МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА ВЕБ-ПЛАТФОРМЫ ДЛЯ СОЗДАНИЯ ВИДЕО КОНТЕНТА

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Еременко Кирилла Дмитриевича

Научный руководитель
доцент, к. ф.-м. н. _____ А. С. Иванов

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2026

Актуальность работы. В современном цифровом мире видеоконтент является одним из наиболее востребованных и эффективных форматов коммуникации, обучения и развлечения. Процесс создания качественного видео, особенно для коммерческих каналов, таких как YouTube, VK Видео и Яндекс.Дзен, превратился из деятельности одиночек в сложный командный проект, вовлекающий сценаристов, монтажеров, дизайнеров, звукорежиссеров и менеджеров. Управление таким проектом, распределение задач, контроль сроков, отслеживание финансовых расчетов, включая процентные модели оплаты от дохода, и обеспечение прозрачности процесса для всех участников представляют собой нетривиальную организационную задачу. Проведенный анализ существующих решений показал, что универсальные системы управления проектами, такие как Trello, Asana или Jira, не адаптированы под специфику видеопроизводства: они не содержат встроенных моделей рабочих процессов, типичных для видеосъёмки, монтажа или графики, а также полностью отсутствуют механизмы для автоматизации финансовых расчетов по гибридным моделям. Критическим ограничением является и отсутствие глубокой интеграции с платформами видеохостинга через их API, что делает невозможным автоматический сбор данных о просмотрах и монетизации, необходимых для прозрачных расчетов с участниками проекта. Существующие специализированные сервисы решают лишь локальные задачи и не предоставляют инструментов для сквозного управления проектом от идеи до публикации. Таким образом, на рынке наблюдается явный пробел между мощными, но избыточными корпоративными системами и простыми, но неспециализированными трекерами задач, что подтверждает актуальность разработки специализированной веб-платформы.

Цель и задачи работы. Целью работы является разработка и реализация веб-платформы, которая автоматизирует и систематизирует процесс командного создания видеоконтента, обеспечивая управление проектами, пошаговое выполнение задач, гибкую систему расчетов с участниками и аналитику. Для достижения поставленной цели необходимо решить следующие задачи: провести анализ предметной области, изучить существующие решения и обосновать выбор технологического стека для разработки; спроектировать архитектуру веб-платформы, включая функциональные требования, структуру базы данных и схему взаимодействия компонентов системы; реализовать backend-часть системы на основе Spring Boot, включая REST API, бизнес-логику для управления

проектами, этапами, ролями пользователей и финансовыми расчетами, а также интеграцию с внешними API аналитики.

Структура и объём работы. Бакалаврская работа состоит из введения, двух разделов, заключения, списка использованных источников и двух приложений. Общий объём работы – 52 страницы, из них 45 страниц – основное содержание, включая 22 рисунка. Список использованных источников информации содержит 20 наименований.

Архитектура веб-платформы. Разработанная система реализована по классической трехуровневой клиент-серверной архитектуре, что обеспечивает четкое разделение ответственности, масштабируемость и удобство поддержки. Уровень представления функционирует непосредственно в браузере пользователя и отвечает за отображение интерфейса, интерактивные элементы и визуализацию данных. Серверная часть развернута на удаленном облачном сервере и служит вычислительным ядром приложения, отвечая за выполнение всей бизнес-логики: управление проектами и этапами, авторизацию пользователей, расчет вознаграждений и координацию рабочих процессов. Взаимодействие между клиентской и серверной частями организовано через REST API с использованием протокола HTTP, все данные передаются в формате JSON. Хранение и управление данными осуществляются в отдельной реляционной базе данных PostgreSQL, также размещенной на серверной инфраструктуре. Backend-приложение взаимодействует с СУБД через высокоуровневые абстракции JPA, что позволяет эффективно и безопасно выполнять операции создания, чтения, обновления и удаления данных, обеспечивая целостность и консистентность всей информации системы.

