# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

В современном цифровом мире визуальный контент играет ключевую роль в коммуникации, образовании, развлечениях и профессиональной деятельности. Создание динамичных анимаций на основе статичных изображений остается ресурсоемкой задачей, требующей специализированных навыков и значительных временных затрат. Актуальность автоматизации этого процесса обусловлена растущим спросом на персонализированный и адаптивный визуальный контент в различных прикладных сферах – от кинематографа и видеоигр до AR/VR-приложений и социальных медиа.

Современные достижения в области искусственного интеллекта, в частности методы глубокого обучения, открывают новые возможности для преобразования статических изображений в реалистичные анимации. Эти технологии позволяют алгоритмам «понимать» пространственные и временные закономерности визуальных данных, генерируя плавные переходы между кадрами без прямого вмешательства человека.

Целью данного дипломного проекта является разработка программного решения, автоматизирующего процесс создания анимаций из изображений с использованием передовых методов машинного обучения.

Архитектура решения строится на модульном подходе, где независимые компоненты системы (интерфейс взаимодействия, вычислительные модули и хранилища данных) взаимодействуют через стандартизированные интерфейсы, обеспечивая гибкость, масштабируемость и надежность.

Для достижения цели поставлены следующие задачи:

– изучение существующих методов анимации изображений;

– реализация нейронной сети для анимации изображений;

– разработка пользовательского интерфейса для работы с моделью;

– тестирование, сбор метрик и оценка качества результатов работы.

Практическая ценность работы заключается в снижении порога входа для создания анимации, что может способствовать ускорению рабочих процессов в креативных индустриях и расширению возможностей визуализации данных. Модульная архитектура обеспечивает не только удобство использования, но и возможность дальнейшего расширения функционала, включая интеграцию новых алгоритмов и адаптацию под различные платформы.

Данный дипломный проект выполнен мной лично, проверен на заимствования, процент оригинальности составляет 95,84% (отчет о проверке на заимствования прилагается).

# 1 ОБЗОР ЛИТЕРАТУРЫ

## 1.1 Аналоги современных решений

На сегодняшний день существует ряд программных решений и онлайн-сервисов, которые используют нейросетевые методы для создания анимации из статичных изображений.

**1.1.1** RunwayML [1] – это платформа, ориентированная на креативных специалистов, художников и разработчиков, предоставляющая доступ к множеству моделей машинного обучения, включая инструменты для генерации и анимации изображений. Одним из популярных инструментов на платформе является реализация модели First Order Motion Model (FOMM), позволяющей анимировать изображение по примеру движения на видео.

Данная платформа предлагает возможность гибкой настройки моделей и взаимодействия с ними через визуальный интерфейс (рисунок 1.1).

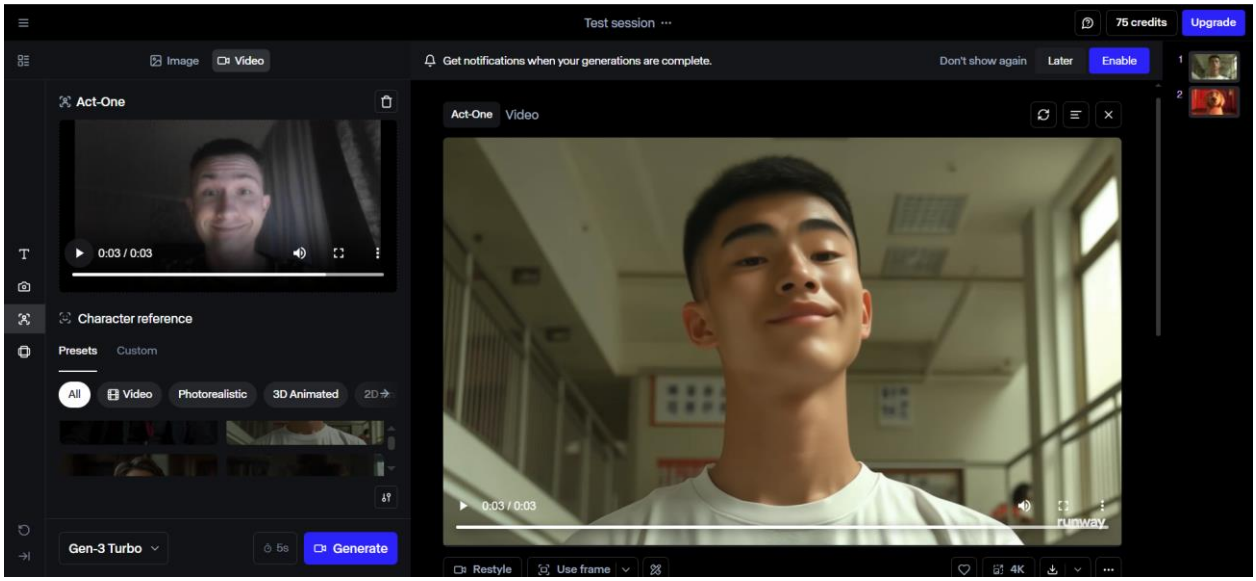


Рисунок 1.1 – Интерфейс взаимодействия RunwayML для анимации изображений [1]

Также предусмотрена поддержка большого количества предобученных моделей. Данная платформа подходит для профессионального и полупрофессионального использования.

Основной недостаток платформы – необходимость платной подписки для доступа ко многим функциям. Также требуются определенный уровень технической подготовки пользователя и высокие системные требования при локальном запуске.

**1.1.2** Deep Nostalgia [2] – функция платформы MyHeritage, предназначенная для оживления старых семейных фотографий. Сервис

автоматически применяет предобученные нейросетевые модели, чтобы создать короткие анимации лиц на изображениях, имитирующие естественные движения головы, глаз и мимику.

Данный сервис обладает следующими достоинствами:

– простой пользовательский интерфейс (рисунок 1.2);

– не требует от пользователя навыков работы с нейросетями или графическими редакторами;

– быстрая обработка на стороне сервера;

– достаточно высокая реалистичность результатов.

Однако сервис позволяет проводить работу только с лицами на фотографиях и имеет ограниченный сценарий использования. Также число бесплатных анимаций ограничено.

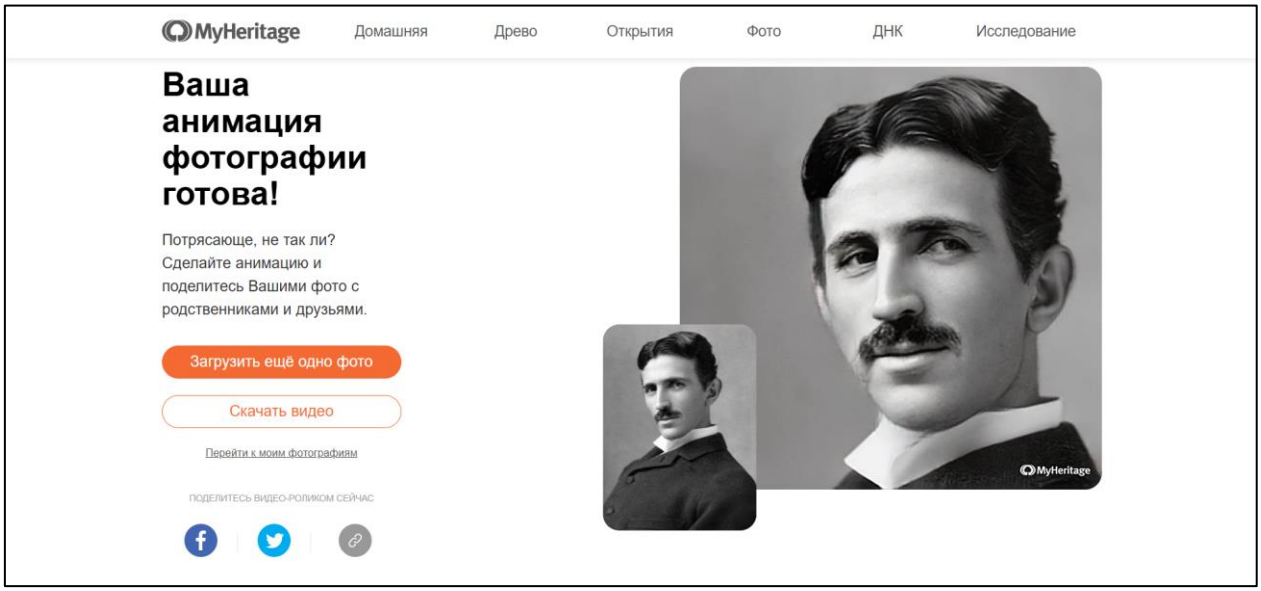


Рисунок 1.2 – Интерфейс взаимодействия Deep Nostalgia (MyHeritage) для анимации изображений [2]

**1.1.3** Fotor – это онлайн-редактор изображений с интеграцией ИИ-функций. Одной из таких функций является AI Face Animator [3], позволяющий оживлять лицо на фотографии, создавая простые анимации, например, улыбки, моргания и другие выражения.

