процесса обработки видео в отдельном потоке.

Отладка проводилась в среде разработки Cursor IDE с использованием встроенного отладчика. Основное внимание было уделено корректной обработке исключений, возникающих при ошибке запуска команды FFmpeg.

До внесения изменений в модуле отсутствовала полноценная обработка ошибок при запуске команды FFmpeg. В случае сбоя программа завершалась без уведомления пользователя, что снижало удобство использования и делало отладку затруднительной.

Рассмотрим рисунок 11.1, на котором представлен фрагмент кода до отладки. На нём изображён класс FFmpegProcessor, в котором отсутствует перехват исключений при выполнении команды.

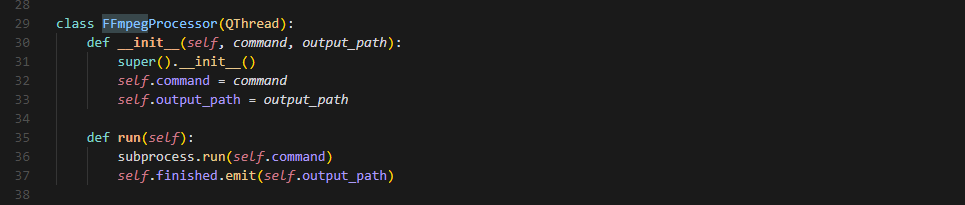


Рисунок 11.1 – Код до отладки модуля video\_processor.py

Мы увидели, что при возникновении ошибок FFmpeg программа не будет выводить никаких сообщений и аварийно завершит выполнение, не уведомляя пользователя.

После проведения отладки в класс FFmpegProcessor была добавлена обработка исключений с передачей сообщения об ошибке через сигнал error. Теперь при возникновении ошибки запуска команды пользователь получает уведомление в графическом интерфейсе, а программа продолжает работу.

Рассмотрим рисунок 11.2, на котором изображён код после отладки. На нём видно, что метод run() обёрнут в конструкцию try/except с перехватом ошибки subprocess.CalledProcessError. Ошибка передаётся в интерфейс с помощью сигнала PyQt5.

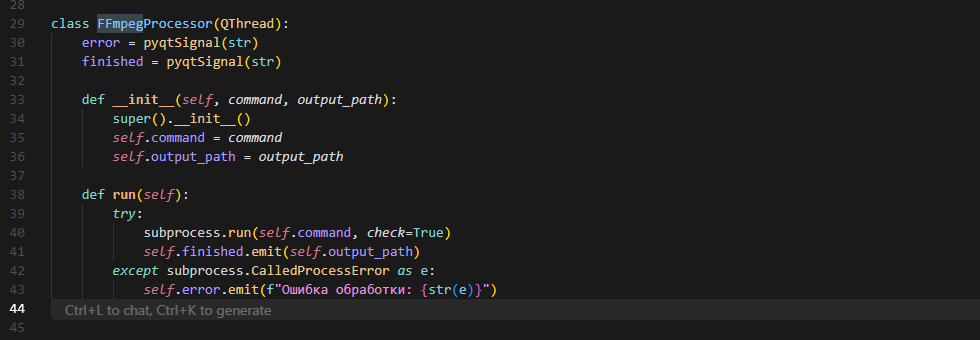


Рисунок 11.2 – Модифицированный код FFmpegProcessor с обработкой исключений

Мы увидели, что теперь программа при сбоях корректно перехватывает исключения и уведомляет пользователя через интерфейс.

В результате отладки были достигнуты следующие результаты:

1. Обеспечена устойчивость приложения к ошибкам вызова FFmpeg;
2. Пользователь получает сообщение об ошибке, а не наблюдает внезапное завершение программы;
3. Интерфейс информирует о причинах сбоя с помощью окна уведомления;
4. Поведение системы стало более предсказуемым и удобным для пользователя.

Проведённая отладка значительно повысила надёжность и удобство использования программного обеспечения Vidify. Обработка исключений теперь встроена в логику приложения, и её легко расширить при необходимости в будущем.