Серверная часть на Spring Boot. В качестве основного языка программирования серверной части была выбрана Java, что обусловлено ее зрелостью, кроссплатформенностью, строгой типизацией и широким распространением в разработке высоконагруженных enterprise-приложений. Основным фреймворком применен Spring Boot, который позволяет значительно ускорить и стандартизировать процесс разработки за счет предоставления готовых конфигураций. Spring Boot предлагает решения для инъекции зависимостей, управления транзакциями, работы с базами данных через Spring Data JPA, построения RESTful API с помощью Spring MVC и обеспечения безопасности через Spring Security. Для сокращения шаблонного кода и повышения читаемости использована биб-

лиотека Project Lombok, позволяющая с помощью аннотаций автоматически генерировать геттеры, сеттеры, конструкторы и другие методы. Для автоматической генерации интерактивной документации REST API интегрирован Swagger, который на основе аннотаций в коде контроллеров создает детальное описание всех доступных эндпоинтов, их параметров и форматов запросов и ответов.

Обеспечение безопасности системы. Безопасность доступа к данным и функционалу платформы реализована на нескольких уровнях. Первым уровнем защиты является использование протокола HTTP/2 поверх TLS, что исключает перехват и подмену данных при передаче по сети. Выбор HTTP/2 обусловлен его преимуществами: мультиплексирование позволяет отправлять множество запросов параллельно по одному соединению, а сжатие заголовков уменьшает объем передаваемых данных. Вторым уровнем защиты является обратный прокси-сервер nginx, который завершает TLS-соединение, обеспечивает балансировку нагрузки и маршрутизацию, а также настроен на отдачу статических файлов, что снижает нагрузку на Spring Boot-приложение. Третьим уровнем безопасности является система аутентификации на основе JWT. При регистрации пароль хешируется с помощью алгоритма bcrypt, после успешной проверки сервер генерирует JWT-токен, содержащий информацию о пользователе, и подписывает его секретным ключом. В отличие от классических сессий на сервере, JWT позволяет горизонтально масштабировать систему без необходимости синхронизировать хранилище сессий между экземплярами приложения.

Структура и технологии хранения данных. Выбор реляционной базы данных PostgreSQL обусловлен комплексом требований к надежности, функциональности и согласованности данных. Архитектура хранения медиафайлов, таких как исходные видео, готовые ролики и графические макеты, вынесена за пределы реляционной базы данных, поскольку хранение бинарных объектов большого размера в БД неэффективно. Вместо этого используется объектное хранилище MinIO, представляющее собой open-source систему, совместимую с API Amazon S3. MinIO можно развернуть на собственных серверах, что дает полный контроль над данными и исключает расходы на оплату внешнего облачного трафика и хранения. Для видеопродакшна, где объемы файлов могут достигать десятков гигабайт, локальное размещение становится существенно более экономичным. MinIO также поддерживает разбиение на части, что удобно для загрузки тяжелых видео через нестабильное соединение.

Контейнеризация и развертывание. Для обеспечения согласованности работы всех компонентов системы в различных средах и упрощения процесса развертывания применена технология контейнеризации на базе Docker. Каждый ключевой сервис платформы упакован в отдельный контейнер: backend-приложение на Spring Boot, база данных PostgreSQL, кэш-сервер Redis и объектное хранилище MinIO. Все зависимости, настройки и версии программного обеспечения зафиксированы в Dockerfile и docker-compose-конфигурациях, что гарантирует идентичное поведение системы на любом сервере. Помимо контейнеризации, в проекте настроен пайплайн непрерывной интеграции и непрерывной доставки на базе GitLab. При каждом пуше в основную ветку репозитория автоматически запускается сборка Docker-образов и обновление доставляется на сервер. Это позволило сократить время выхода новых функций с нескольких часов до нескольких минут.

Использование gRPC и Redis. Взаимодействие между основным backend-приложением на Spring Boot и сервисом аналитики организовано по протоколу gRPC, который использует бинарный протокол на основе Protocol Buffers, обеспечивая более высокую скорость сериализации и меньший объем передаваемых данных по сравнению с текстовым JSON. Для сервиса, который регулярно передает списки видео с метриками, это дает заметный выигрыш в производительности. Для ускорения работы системы и экономии запросов к внешним API видеохостингов используется Redis — быстрое хранилище данных в оперативной памяти. Когда пользователь запрашивает статистику по проекту, система сначала проверяет наличие данных в кэше. Если данные есть и не устарели, ответ возвращается мгновенно, без обращения к внешнему API. Если данных в кэше нет, система делает запрос к внешнему API, сохраняет результат в Redis и отдает его пользователю. Каждая запись живет в Redis один час, что гарантирует актуальность статистики не реже одного раза в час.