AI Face Animator имеет дружелюбный интерфейс (рисунок 1.3), не требующий специальных знаний, быструю генерацию анимации и возможность комбинировать с другими функциями редактирования изображения.

Сервис имеет следующие недостатки:

– отсутствие пробного бесплатного запуска;

– фокус только на лицах, невозможность полной телесной анимации;

– ограниченный выбор движений;

– результаты зачастую выглядят менее реалистично по сравнению с более продвинутыми решениями.

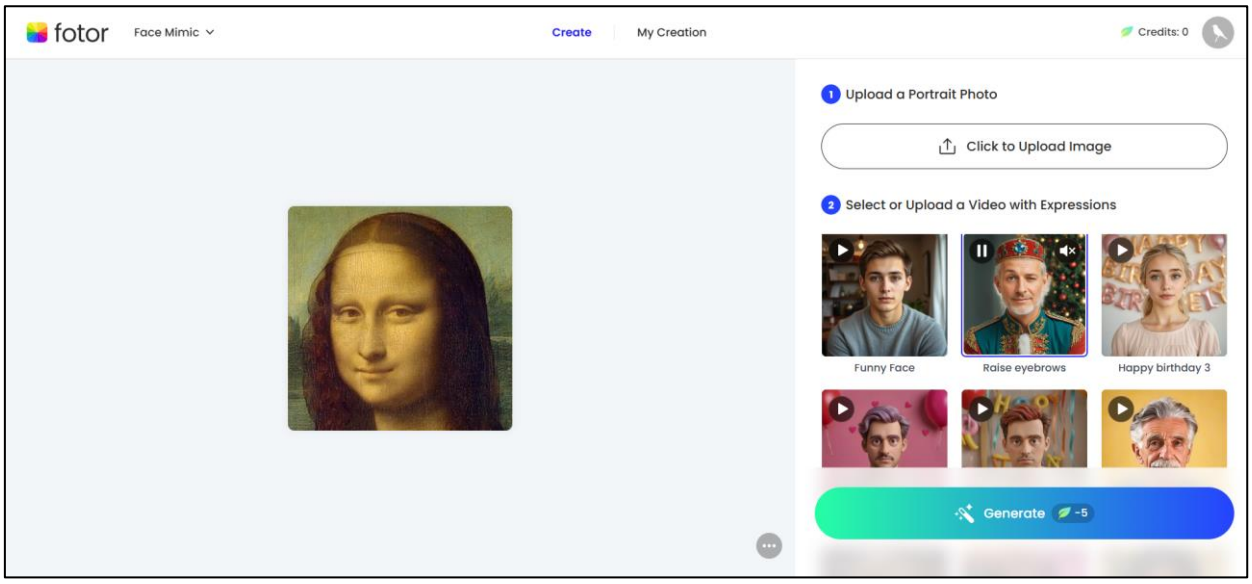


Рисунок 1.3 – Интерфейс взаимодействия AI Face Animator (Fotor) для анимации изображений [3]

**1.1.4** Cutout.pro [4] – платформа, предоставляющая разнообразные инструменты на базе ИИ, включая Photo Animation (рисунок 1.4), позволяющий оживить лица на изображениях. Сервис напоминает Deep Nostalgia, предлагая короткие анимации лиц с базовыми движениями.

У сервиса имеется простой доступ к функциям без необходимости установки. Кроме быстрого выполнения задач, он также поддерживает пакетную обработку.

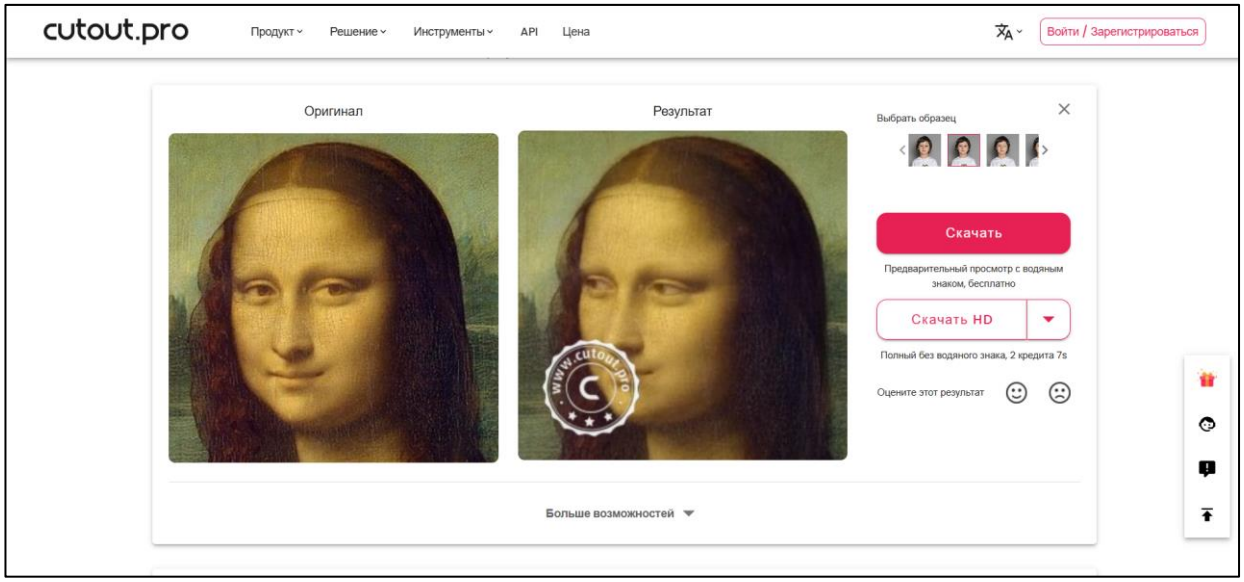


Рисунок 1.4 – Интерфейс взаимодействия Photo Animation (Cutout.pro) для анимации изображений [4]

К сожалению, контроль над результатом ограничен, поэтому сервис обладает низкой гибкостью в сценариях использования. Также имеются

некоторые ограничения на бесплатную версию. Сервис имеет меньшее качество генерации по сравнению с более специализированными инструментами.

### 1.2 Основные подходы для анимации изображений

**1.2.1** X2Face [5] – одна из первых нейронных сетей, предложенных для решения задачи анимации лиц (создания дипфейков) по видеопримерам. Она была представлена в 2018 году исследователями из Оксфорда и компании Visual Geometry Group. Нейросеть основана на концепции переноса движения с управляющего видео на исходное изображение. Для этого используется лицо другого человека для управления позой и выражение целевого лица. Это один из первых проектов, который в принципе смог доказать, что анимация изображений на основе нейронных сетей возможна.

Модель решает задачу переноса движений, разделяя ее на две подзадачи, реализуемые двумя подсетями. Встраиваемая сеть изучает фронтальное изображение лица, а управляющая наделяет это лицо позой и выражением человека, анимирующего изображение. Однако результирующее изображение нередко содержит визуальные артефакты – искажения формы лица, размытость или некорректную передачу мимики, что делает подмену заметной для наблюдателя.

В отличие от более поздних моделей, X2Face не использует явные ключевые точки, а полагается на скрытые представления, изученные в ходе обучения. Это делает модель менее зависимой от точной аннотации данных, но также менее устойчивой к сложным или резким движениям.

Данный проект позволяет управлять изображением лица, используя:

– видео;

– другие изображения;

– аудио или код позы.

Вся модель обучается «self-supervised» методом, то есть без ключевых точек, 3D-моделей или размеченных данных.

**1.2.2** Monkey-Net [6] – открытая нейросеть, представленная в 2019 году группой людей из итальянского университета в сотрудничестве с компанией Snap Inc. Данная модель предлагает универсальный способ анимации изображений любых объектов (не только лиц), основанный на обучаемых ключевых точках и модели движения, получаемой напрямую из видео. Она также работает в «self-supervised» режиме – не требует разметки или заранее заданных ключевых точек.

Сеть состоит из следующих модулей:

– детектор, который обучается без учителя и извлекает ключевые точки объекта;

– сеть прогнозирования для создания тепловых карт и кодирования информации о движении;

– сеть передачи движения, которая синтезирует выходные кадры на

основе тепловых карт движения и входного изображения.

Однако для Monkey-Net сложно моделировать преобразование внешнего вида объектов вблизи ключевых точек, что приводит к плохому качеству генерации, когда масштаб изменения объекта довольно велик.

**1.2.3** Одним из наиболее эффективных подходов к задаче анимации изображения на основе видео является модель First Order Motion Model (FOMM) [7], предложенная в 2020 году. Данный проект является улучшением предыдущего проекта (Monkey-Net), разработанного теми же людьми.