# **12 ОТЛАДКА ПРОЕКТА. ИНСПЕКЦИЯ КОДА МОДУЛЕЙ ПРОЕКТА**

Перед проведением инспекции кода была выбрана основная среда разработки – IDE Cursor. Объектами инспекции стали ключевые модули проекта Vidify: main.py, video\_processor.py, downloader.py и app.py. Цель – выявление синтаксических, логических и структурных ошибок до финального этапа оформления отчета.

Инспекция проводится после полной интеграции модулей и базового тестирования. План охватывает следующие аспекты:

* корректность потоков и сигналов pyqt5;
* безопасная работа с внешними библиотеками (yt\_dlp, ffmpeg);
* наличие проверки входных данных;
* обработка ошибок и исключений;
* читаемость и оформление кода.

Проверка выполняется вручную – без использования автоматических линтеров.

Проведем подготовку к нашей экспертизе. Были проанализированы следующие модули:

* video\_processor.py: проверена корректность выполнения команд FFmpeg, работа метода stop(), логика обновления прогресса;
* downloader.py: проанализированы функции is\_valid\_url, extract\_video\_id, классы VideoInfoFetcher и VideoDownloader. Убедились, что логирование ошибок реализовано через log\_error();
* app.py: проверена структура интерфейса и реализация переключения между вкладками, а также связь с модулями скачивания и обработки;
* main.py: проверен старт приложения, наличие конструкции if \_\_name\_\_ == "\_\_main\_\_" и вызов run\_app().

Ошибок, нарушающих работу программы, не обнаружено.

В процессе инспекции выявлены следующие сильные стороны проекта:

* все сигналы и слоты в pyqt5 реализованы корректно и чётко разделены по зонам ответственности;
* модули структурированы по папкам и логически разделены;
* исключения обрабатываются с логированием и уведомлением пользователя;
* архитектура кода позволяет легко масштабировать проект и вносить изменения.

В качестве рекомендаций предложено:

* добавить краткие docstring-описания к основным классам и функциям;
* вынести повторяющиеся пути (например, data/input/, data/output/) в отдельный конфигурационный модуль;
* добавить логирование успешных операций для полноты журналов.

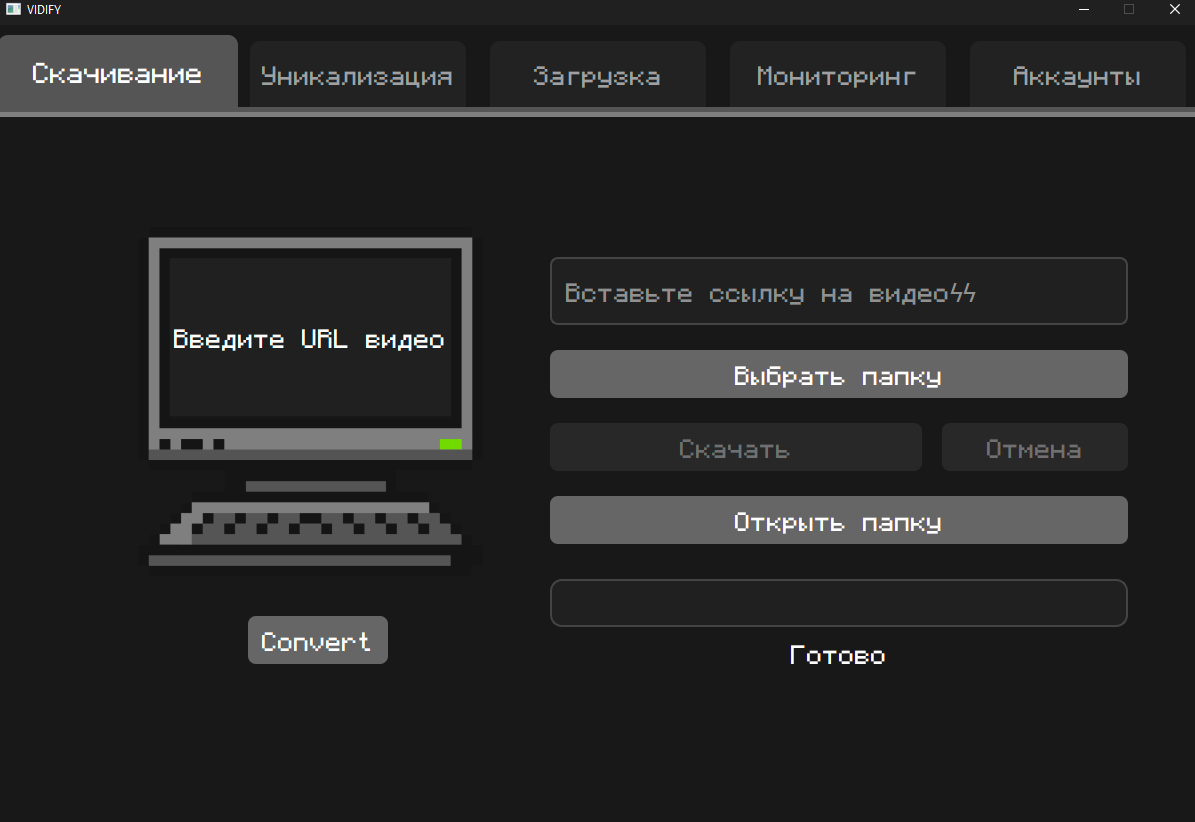
Инспекция завершена. Проект Vidify соответствует основным требованиям надёжности, читаемости и безопасности. Все основные модули работают корректно и готовы к финальному этапу – подготовке итогового отчета и демонстрации.