Функциональные требования. Перед непосредственной реализацией веб-платформы был проведён этап проектирования, включающий анализ интерфейса и пользовательского опыта. Разработан тестовый макет пользовательского интерфейса, который позволил визуализировать основные экраны платформы, продумать навигацию между ними и выявить все необходимые сущности. Страница авторизации содержит вкладки «Вход» и «Регистрация», поля для ввода логина и пароля, а также проверку уникальности никнейма при регистрации.

После успешного входа пользователь попадает на страницу профиля, где отображаются его проекты в виде карточек. Здесь же находятся карточки приглашений в новые проекты с кнопками принятия и отклонения, а также карточка для создания нового проекта. При нажатии на карточку проекта открывается главная страница проекта с панелью навигации, содержащей ссылки на разделы «Видео», «Команда», «Аналитика» и для владельца — «Настройки проекта». На странице «Видео» все видео представлены карточками, окрашенными в зависимости от статуса (просрочено, в работе, выполнено). На странице команды отображаются участники с их ролями и количеством непрочитанных сообщений; при нажатии на карточку участника открывается чат с возможностью отправки и редактирования сообщений. Страница аналитики отображает статистику с внешнего видеохостинга: карточки видео с просмотрами, лайками и датой публикации, а также общую статистику по каналу. Страница настроек проекта доступна только владельцу и позволяет изменять параметры проекта, исключать участников и приглашать новых пользователей с указанием роли. Ключевой элемент платформы — канбан-доска с этапами для каждого видео. Карточки задач имеют цветовое обозначение (выполнена, в работе, недоступна) и отображают название, описание и требуемую роль. Владелец проекта может добавлять новые стадии, задавая их зависимость от других стадий. При нажатии на карточку открывается окно с подробной информацией, где пользователь с соответствующей ролью может прикрепить файлы (с отображением прогресса загрузки), а любой участник может скачать прикреплённые файлы. Создание видео организовано в два этапа: сначала заполняются название, описание и дедлайн, затем выбирается — скопировать стадии из существующего видео или создать новые стадии с нуля, с возможностью добавления произвольного количества стадий. Все описанные сценарии полностью реализованы и обеспечивают комфортную командную работу над видеоконтентом.

Структура базы данных. На основе анализа функционала и разработанных макетов интерфейса спроектирована реляционная база данных, состоящая из шести основных сущностей: users, projects, connections, videos, stages и files, а также таблицы messages для хранения переписки. Таблица users хранит информацию о пользователях, включая уникальный никнейм и хешированный пароль. Таблица projects содержит данные о каналах, включая название, описание и ссылку на внешний видеохостинг. Таблица connections реализует связь многие-

ко-многим между пользователями и проектами с атрибутами role и accepted, что позволяет реализовать механизм приглашений. Таблица videos содержит информацию о каждом видео в рамках проекта, включая дедлайн и процент выполнения. Таблица stages является ключевой для организации рабочего процесса: каждая запись представляет задачу с указанием требуемой роли, при этом поле depends_on_stage_id позволяет строить иерархические зависимости этапов, а поле level хранит уровень вложенности для оптимизации отображения канбан-доски. Таблица files хранит метаданные о прикрепленных файлах с указанием пути в MinIO. Таблица messages обеспечивает функционал обмена сообщениями с поддержкой редактирования и отслеживанием прочтения.

Архитектура серверной части и слои приложения. Серверная часть платформы реализована как монолитное приложение на Spring Boot, организованное по многослойной архитектуре, включающей три основных слоя: контроллеры, сервисы и репозитории. Уровень репозиториев с использованием Spring Data JPA отвечает за взаимодействие с базой данных. Уровень сервисов содержит бизнес-логику приложения, инкапсулирует сложные операции, включающие взаимодействие с несколькими репозиториями, вызовы внешних API и выполнение транзакций. Уровень контроллеров является точкой входа для HTTP-запросов, отвечает за прием запросов, валидацию входящих данных и возврат клиенту объектов в формате JSON. Для каждой сущности реализованы стандартные CRUD-операции, а также дополнительные эндпоинты для выполнения специфичных задач, выявленных при анализе пользовательских сценариев. Все эндпоинты спроектированы в соответствии с принципами REST: ресурсы идентифицируются через URL, методы HTTP соответствуют выполняемым операциям, а ответы возвращаются с корректными HTTP-статусами. Все операции с чувствительными данными требуют подтверждения прав доступа, что реализовано на уровне сервисов с проверкой роли текущего пользователя.