Модель решает задачу переноса движения с видеопоследовательности на статичное изображение, позволяя создавать реалистичную анимацию объекта, изображенного всего на одном кадре. Главной особенностью FOMM является использование изучаемых ключевых точек и оценка локальных аффинных преобразований, что обеспечивает гибкость и универсальность в работе с различными типами объектов – от лиц до произвольных форм.

Принцип работы модели основан на обучении с использованием видеопар, состоящих из двух кадров из одной видеопоследовательности: исходного (опорного) и движущегося. Модель изучает, как внешний вид объекта меняется от одного кадра к другому, и представляет движение как комбинацию смещений изученных ключевых точек и локальных преобразований в их окрестности. Эти преобразования позволяют учитывать сложные деформации, включая вращения и изменения перспективы.

Архитектура FOMM делится на два основных модуля:

– оценка движения;

– генерация изображения.

Модуль оценки движения состоит из сети, которая извлекает ключевые точки и определяет локальные аффинные преобразования в их окрестностях. Эти ключевые точки и соответствующие преобразования обучаются в «self-supervised» режиме, то есть без необходимости в заранее размеченных данных. В отличие от моделей, опирающихся только на смещения точек, FOMM использует также локальные преобразования, что значительно расширяет выразительные возможности модели и позволяет точнее описывать движение объектов.

На следующем этапе плотная сеть движения объединяет локальные аффинные преобразования, создавая полное поле движения между исходным и целевым изображением. Кроме того, она формирует маску окклюзии, указывающую, какие части изображения могут быть получены деформацией исходного кадра, а какие нуждаются в восстановлении, например, из-за поворота объекта или его частичного перекрытия.

Модуль генерации изображения использует эту информацию для синтеза нового изображения, в котором объект из исходного кадра отображается в положении и с выражением, соответствующими движущемуся кадру. Генератор деформирует изображение с учетом рассчитанного движения и достраивает области, не видимые в исходном изображении. Это позволяет добиться высокой реалистичности при сохранении визуальной

согласованности и идентичности объекта.

Структура FOMM показана на рисунке 1.5.

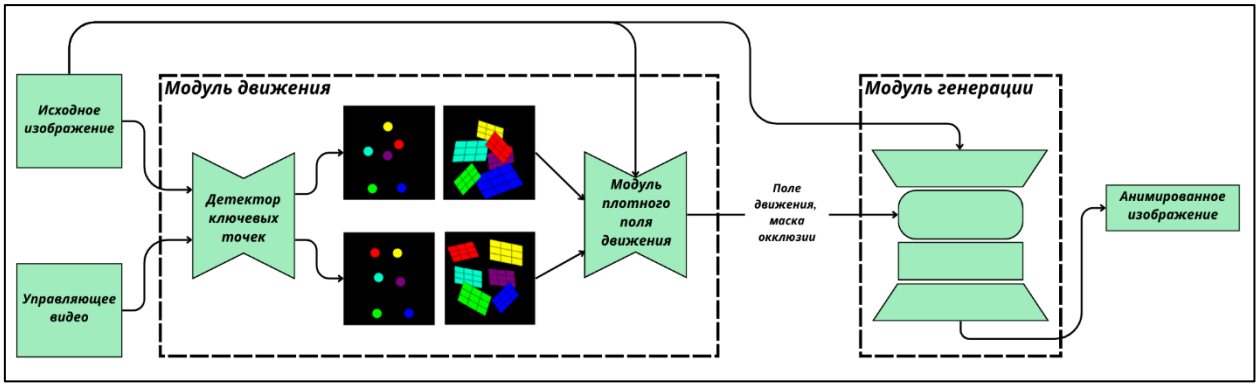


Рисунок 1.5 – Структура FOMM

В таблице 1.1 представлен сравнительный анализ моделей X2Face, Monkey-Net и FOMM.

Таблица 1.1 – Сравнение моделей X2Face, Monkey-Net и FOMM

| Признак | X2Face | Monkey-Net | FOMM |
|---|---|---|---|
| Архитектура | Encoder + Driving Network + Decoder | Keypoint Detector + Dense Motion + Generator | Keypoint Detector + Jacobian Estimator + Motion Module + Generator |
| Модель движения | Код позы/выражения лица (вектор) | Поток с ключевыми точкам | Поток + локальные аффинные преобразования |
| Ключевые точки, маска окклюзии | ✗ | ☑ | ☑ + Якобианы |
| Скорость генерации (FPS) | ~25–30 | ~20–25 | ~18–22 |
| Достоинства | Простая архитектура, мульти-модальность | Хорошая генерации с ключевыми точками | Лучшее качество, устойчивость, точность |
| Недостатки | Потеря идентичности, искажения на поворотах | Ограниченность ключевых точек, артефакты | Тяжелая модель, требуются качественные входные данные |

Таким образом, в контексте задачи анимации по одному изображению архитектура FOMM на сегодняшний день считается одной из наиболее сбалансированных по качеству, универсальности и доступности. Однако, в зависимости от специфики задачи, могут применяться и другие модели, предлагающие компромисс между производительностью и качеством результата. Анализ существующих архитектур позволяет сделать обоснованный выбор модели, наиболее подходящей для реализации в рамках разрабатываемого программного средства.

## 1.3 Выбор основного технологического стека

**1.3.1** Python [8] – один из самых популярных языков программирования [9]. Он был создан Гвидо ван Россумом и выпущен в 1991 году. Имеет эффективные высокоуровневые структуры данных и простой, но действенный подход к объектно-ориентированному программированию. Элегантный синтаксис и динамическая типизация Python, вместе с его интерпретируемой природой, делают его идеальным языком для написания скриптов и быстрой разработки приложений во многих областях на большинстве платформ.

Этот язык считается довольно полезным и удобным для работы с искусственным интеллектом (ИИ), и по статистике это третий по популярности язык после JavaScript и HTML/CSS. Индекс TIOBE, который учитывает популярность поисковых запросов в рейтинге, в настоящее время ставит Python на первое место [10]. Сегодня на Python работают множество известных организаций, например, Google, Facebook, Instagram, Spotify, Netflix, Quora.

Его мощь проявляется в способности автоматизировать задачи и оптимизировать рабочие процессы. Удобный синтаксис Python делает его более привлекательным для программистов по сравнению с другими языками, что в конечном итоге способствует повышению эффективности разработчиков. Также сегодня на GitHub есть более миллиона репозиториев. В результате все эти факторы делают Python одним из наиболее предпочтительных языков программирования.

Кроме того, Python используется для веб-разработки (серверной), разработки ПО, математики, системных скриптов.

**1.3.2** Для реализации нейросетевой архитектуры, связанной с генерацией анимации на основе изображения и видео, критически важен выбор подходящего фреймворка. Он должен обеспечивать гибкость при работе с архитектурой модели, поддержку вычислений на GPU, удобство отладки, широкое сообщество и доступ к готовым решениям.

PyTorch [11] и TensorFlow [12] – ведущие фреймворки (рисунок 1.6) глубокого обучения, широко используемые специалистами по данным, инженерами машинного обучения (ML) и исследователями за их простоту использования, масштабируемость и открытый исходный код. Выбор между

PyTorch и TensorFlow требует тщательного рассмотрения их функций и пригодности для конкретного проекта или потребностей в обучении модели.

PyTorch – это библиотека глубокого обучения с открытым исходным кодом от Facebook AI Research, известная своим интуитивно понятным API, динамическим вычислительным графом и ускорением GPU. Она преуспевает в области компьютерного зрения, обработки естественного языка и обучения с подкреплением, предлагая такие функции, как TorchVision для предварительно обученных моделей и бесшовную интеграцию Python. Ее активное сообщество обеспечивает надежную поддержку и ресурсы.