# **13 ТЕСТИРОВАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ СРЕДСТВАМИ ИНСТРУМЕНТАЛЬНОЙ СРЕДЫ РАЗРАБОТКИ. ВЫПОЛНЕНИЕ ФУНКЦИОНАЛЬНОГО ТЕСТИРОВАНИЯ**

В данном разделе рассматривается процесс тестирования пользовательского интерфейса приложения Vidify. Тестирование проводилось средствами среды разработки Cursor IDE, а также вручную, с анализом реакции интерфейса на типичные пользовательские действия. Основное внимание уделялось корректности отображения элементов интерфейса, логике переходов между вкладками и адекватной обработке ошибок.

Тестирование проводилось на интерфейсе MainWindow, реализованном в модуле ui/app.py.

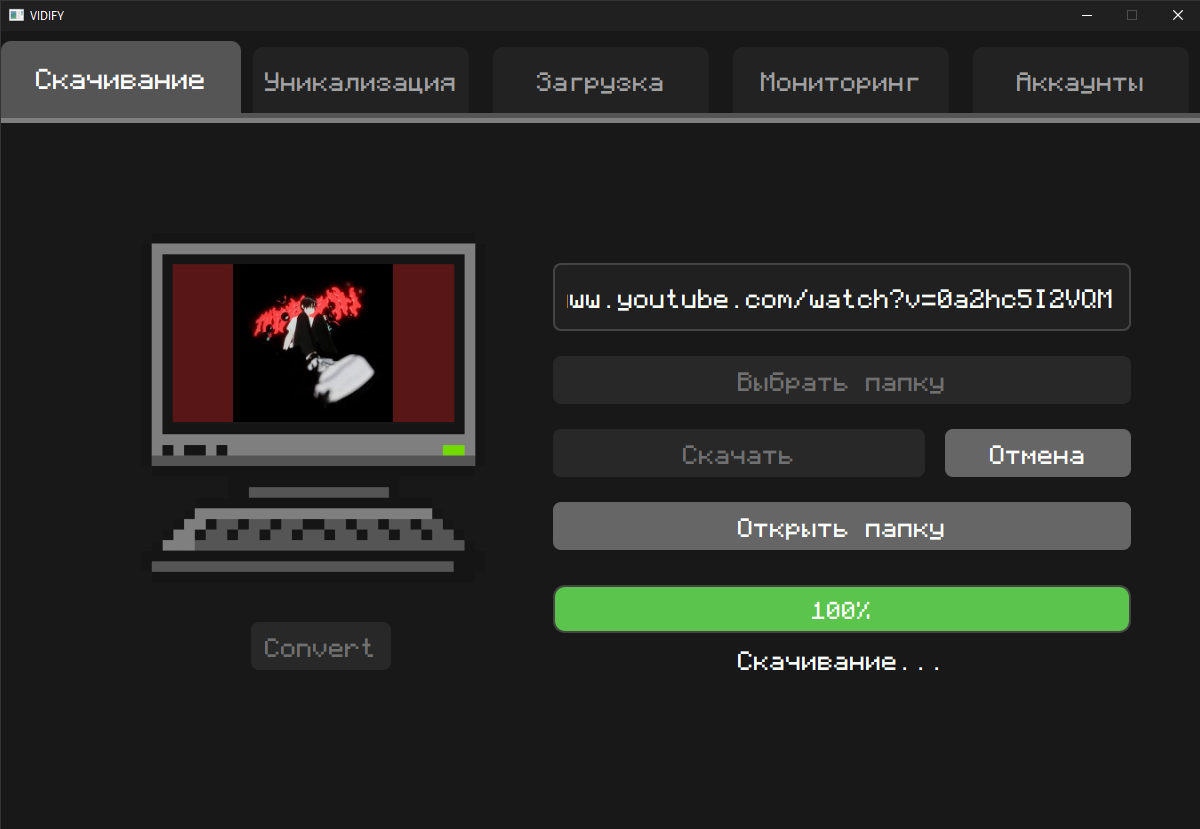
Рассмотрим рисунок 13.1, на котором изображено главное окно приложения Vidify при запуске.