Сервис аналитики и интеграция с внешними API. Для сбора статистики о просмотрах, подписчиках и доходах от монетизации разработан отдельный сервис аналитики, выделение которого продиктовано несколькими причинами. Работа с внешними API связана с сетевыми задержками и возможными сбоями, поэтому вынос этой логики из основного приложения изолирует потенциальные проблемы. Сбор статистики выполняется в фоновом режиме, а само обращение к внешнему API является долгой операцией, которую нежелательно выполнять в

потоке основного приложения. Такое разделение также упрощает масштабирование: при росте числа каналов можно запустить несколько экземпляров сервиса аналитики независимо от остальной системы. Взаимодействие между основным backend-приложением и сервисом аналитики организовано по протоколу gRPC, который поддерживает строгую типизацию данных через proto-файлы, исключая ошибки при парсинге ответов. Для оптимизации использования токенов доступа реализовано кэширование результатов с помощью Redis: при первом запросе статистики система обращается к сервису аналитики, сохраняет результат в Redis и возвращает пользователю, а при повторных запросах в течение часа данные возвращаются мгновенно из кэша.

Аутентификация и безопасность на основе JWT. Система аутентификации на основе JWT выстроена следующим образом. При входе пользователя сервер получает логин и пароль, при этом пароль хешируется перед сохранением в базе данных. При попытке входа система проверяет соответствие введенного пароля сохраненному хешу, и если данные верны, сервер генерирует JWT-токен, содержащий идентификатор пользователя, дату выдачи и время истечения, подписанный секретным ключом. Клиент сохраняет полученный токен и при каждом последующем запросе добавляет его в HTTP-заголовок в формате Bearer. На сервере специальный фильтр перехватывает каждый запрос, извлекает токен, проверяет его валидность и подпись, после чего извлекает идентификатор пользователя, создает объект аутентификации и помещает его в контекст безопасности. Такая архитектура обеспечивает изолированность механизма аутентификации от остальной бизнес-логики. Для операций, требующих специфических прав, например удаления проекта, сервис проверяет, что идентификатор пользователя из токена совпадает с идентификатором владельца проекта, и при отсутствии совпадения доступ к операции запрещается.

Контейнеризация и непрерывная интеграция. Сборка Docker-образа для основного приложения организована в два этапа: на первом этапе используется образ с Maven и JDK для компиляции исходного кода и сборки JAR-файла, на втором этапе используется более легкий образ с JRE, в который копируется только собранный JAR-файл, что позволяет уменьшить размер финального образа. Все сервисы описаны в docker-compose конфигурации, что позволяет запускать систему целиком одной командой. Для автоматизации сборки и развертывания настроен CI/CD-пайплайн в GitLab. При каждом пуше в основную

ветку репозитория запускается пайплайн, состоящий из стадий сборки и развертывания. На стадии сборки создается Docker-образ приложения, на стадии развертывания выполняются остановка старых контейнеров и запуск обновленных. Внедрение CI/CD и контейнеризации позволило автоматизировать процесс доставки кода на сервер, ускорить разработку и обеспечить согласованность работы всех компонентов системы в различных средах.

Финальная архитектура системы. Итоговая архитектура веб-платформы (рисунок 1) объединяет все описанные компоненты. Клиентская часть представляет собой фронтенд, который отвечает за отображение интерфейса, обработку действий пользователя и отправку запросов к серверной части. Все запросы передаются по защищенному протоколу HTTPS и попадают на сервер nginx, выступающий в роли обратного прокси. Nginx проксирует запросы к соответствующим эндпоинтам REST API основного сервиса. Основной сервис платформы реализован на Java с использованием фреймворка Spring Boot и содержит всю бизнес-логику приложения. Данные пользователей хранятся в реляционной базе данных PostgreSQL, а файлы — в объектном хранилище MinIO. Для ускорения доступа к статистике используется кэш-сервер Redis, а для получения данных с внешних видеохостингов выделен отдельный сервис аналитики, взаимодействующий с основным приложением через gRPC. Все компоненты упакованы в Docker-контейнеры и управляются через CI/CD-пайплайн.