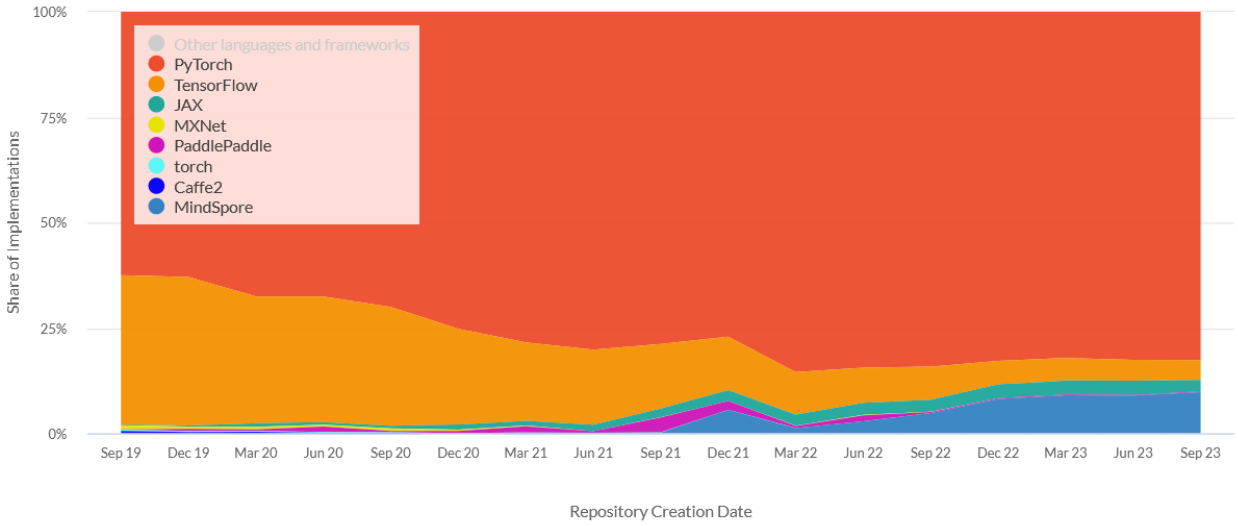


Рисунок 1.6 – Тенденции внедрения ML фреймворков (2023 – 2024) [13]

Кроме того, PyTorch предлагает динамические вычислительные графы, позволяющие вносить изменения на лету во время обучения. Это делает его идеальным для проектов, требующих адаптивных архитектур.

TensorFlow – это фреймворк, разработанный компанией Google, с открытым исходным кодом для создания и развертывания моделей машинного обучения. Он поддерживает вычислительные графы, аппаратное ускорение с помощью GPU/TPU и ряд архитектур, таких как CNN и transformers. Благодаря таким инструментам, как Keras, и сильному сообществу она упрощает экспериментирование и развертывание производства.

В отличие от этого, TensorFlow использует статические вычислительные графы, которые фиксируются после определения.

Данный фреймворк оптимизирован для обработки больших наборов данных и отлично подходит для распределенных вычислений, что делает его более подходящим для крупномасштабных проектов.

С учетом всех перечисленных факторов, наиболее подходящим фреймворком для разработки программного средства генерации анимации на основе изображений является PyTorch. Он сочетает в себе гибкость, высокую производительность и простоту разработки, что делает его особенно удобным для реализации и тестирования моделей, требующих сложных архитектурных

решений и интенсивных вычислений. PyTorch активно развивается, используется как в научных публикациях, так и в прикладных разработках, что делает его оптимальным выбором для данного дипломного проекта.

**1.3.3** FastAPI [14] – это современный, быстрый и высокопроизводительный веб-фреймворк для создания API используя Python.

Фреймворк полностью поддерживает асинхронное программирование, позволяя писать асинхронные обработчики маршрутов и использовать преимущества синтаксиса `async/await` в Python для неблокирующих операций ввода-вывода.

Имеется автоматическая генерация интерактивной и удобной для пользователя документации по API. Используются стандарты OpenAPI и JSON Schema для предоставления исчерпывающей документации для вашего API, включая проверку входных данных, ожидаемые ответы и многое другое. Существует несколько вариантов документирования, два из которых включены по умолчанию: Swagger UI и ReDoc.

Благодаря интеграции FastAPI с Pydantic можно определять модели данных с помощью подсказок типов Python, обеспечивая автоматическую проверку данных и сериализацию. Эта функция помогает выявлять ошибки на ранних стадиях процесса разработки и повышает читаемость кода.

FastAPI поддерживает внедрение зависимостей, позволяя эффективно управлять зависимостями и организовывать их. Эта функция особенно полезна при работе с подключениями к базе данных, аутентификацией и другими общими ресурсами.

Синтаксис FastAPI понятен, легок для чтения и очень похож на стандартное определение функций Python, что делает его доступным как для начинающих, так и для опытных разработчиков.

У FastAPI существует множество альтернатив:

1 Flask – легкий и широко используемый веб-фреймворк в экосистеме Python. Он прост в использовании и с ним легко начать работу, но ему не хватает встроенной поддержки асинхронности и автоматической проверки на основе подсказок типов.

2 Django – полнофункциональный веб-фреймворк, который следует философии «батарейки включены». Он предоставляет множество готовых функциональных возможностей, включая ORM, интерфейс администратора и многое другое, но для небольших и простых API это может оказаться излишним. В нем зашит паттерн MVC (Model-View-Controller). По сути, это модульный монолит, который не такой гибкий, как FastAPI.

В целом это два самые популярные фреймворки, которые носили по-настоящему массовый характер.

Описанные ниже фреймворки были и есть не очень популярны, не очень удобны, либо не очень производительны:

1 Tornado – асинхронный веб-фреймворк, который может обрабатывать большое количество одновременных подключений. Он часто используется в сценариях, требующих высокого уровня параллелизма, но может иметь более

крутую кривую обучения по сравнению с FastAPI (тяжел, местами не очевиден, не очень удобен).

2 Bottle – минималистичный веб-фреймворк, разработанный для маломасштабных приложений. Он легкий и простой в использовании, но ему не хватает скорости, масштабируемости и оптимизации производительности, присутствующие в FastAPI.

Таким образом, можно перечислить основные достоинства FastAPI:

– скорость;

– встроенная поддержка проверки типов;

– возможность реализации высоконагруженных API за счет асинхронности.

### 1.4 База данных

Для хранения и управления структурированными данными в рамках данного проекта была выбрана система управления базами данных PostgreSQL.

PostgreSQL [15] – это объектно-реляционная система управления базами данных с открытым исходным кодом, известная своей надежностью, соответствием стандартам SQL, гибкостью и расширяемостью.

Основные преимущества PostgreSQL:

– строгая поддержка ACID-транзакций;

– поддержка сложных запросов и индексации;

– расширяемость – можно подключать пользовательские функции, типы данных и модули;

– надежность и популярность – проект развивается более 25 лет и применяется в высоконагруженных и критичных системах;

– хорошая интеграция с Python.

В таблице 1.2 представлены основные недостатки других баз данных в контексте данного проекта.

Таблица 1.2 – Основные недостатки других СУБД в контексте данного проекта

| СУБД | Тип | Недостатки |
|---|---|---|
| MySQL | Реляционная | Ограниченная поддержка JSON и сложных SQL-функций, слабее транзакции |
| MongoDB | Документо-ориентированная | Нет строгой схемы, сложно валидация и связи между сущностями |
| SQLite | Встраиваемая | Не подходит для многопользовательских веб–приложений |
| Redis | In-memory, key-value | Не предназначен для долговременного хранения и сложных запросов |
| Firebase | Облачная NoSQL | Привязка к экосистеме Google, сложные миграции |

В контексте данного проекта PostgreSQL используется для хранения

информации о пользователях и их подписках на сервис анимации изображений.

MongoDB часто выбирают за гибкость, но в ML-приложениях, где важна воспроизводимость и верифицируемость, слабая типизация и отсутствие схем скорее минусы. MySQL проигрывает PostgreSQL по гибкости запросов, полноте стандартов SQL и работе с JSON. Redis и Firebase вообще не решают задачи полноценного хранения структурированных данных.

В итоге, PostgreSQL обеспечивает надежное хранение данных, гибкость и расширяемость, при этом оставаясь опенсорсным и легко разворачиваемым как локально, так и в Docker-контейнерах. В условиях проекта, сочетающего веб-интерфейс, модель ИИ и серверную логику, это делает PostgreSQL безальтернативным вариантом.

### 1.5 Среда разработки и способы представления результатов

**1.5.1** Веб-приложения являются удобным способом представления результатов, позволяющим пользователю получать доступ к системе через любой браузер без необходимости установки дополнительного ПО. Веб-приложения могут быть более сложными и функциональными, чем простые интерфейсы, и предлагают больше возможностей для визуализации и настройки.

Существует множество вариантов в Python для развертывания модели. Так на сегодняшний день есть несколько популярных веб-фреймворков – Django, FastAPI, Flask. Но проблема с их использованием заключается в том, что для красивого визуального и удобного взаимодействия пользователя с моделью должна быть создана некоторая Frontend-часть и, очевидно, иметься некоторые знания об HTML, CSS и JavaScript. Учитывая эти предварительные условия, был создан Streamlit.

Streamlit [16] – это библиотека Python, предназначенная для создания веб-приложений без необходимости разработки frontend-части. Благодаря этому инструменту можно в короткие сроки реализовать удобный пользовательский интерфейс, позволяющий загружать изображения, запускать генерацию анимации и просматривать результаты прямо в браузере. Простота кода и встроенная визуализация делают Streamlit оптимальным выбором для представления результатов работы модели конечному пользователю. Дополнительным преимуществом является возможность бесплатного развертывания приложения через облачные сервисы компании Streamlit.