  
Рисунок 13.1 – Главное окно Vidify после запуска

Мы рассмотрели рисунок 13.1 и убедились, что при запуске инициализируются все вкладки: «Скачивание», «Уникализация», «Конвертер». Все элементы загружаются без задержек и ошибок.

Проведена проверка ввода URL, старта загрузки и отображения прогресса.

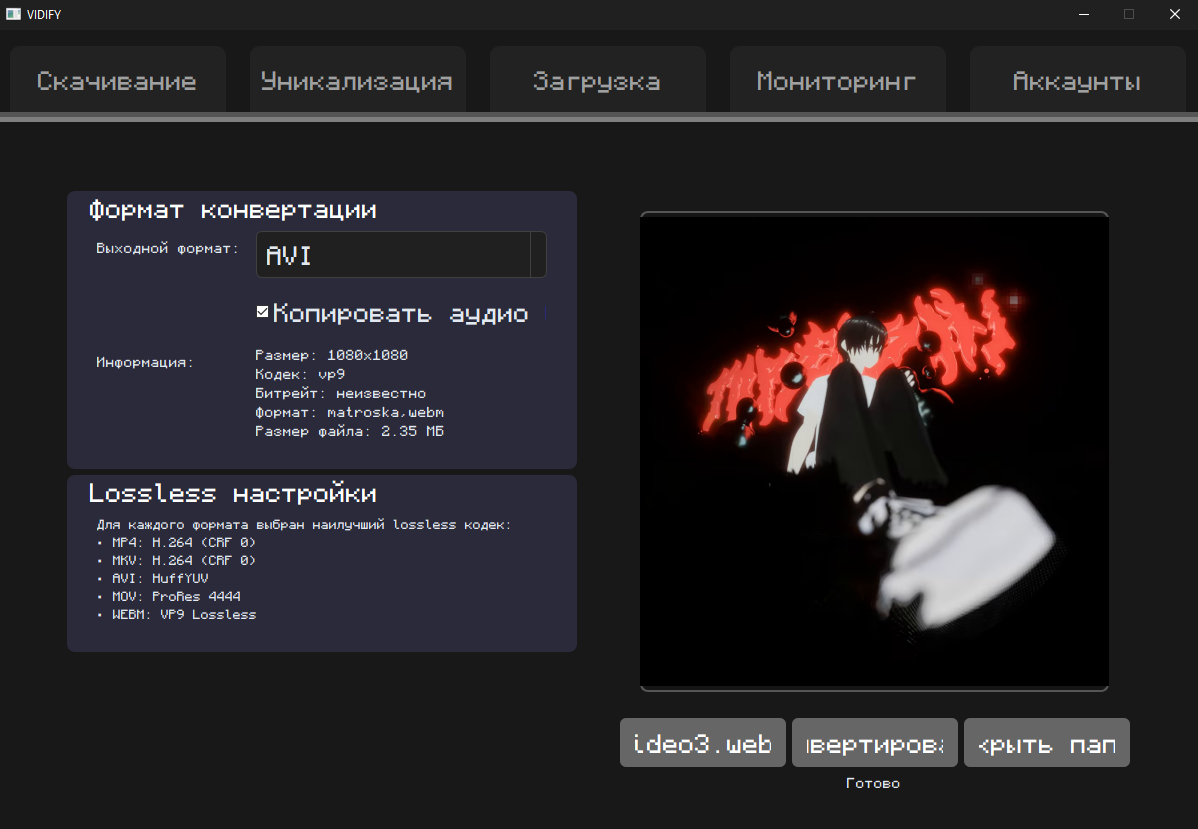
Рассмотрим рисунок 13.2, на котором показан ввод ссылки и отображение статуса загрузки.