Основные результаты работы. Было проведено комплексное исследование, посвященное анализу процессов командного создания видеоконтента и разработке инструментов для их автоматизации. В результате исследования были выявлены ключевые проблемы современных медиа-команд, включая фрагментацию рабочих процессов, отсутствие специализированных решений для управления видеопроектами, невозможность автоматического сбора данных о монетизации с видеохостингов и отсутствие гибких механизмов финансовых расчетов. На основании этого анализа были сформулированы требования и разработана архитектура веб-платформы, призванная решить обозначенные проблемы. Основными требованиями стали: поддержка пошаговой декомпозиции видеопроектов на зависимые этапы с назначением ролей, наличие интерактивной канбан-доски для визуализации прогресса, возможность обмена файлами между участниками на разных этапах, встроенная система чатов для коммуникации команды, а также интеграция с внешними API видеохостингов для

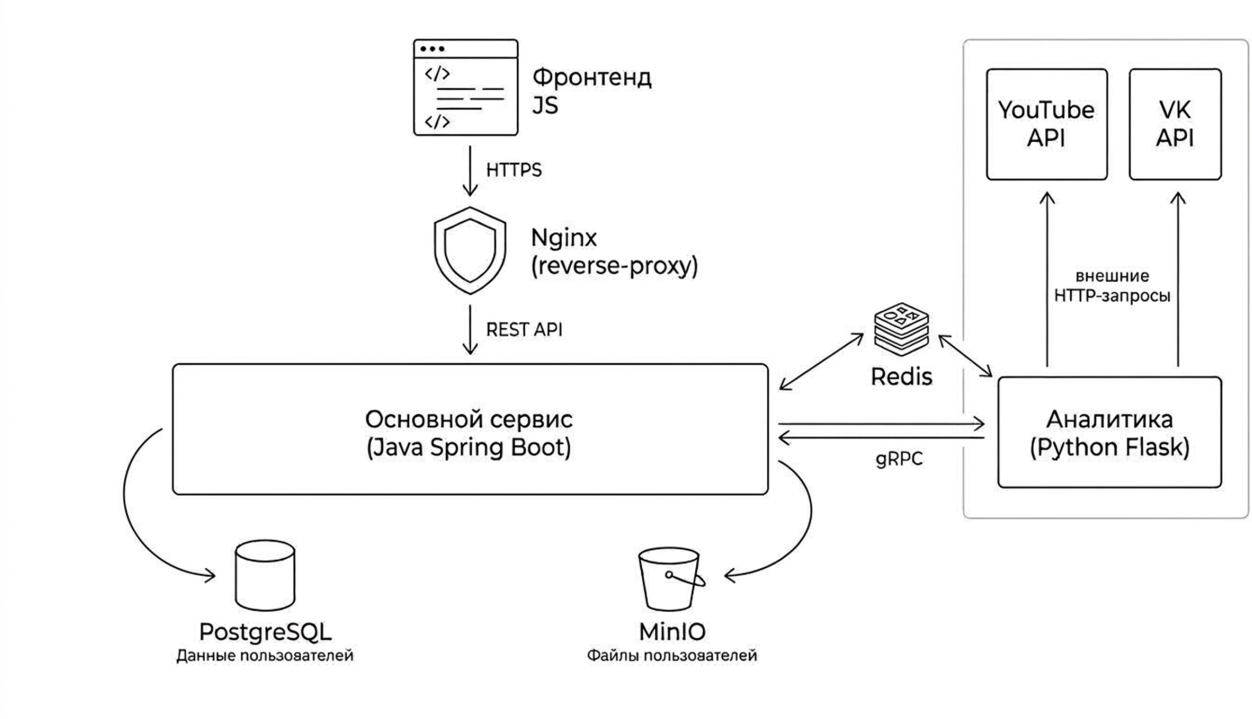

Рисунок 1 – Архитектура платформы

автоматического получения статистики просмотров и дохода от монетизации.