**1.5.2** Для более мощных решений, а также с целью увеличения доступности системы можно интегрировать модель с облачными сервисами. Эти платформы позволяют работать с большими объемами данных и мощными вычислительными ресурсами, что важно для эффективной генерации видео.

Google Colab [17] – это платформа для работы с Jupyter-ноутбуками,

предоставляющая бесплатные вычислительные ресурсы, включая графические процессоры (GPU). Google Colab является отличным вариантом для демонстрации и тестирования моделей нейронных сетей, так как позволяет запускать модели в облаке, использовать GPU для ускорения процесса генерации и легко делиться результатами с другими пользователями.

Google Colab построен на основе кода Project Jupyter и размещает блокноты Jupyter без необходимости установки локального программного обеспечения. Но в то время, как блокноты Jupyter поддерживают несколько языков, включая Python, Julia и R, Colab в настоящее время поддерживает только Python.

Блокноты Colab хранятся в учетной записи Google Drive и могут совместно использоваться с другими пользователями, как и другие файлы Google Drive. Блокноты также включают функцию автоматического сохранения, но они не поддерживают одновременное редактирование, поэтому совместная работа должна быть последовательной, а не параллельной.

Colab бесплатен, но имеет ограничения. Некоторые типы кода запрещены, например, обслуживание мультимедиа и майнинг криптовалют. Доступные ресурсы также ограничены и зависят от спроса, хотя Google Colab предлагает профессиональную версию с более надежными ресурсами. Существуют и другие облачные сервисы на основе Jupyter Notebook, включая Azure Notebooks от Microsoft и SageMaker Notebooks от Amazon.

**1.5.3** PyCharm [18] используется как основная среда для локальной разработки и интеграционного тестирования проекта. Эта IDE предоставляет широкие возможности для написания, отладки и управления кодом, а также облегчает работу с виртуальными окружениями, git-репозиториями и зависимостями.

PyCharm предлагает встроенную поддержку Python, баз данных, Jupyter, Git, Conda, PyTorch, TensorFlow, Hugging Face, Django, Flask, FastAPI и т. д. Благодаря бесшовной интеграции контекстно-зависимого помощника AI можно быстро приступить к работе и добиться более эффективного прогресса. PyCharm предоставляет вам все необходимое для всех видов проектов: от веб-разработки и конвейеров данных до прототипирования моделей машинного обучения и анализа данных.

С помощью PyCharm осуществляется структурирование проекта, разработка вспомогательных скриптов и подготовка всего программного решения к развертыванию.

### 1.6 Набор данных для обучения

Качественный набор данных играет ключевую роль в обучении моделей, особенно в задачах генерации видео и анимации изображений. В рамках данного проекта, для обучения модели синтеза движения лица на основе одного изображения, был выбран датасет VoxCeleb [19].

VoxCeleb – это крупный открытый набор видеоданных, содержащий записи с участием более 7000 известных персон, собранных из 22496 видео YouTube. Он включает видеофрагменты с лицами в различных ракурсах, условиях освещенности, выражениях и движениях, а также с различным качеством видео и шумами, что приближает данные к условиям реального мира. VoxCeleb отличается высоким качеством и готовностью к использованию – лица уже автоматически детектированы и выровнены.

Данный датасет также широко признан в научном сообществе и применяется в большинстве современных исследований по генерации движений лица.

### 1.7 Развертывание проекта

Развертывание проекта является важным этапом, обеспечивающим переносимость, воспроизводимость и удобство эксплуатации программного решения. В данном проекте для упаковки и развертывания всех компонентов – модели, интерфейса и зависимостей – используется Docker в связке с Docker Compose.

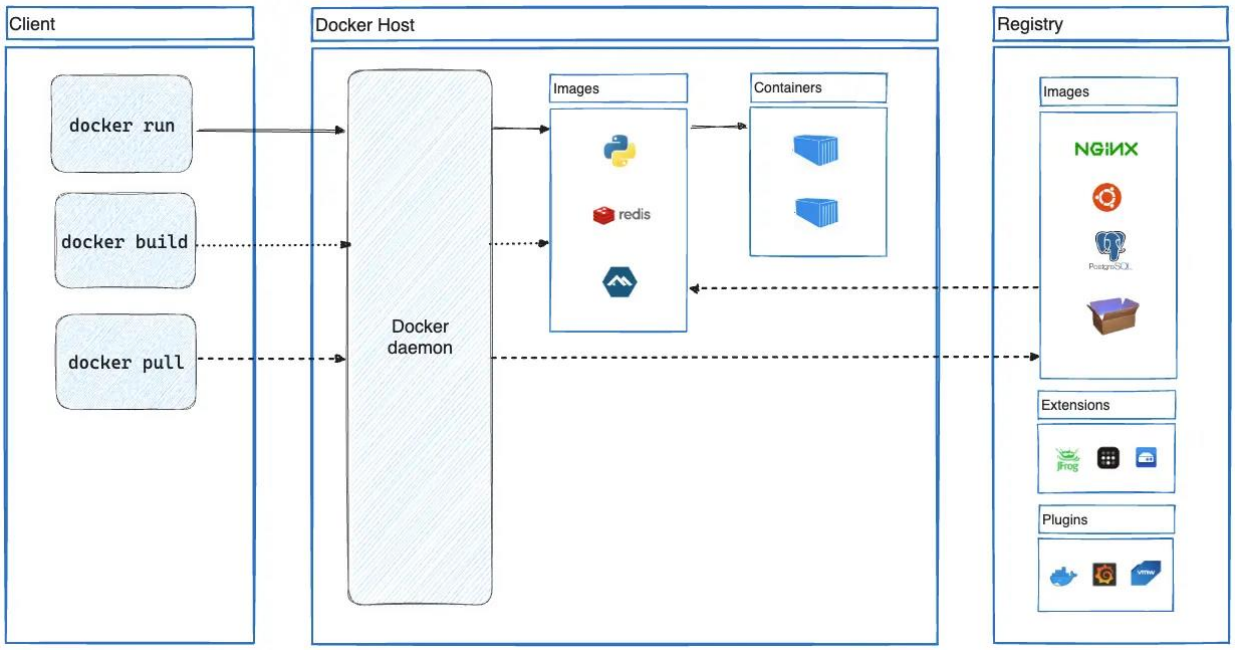


Рисунок 1.7 – Архитектура Docker [20]

Docker [20] – это контейнеризационная платформа, позволяющая создавать изолированные и воспроизводимые среды, в которых можно запускать приложения вне зависимости от операционной системы и конфигурации основной системы пользователя. Контейнеры Docker легче и быстрее виртуальных машин, поскольку используют ядро хост-системы, при этом обеспечивают ту же степень изоляции. На рисунке 1.7 представлена архитектура Docker.

Все компоненты проекта кроме веб-интерфейса упакованы в отдельные

контейнеры и управляются с помощью Docker Compose, что обеспечивает гибкое масштабирование и удобное локальное тестирование. Для веб-интерфейса также предусмотрена возможность создания Docker-контейнера, но он не входит в единый Docker Compose с другими частями проекта.

Также стоит упомянуть Docker AI – новую инициативу от Docker, которая предлагает интеграцию ИИ-моделей и возможностей генеративного ИИ в рабочие процессы разработки. Хоть технология пока находится в стадии активного развития, она уже предлагает такие функции, как генерация Dockerfile по описанию и автоконфигурация окружений. Это упрощает создание контейнеров даже тем, кто не имеет большого опыта работы с инфраструктурой.

На сегодняшний день Docker не имеет полноценных альтернатив, сопоставимых по уровню зрелости, популярности и экосистеме. Существуют другие контейнерные технологии, такие как Podman, LXC или Singularity, но они, как правило, либо требуют более сложной настройки, либо имеют ограниченные возможности, либо ориентированы на узкоспециализированные задачи (например, Singularity – для научных вычислений в HPC-среде).

Docker занял лидирующую позицию в индустрии, став фактически стандартом де-факто для разработки, тестирования и развертывания приложений. Он поддерживается всеми основными облачными платформами (AWS, Azure, GCP), CI/CD системами и хостингами, такими как GitHub Actions, GitLab CI, Heroku и другими. Большое сообщество, развитая документация и доступ к огромному количеству готовых образов через Docker Hub делают Docker не просто инструментом, а неотъемлемой частью современного цикла разработки.

Именно поэтому было принято решение использовать Docker и Docker Compose для развертывания всех компонентов проекта. Это обеспечивает надежность, переносимость и простоту использования как при локальной разработке, так и при последующей интеграции в облачную инфраструктуру.

## 2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Процесс проектирования системы для анимации изображений включает несколько ключевых компонентов, каждый из которых выполняет свою роль в обеспечении качественного результата. Данный проект включает следующие компоненты: веб-интерфейс, backend-сервер, модель нейронной сети, база данных. На рисунке 2.1 показан схема взаимодействия компонентов данного проекта.