  
Рисунок 13.2 – Ввод URL и процесс загрузки видео

Мы рассмотрели рисунок 13.2 и убедились, что при валидном URL система корректно показывает статус, генерирует превью и сохраняет видеофайл в папку data/input/.

На вкладке «Конвертер» проводилось функциональное тестирование: выбор формата, запуск конвертации и просмотр результата.

Рассмотрим рисунок 13.3, на котором отображены параметры конвертации видео.

  
Рисунок 13.3 – Настройка параметров конвертации

Мы рассмотрели рисунок 13.3 и убедились, что параметры применяются корректно, а итоговый файл сохраняется в папке data/output/. Конвертация сопровождается индикатором прогресса.

Тестировались функции наложения рамки, водяного знака и зеркалирования.

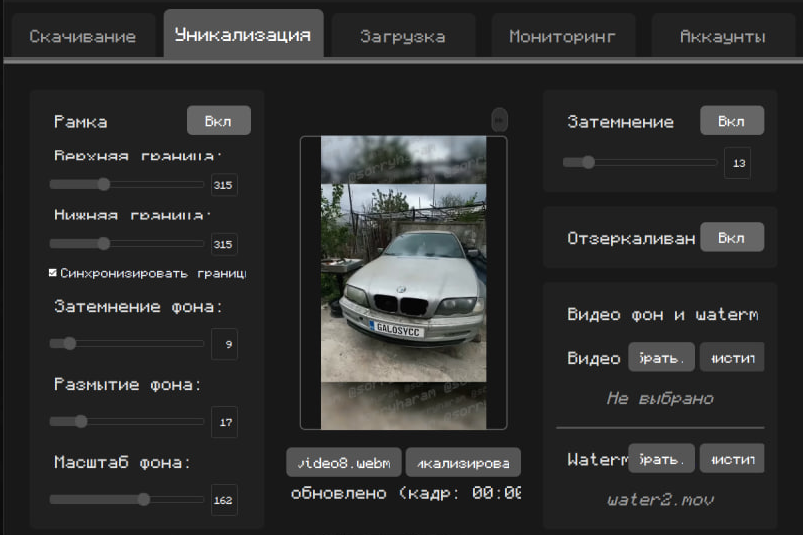
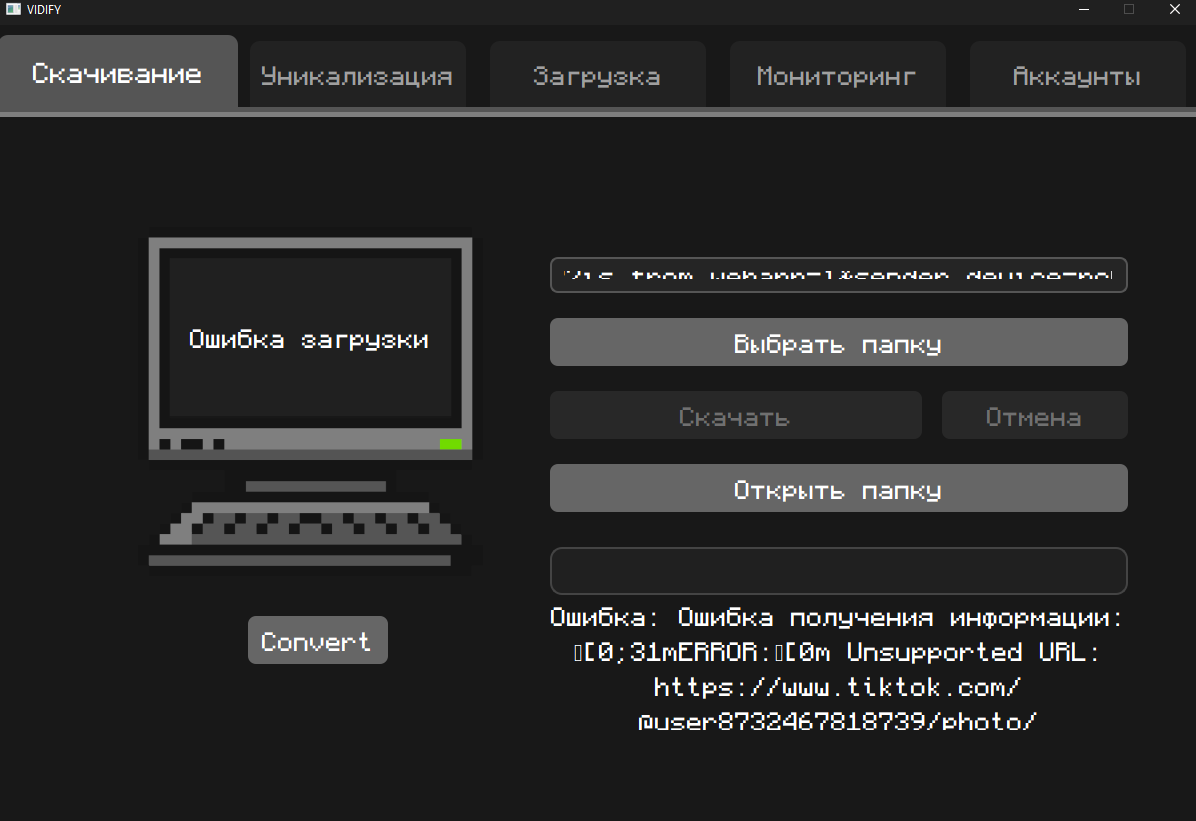
Рассмотрим рисунок 13.4, на котором показано применение эффектов и просмотр превью. 

Рисунок 13.4 – Применение эффектов уникализации

Мы рассмотрели рисунок 13.4 и убедились, что все фильтры применяются корректно, итог сохраняется как новый файл, а интерфейс отображает превью и статус выполнения.

Были протестированы сценарии с неверным URL, прерыванием загрузки и отсутствием интернета.

Рассмотрим рисунок 13.5, на котором показано отображение сообщения об ошибке.

  
Рисунок 13.5 – Обработка ошибки при загрузке

Мы рассмотрели рисунок 13.5 и убедились, что система адекватно реагирует на ошибки, записывает их в download\_errors.log и уведомляет пользователя через всплывающее окно.

Тестирование проводилось с использованием среды Cursor IDE, с применением встроенного отладчика, ручного анализа логов, сигналов PyQt5 и запусков через терминал.

Все ключевые сценарии были успешно выполнены. Приложение корректно реагирует на пользовательские действия, отображает прогресс, обрабатывает ошибки и обеспечивает сохранение результатов. Интерфейс устойчив к сбоям и готов к дальнейшему расширению.

# **ЗАКЛЮЧЕНИЕ**

В ходе прохождения учебной практики мной был выполнен полный цикл проектирования и разработки программного продукта Vidify – инструмента для автоматизированной работы с видеоконтентом, включающего загрузку, конвертацию и уникализацию видеофайлов.

Проектирование началось с анализа предметной области, где были выделены ключевые подсистемы, определены функции, а также сформулированы требования к системе. На основе собранной информации было составлено техническое задание в соответствии с ГОСТ, что позволило структурировать цели, задачи и характеристики программного продукта.