В ходе практической реализации была создана полнофункциональная система, включающая базовый набор необходимых функций. Разработана подсистема управления пользователями с регистрацией, аутентификацией на основе JWT-токенов и разграничением прав доступа. Реализовано управление проектами, позволяющее создавать каналы, приглашать участников с назначением конкретных ролей. Создана подсистема управления видео, обеспечивающая пошаговую декомпозицию производства на этапы с возможностью задания зависимостей между ними — этап становится доступным только после завершения всех предшествующих этапов, от которых он зависит. Реализована интерактивная канбан-доска с цветовой индикацией статуса каждой задачи: зеленый для выполненных, оранжевый для находящихся в работе, серый для недоступных. Разработана система обмена файлами между этапами с отображением прогресса загрузки, где каждый участник с соответствующей ролью может прикреплять, удалять и скачивать файлы. Создана система чатов для коммуникации участников команды с поддержкой редактирования и удаления сообщений, а также отслеживанием непрочитанных сообщений. Отдельное внимание уделено интеграции с внешними API YouTube и VK для получения статистики по каналам и видео, что позволяет участникам проекта отслеживать показатели просмотров,

лайков и подписчиков и использовать их для прозрачных финансовых расчетов. Для оптимизации запросов к внешним API реализовано кэширование статистики с использованием Redis, что сокращает количество токенизируемых запросов и ускоряет загрузку страниц аналитики с нескольких секунд до миллисекунд.

Разработанная архитектура основана на современных технологиях. Серверная часть реализована на Java с использованием фреймворка Spring Boot, обеспечивающего мощные возможности для инъекции зависимостей, управления транзакциями и построения REST API. В качестве реляционной базы данных для хранения структурированных данных о пользователях, проектах, видео, этапах, файлах и сообщениях выбрана PostgreSQL, что обусловлено ее надежностью, функциональностью и производительностью. Для хранения медиафайлов, таких как исходные видео, готовые ролики и графические макеты, используется объектное хранилище MinIO, совместимое с API Amazon S3, что позволяет развернуть хранилище на собственных серверах и полностью контролировать размещение данных. Кэш-сервер Redis применяется для оптимизации запросов к внешним API видеохостингов, обеспечивая время жизни кэшированных данных в один час и сокращая нагрузку на внешние сервисы. Взаимодействие с отдельным сервисом аналитики организовано по протоколу gRPC, который обеспечивает более высокую скорость сериализации за счет бинарного протокола на основе Protocol Buffers и строгую типизацию данных. Все компоненты системы упакованы в Docker-контейнеры, а процесс сборки и развертывания автоматизирован с помощью CI/CD-пайплайна на базе GitLab, который при каждом пуше в основную ветку репозитория автоматически собирает Docker-образы и доставляет обновление на сервер. Такое решение обеспечивает высокую производительность, масштабируемость и надежность платформы, а также позволяет новым разработчикам быстро развернуть окружение для работы над проектом.

Проведенное исследование и созданный рабочий прототип составляют существенную основу для последующей доработки платформы. В процессе разработки был выявлен ряд направлений для дальнейшего развития системы. В перспективе платформа может быть расширена добавлением полноценного механизма финансовых расчетов с автоматическим начислением вознаграждений участникам. Возможна интеграция с дополнительными видеохостингами, такими как Яндекс.Дзен и Rutube, что расширит аудиторию потенциальных

пользователей. Планируется внедрение системы уведомлений по электронной почте и push-уведомлений в браузере для информирования участников о приближении дедлайнов, изменении статусов этапов и новых сообщениях в чатах. Также необходимо полномасштабное тестирование с привлечением реальных медиа-команд для выявления дополнительных требований и сценариев использования. Внедрение разработанной платформы в реальные рабочие процессы медиа-индустрии позволит существенно повысить прозрачность и эффективность командного создания видеоконтента, централизовав все процессы управления проектами, задачами, файлами, коммуникацией и аналитикой в едином пространстве и устранив необходимость использования нескольких разрозненных инструментов.