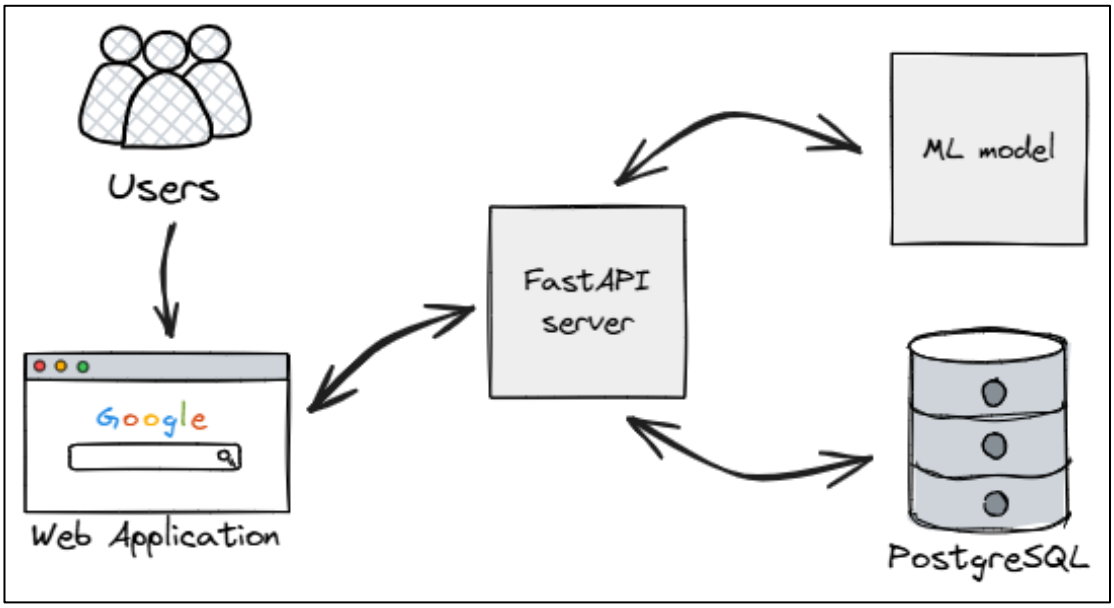


Рисунок 2.1 – Схема взаимодействия компонентов проекта

Было выделено несколько блоков, каждый из которых отвечает за определенные функции в этой системе:

– блок API нейронной сети;
– блок модели нейронной сети;
– блок данных нейронной сети;
– блок сервера для анимации изображений;
– блок сервера для аутентификации;
– блок сервера для подписок;
– блок сервера для оплаты;
– блок пользовательского интерфейса.

Это структурное разделение позволяет эффективно управлять процессом генерации анимации, а также улучшать каждый компонент системы по мере необходимости. Взаимосвязь между основными блоками проекта отражена на структурной схеме ГУИР.400201.042 С1.

### 2.1 Блок API нейронной сети

Блок API нейронной сети является составной частью одного из четырех основных компонентов системы – модели нейронной сети. Он выполняет

функцию промежуточного слоя между пользовательскими запросами и самой моделью генерации анимации.

Блок инкапсулирует вызовы модели и предоставляет внешний REST API, через которое другие части системы могут передавать входные изображения и параметры генерации, а затем получать результат – сгенерированное видео. Это упрощает модификацию архитектуры модели, обновление ее версий, проведение тестирования или замену на альтернативную архитектуру без необходимости менять остальную часть приложения.

К основным функциям блока можно отнести:

– прием, валидация и сериализация входных данных;

– вызов модели генерации;

– постобработка и сохранение результата;

– возврат результата другим компонентам (серверу);

– логирование запросов и ошибок.

API реализован с использованием легкого и высокопроизводительного сервера, построенного на FastAPI. Это позволяет добиться высокой скорости обработки запросов и автоматически генерировать удобную и наглядную документацию в формате Swagger, доступную по запуску. Также будет предоставлен endpoint для запуска генерации анимации изображения.

Размещение этого блока в виде изолированного микросервиса (в Docker-контейнере) позволяет масштабировать его независимо от других частей системы, например, запускать несколько экземпляров модели параллельно или размещать их на отдельных графических процессорах. Такой подход упрощает управление инфраструктурой, улучшает воспроизводимость и ускоряет отладку. Кроме того, благодаря четкому разделению, блок может повторно использоваться в других проектах или расширен для поддержки других моделей генерации, сохраняя тот же интерфейс взаимодействия.

Для предотвращения несанкционированного доступа API защищен проверкой специальных данных, передаваемых вместе с запросом. Только при успешной авторизации будет выполнен запуск модели.

### 2.2 Блок модели нейронной сети

Блок модели нейронной сети является центральной частью одного из четырех компонентов системы – модели нейронной сети. Этот блок непосредственно реализует архитектуру генерации анимации изображений и отвечает за всю основную вычислительную логику, связанную с обработкой входных данных и синтезом выходного видео.

Блок включает в себя реализованную модель (в данном проекте – архитектура FOMM), а также все вспомогательные модули, необходимые для ее функционирования. Он работает с тензорными представлениями изображений и видео, выполняя предварительную обработку, извлечение ключевых точек, аппроксимацию движений, генерацию плотного поля движения и, в конечном итоге, синтез новых кадров с анимацией.

К основным функциям блока относятся:

– загрузка предварительно обученной модели;

– генерация последовательности кадров на основе исходного изображения и видео-референса;

– выполнение всех этапов архитектурного конвейера (оценка движения, генерация изображения, построение маски окклюзии);

– обработка промежуточных и выходных данных;

– оптимизация производительности (использование GPU, если он доступен).

Для реализации модели и ее компонентов используется фреймворк PyTorch, обеспечивающий гибкость в построении архитектур, высокую производительность вычислений и широкую поддержку инструментов для обучения и вывода.

Блок модели реализуется как отдельный Python-модуль и может быть легко загружен как часть сервиса. Его изоляция позволяет независимо работать над улучшением качества анимации, обновлять архитектуру модели, переобучать ее на новых данных или оптимизировать для конкретных задач.

Важно отметить, что данный блок сам по себе не взаимодействует с внешними пользователями или системами: он вызывается из блока сервисов нейронной сети, что обеспечивает четкое разделение логики и упрощает масштабирование и тестирование.

### 2.3 Блок данных нейронной сети

Блок данных нейронной сети также, как и предыдущие два блока, входит в состав модели нейронной сети, и обеспечивает подготовку, трансформацию и загрузку данных, необходимых для корректной работы модели. Этот блок служит фундаментом при обучении, тестировании и использовании модели, обеспечивая ее чистыми, структурированными и согласованными входными данными.

В рамках проекта данный блок реализует следующий функционал:

– загрузка и структурирование датасета;

– разделение данных;

– аугментация данных;

– создание пользовательских классов датасетов и загрузчиков данных.

Таким образом, блок данных нейронной сети является неотъемлемой частью архитектуры модели. Он изолирует всю логику, связанную с подготовкой и обработкой датасета, обеспечивая надежную и воспроизводимую подачу информации в модель, что критически важно для стабильности, качества и скорости ее работы.

### 2.4 Блок сервера для анимации изображений

Блок сервера для анимации изображений представляет собой звено между пользовательским интерфейсом и вычислительными компонентами

системы. Его основное назначение – организация обработки пользовательских запросов, связанных с генерацией анимации, а также маршрутизация данных внутри системы. Этот блок входит в состав архитектурного компонента, отвечающего за серверную логику, и обеспечивает согласованное взаимодействие всех модулей, задействованных в процессе анимации.

Блок реализуется как часть веб-сервера с использованием современного фреймворка, обеспечивающего высокую производительность и простоту масштабирования. В данном проекте для этого используется FastAPI, что, как и в случае с сервером для модели нейронной сети, позволяет быстро разрабатывать, документировать и отлаживать REST-интерфейсы.

После получения запроса от клиента сервер принимает изображение, видео-референс и другие параметры, валидирует входные данные, обрабатывает их и направляет к вычислительным модулям. Внутри реализована логика взаимодействия с API модели нейронной сети, к которому осуществляется обращение через защищенный эндпоинт (англ. endpoint). Результатом взаимодействия является сгенерированное видео, которое возвращается обратно в интерфейс пользователя.

Особое внимание уделено обработке ошибок и устойчивости: при возникновении сбоев сервер способен выявлять проблему и возвращать клиенту понятный ответ, не прерывая работу всей системы. Полученные результаты форматируются и возвращаются в интерфейс пользователя. Безопасность взаимодействия обеспечивается современными методами защиты.

### 2.5 Блок сервера для аутентификации

Блок сервера для аутентификации представляет собой элемент системы, отвечающий за обеспечение безопасности и контроль доступа пользователей к функциональности проекта. Он входит в состав серверной инфраструктуры и играет ключевую роль в управлении пользовательскими сессиями и защите персональных данных.