Разработка велась поэтапно: от построения архитектуры проекта и проектирования интерфейсов до реализации, отладки, модификации и интеграции программных модулей. В рамках практики были применены методы структурного и объектно-ориентированного анализа, создано множество диаграмм (IPO, IDEF0, DFD), что способствовало глубокой проработке логики функционирования системы. Работа с системой контроля версий Git и платформа GitHub обеспечили надёжное отслеживание изменений и контроль над ходом реализации проекта.

Особое внимание было уделено качеству кода и устойчивости системы. Были реализованы механизмы обработки исключений, логирования, а также проведена отладка. Все модули прошли инкрементную интеграцию, что подтвердило их корректное взаимодействие в составе целостного приложения.

Результатом практики стал полностью функционирующий проект, реализующий современные подходы к обработке мультимедийного контента. Я получил ценные навыки системного анализа, разработки модулей, работы с репозиторием и применения стандартов проектной документации. Опыт, полученный в ходе выполнения данного проекта, будет полезен при реализации более масштабных и комплексных решений в профессиональной деятельности.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. ГОСТ 7.32–2017. Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления. – Введ. 2017-10-24.
2. ГОСТ 19.201–78. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению.
3. Python Software Foundation. Официальная документация Python 3. – [Электронный ресурс]. URL: https://docs.python.org/3/ (дата обращения: 25.05.2025).
4. Git. Документация по системе контроля версий. – [Электронный ресурс]. URL: https://git-scm.com/doc (дата обращения: 25.05.2025).
5. GitHub. Репозиторий проекта Vidify. – [Электронный ресурс]. URL: https://github.com/sorryharam/vidify (дата обращения: 25.05.2025).
6. FastAPI. Современный веб-фреймворк для Python. – [Электронный ресурс]. URL: https://fastapi.tiangolo.com/ru/ (дата обращения: 25.05.2025).
7. MoviePy. Видеообработка на Python. – [Электронный ресурс]. URL: https://zulko.github.io/moviepy/ (дата обращения: 25.05.2025).
8. FFmpeg. Утилита для обработки видео и аудио. – [Электронный ресурс]. URL: https://ffmpeg.org/ (дата обращения: 25.05.2025).
9. Tkinter. Официальная документация по графическому интерфейсу в Python. – [Электронный ресурс]. URL: https://docs.python.org/3/library/tk.html (дата обращения: 25.05.2025).
10. PyDub. Библиотека для работы с аудио. – [Электронный ресурс]. URL: https://pydub.com/ (дата обращения: 25.05.2025).
11. Microsoft Docs. Документация по REST API. – [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/azure/architecture/best-practices/api-design (дата обращения: 25.05.2025).
12. OWASP. Безопасность REST API. – [Электронный ресурс]. URL: https://owasp.org/www-project-api-security/ (дата обращения: 25.05.2025).
13. JSON Specification. Спецификация формата данных. – [Электронный ресурс]. URL: https://www.json.org/json-en.html (дата обращения: 25.05.2025).
14. CodeProject. Руководства по разработке на Python. – [Электронный ресурс]. URL: https://www.codeproject.com/ (дата обращения: 25.05.2025).
15. Романов К.Ю. Программная инженерия: Учебное пособие. – М.: Академия, 2022. – 312 с. – ISBN 978-5-7695-9399-1.
16. Троелсен Э. Язык программирования C# и платформа .NET: учебник. – СПб.: Питер, 2020. – 896 с. – ISBN 978-5-4461-0922-7.
17. Шрайбер Т. Архитектура программного обеспечения. Паттерны проектирования. – М.: ДМК Пресс, 2021. – 560 с. – ISBN 978-5-97060-748-0.
18. Васильев А. Основы программирования на Python. – М.: Эксмо, 2023. – 384 с. – ISBN 978-5-04-120000-8.
19. Харрисон М. Python для профессионалов. – СПб.: БХВ-Петербург, 2022. – 496 с. – ISBN 978-5-9775-1555-1.
20. Visual Studio Code. Документация по редактору кода. – [Электронный ресурс]. URL: https://code.visualstudio.com/docs (дата обращения: 25.05.2025).