Основной задачей данного блока является проверка подлинности пользователей, использующих веб-интерфейс. Когда пользователь впервые взаимодействует с системой, он проходит процедуру регистрации или авторизации. В этот момент блок аутентификации принимает на себя обработку учетных данных – логина и пароля – и сравнивает их с данными, хранящимися в базе. При успешной проверке пользователю предоставляется доступ к необходимому функционалу. В случае неудачной попытки входа блок возвращает четкие и безопасные сообщения об ошибке, не раскрывая конфиденциальную информацию.

После регистрации каждого нового пользователя в базе данных сохраняется соответствующая запись. Пароли пользователей хранятся только в зашифрованном виде, что соответствует требованиям информационной безопасности.

Для управления сессиями и контроля доступа в системе используется

механизм JWT (англ. JSON Web Token). При успешной авторизации пользователь получает JWT-токен, который в дальнейшем прикрепляется к каждому запросу, требующему проверки подлинности. Сервер проверяет подпись токена и извлекает из него информацию о пользователе, тем самым обеспечивая безопасный и масштабируемый способ аутентификации.

Реализация с применением JWT позволяет эффективно разграничивать доступ к функциональности, упрощает интеграцию с другими сервисами и обеспечивает хорошую масштабируемость системы за счет отсутствия необходимости хранить сессионные данные на сервере.

### 2.6 Блок сервера для подписок

Блок сервера для подписок реализует логику управления пользовательскими тарифами и уровнями доступа к различным возможностям системы. Он входит в состав серверной части проекта и отвечает за корректную обработку и учет пользовательских прав, зависящих от наличия активной подписки. Этот блок тесно взаимодействует с базой данных, обновляя и считывая информацию о текущем статусе пользователя и оставшихся возможностях использования сервиса.

В рамках работы блока сервер подписок обрабатывает запросы на пополнение попыток использования сервиса анимации изображений, предоставляет соответствующий уровень доступа к вычислительным ресурсам и управляет данными, связанными с пользовательскими привилегиями. Все это позволяет гибко масштабировать функциональность в зависимости от потребностей пользователей и политики монетизации системы.

Ключевые функции блока:

– взаимодействие с модулем оплаты для активации/деактивации подписки;

– интеграция с системой авторизации для ограничения доступа;

– предоставление API-запросов для пользовательского интерфейса.

При реализации блока используется таблица базы данных, в которой хранятся сведения о подписке каждого пользователя. Каждая запись отражает количество оставшихся запросов, доступных конкретному пользователю. Это значение уменьшается при каждом использовании соответствующей функции.

В рамках API будут доступны отдельные точки доступа для получения информации о текущей подписке пользователя и для просмотра перечня всех доступных уровней подписки с описанием условий и стоимости.

Блок подписок, хотя и является технически вспомогательным, имеет стратегическое значение. Он не только формирует основу бизнес-логики системы, но и обеспечивает устойчивое развитие проекта за счет разделения на платные и бесплатные уровни доступа. Благодаря изолированному построению, данный блок может быть масштабируемым и адаптированным под различные модели монетизации, будь то фиксированная оплата, система уровней или подписка по использованию.

Таким образом, блок сервера для подписок служит связующим элементом между пользовательским опытом, финансовой составляющей проекта и технической реализацией ограничения доступа, обеспечивая гибкость, управляемость и прозрачность внутри всей системы.

### 2.7 Блок сервера для оплаты

Блок сервера для оплаты отвечает за обработку всех аспектов, связанных с транзакциями в системе. Он является составной частью серверной архитектуры и тесно взаимодействует с блоками подписок и аутентификации. Основная задача данного блока – обеспечить надежное, безопасное и корректное выполнение процессов оплаты, а также хранение информации о платежах и статусах транзакций.

Этот компонент играет ключевую роль в реализации платных функций проекта, так как он обеспечивает интеграцию с внешними платежными шлюзами, проверку успешности операций, реагирование на ошибки и автоматическое обновление состояния подписки пользователя на основе полученных данных. Кроме того, он формирует финансовую прозрачность и доверие пользователей за счет корректного учета и обработки всех платежей.

Для реализации платежной логики будет использоваться сервис Stripe [21] – современная платежная платформа, поддерживающая онлайн-оплаты по всему миру. Stripe предоставляет удобные и безопасные инструменты для работы с банковскими картами, управления подписками, выставления счетов. Интеграция со Stripe позволит обеспечить гибкую и масштабируемую платежную инфраструктуру, соответствующую современным требованиям к защите данных и удобству пользователей.

Ключевые задачи, решаемые блоком сервера для оплаты:

– прием и обработка запросов на оплату от клиентов;

– генерация платежных сессий;

– обработка обратных уведомлений от платежной системы о статусе транзакций.

Блок оплаты спроектирован таким образом, чтобы быть независимым от конкретного платежного провайдера. Хотя в текущей реализации используется Stripe, структура системы позволяет при необходимости подключать и другие сервисы без значительных изменений в остальной архитектуре.

Кроме того, обеспечивается высокая устойчивость и безопасность: все чувствительные данные обрабатываются исключительно на стороне платежного провайдера, а сервер системы взаимодействует только с проверенными токенами и подтверждениями транзакций. Это помогает соответствовать стандартам защиты данных и предотвращать мошенничество.

Функционально блок сервера оплаты выполняет роль надежного посредника между пользователем и системой, гарантируя, что финансовые транзакции проходят корректно, а доступ к функциональности обновляется своевременно. Его стабильная и защищенная работа критична как для

коммерческого успеха проекта, так и для построения долгосрочных отношений с пользователями.

### 2.8 Блок пользовательского интерфейса

Блок пользовательского интерфейса отвечает за взаимодействие пользователя с системой и представляет собой внешний уровень архитектуры, через который осуществляется доступ к функциональности проекта. Он является составной частью одного из четырех ключевых компонентов – веб-интерфейса – и предназначен для обеспечения удобства, интуитивной понятности и эффективности работы с системой.

Цель данного блока – предоставить пользователю возможность загружать изображения, запускать процесс генерации анимации, просматривать результаты, а также управлять своей учетной записью, подпиской и платежами. Именно от качества реализации пользовательского интерфейса во многом зависит восприятие всей системы и уровень вовлеченности конечного пользователя.

В реализации пользовательского интерфейса используется Streamlit – удобный и быстрый фреймворк для создания веб-приложений на Python. Он обеспечивает простую интеграцию с серверной частью и позволяет оперативно разрабатывать интерактивные элементы интерфейса без необходимости глубокой frontend-разработки. Для расширения визуальной составляющей дополнительно применяются HTML и CSS, что дает больше гибкости в оформлении и стилизации компонентов.

Ключевые функции, реализуемые в блоке пользовательского интерфейса:

– загрузка исходных изображений и запуск анимации изображений;

– регистрация, вход в систему;

– доступ к информации о подписке и возможности ее обновления;

– обратная связь и отображение уведомлений;

– обработка ошибок и исключительных.

Таким образом, блок пользовательского интерфейса – это связующее звено между пользователем и внутренними компонентами проекта, обеспечивающее простое и удобное управление всей системой.

# 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Функциональное проектирование направлено на формирование внутренней структуры программной системы, чтобы она надежно и эффективно выполняла все поставленные перед ней задачи. В данном разделе приводится описание основных компонентов проекта, включая их внутренние модули, классы и методы, а также способы их взаимодействия. Описываются потоки данных между элементами системы, а также принципы, лежащие в основе архитектурных решений.

Описание проекта можно разделить на три части:

– модель нейронной сети, реализующая всю вычислительную логику генерации анимации изображений (рисунок 3.1);

– серверная часть, отвечающая за маршрутизацию данных, авторизацию пользователей, работу с подписками, оплатой и взаимодействие с моделью (рисунок 3.2);

– пользовательский интерфейс, обеспечивающий удобный доступ к функциональности проекта через веб-приложение (рисунок 3.3).

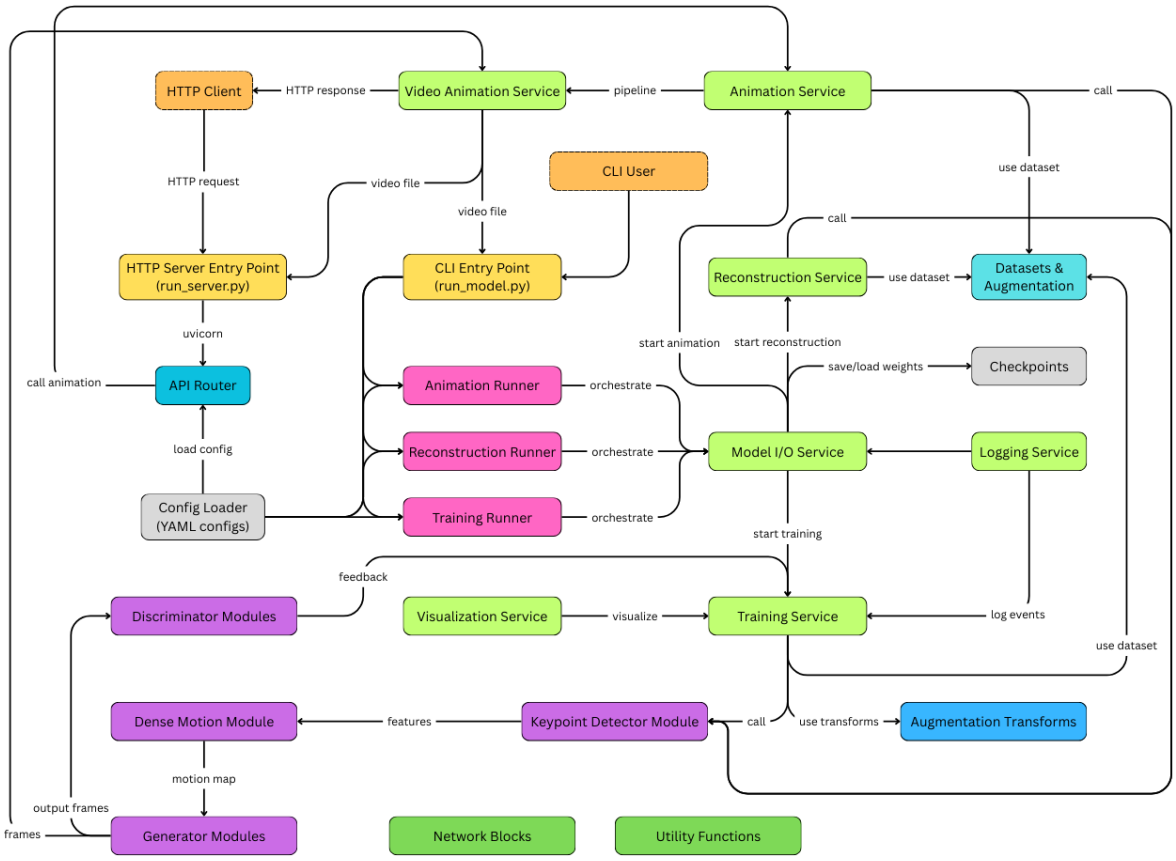


Рисунок 3.1 – Взаимодействие компонентов внутри блока нейронной сети

Внутри каждого из компонентов системы выделяются отдельные модули, логически разграничивающие ответственность и упрощающие сопровождение системы. Каждый модуль, как правило, реализуется в виде одного или нескольких классов, взаимодействующих между собой через явно

определенные интерфейсы. Модули построены на основе блоков, спроектированных на этапе системного проектирования.

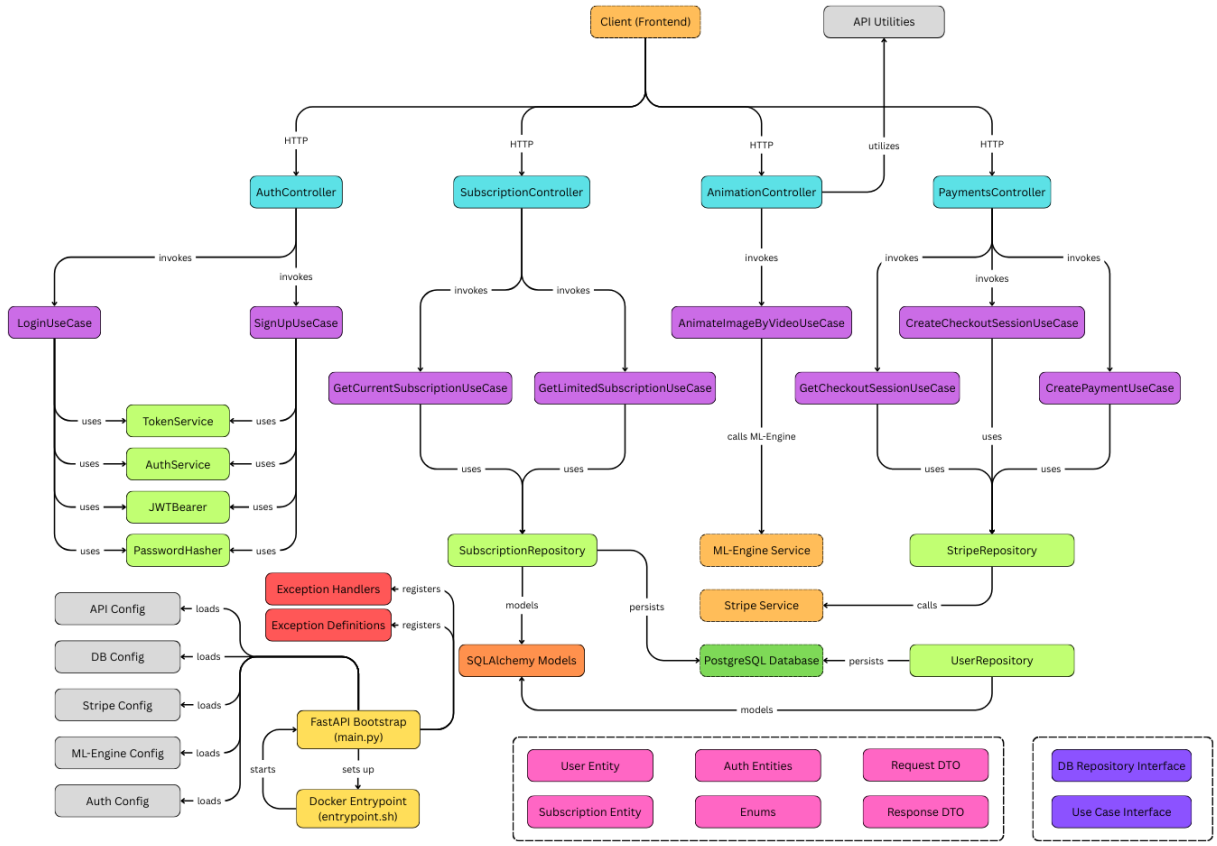


Рисунок 3.2 – Взаимодействие компонентов внутри блока backend-сервера

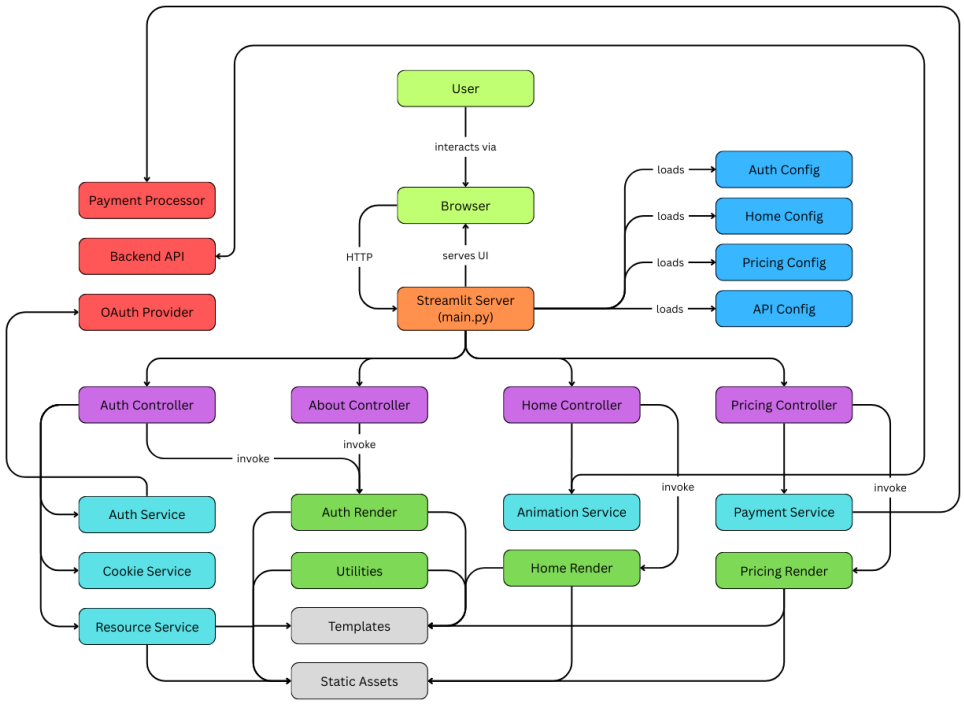


Рисунок 3.3 – Взаимодействие компонентов внутри блока пользовательского интерфейса

В таблице 3.1 приведено логического разделение блоков на соответствующие модули.

Таблица 3.1 – Разделение блоков приложения на модули

| Названия блоков | | Названия модулей |
|---|---|---|
| Блок нейронной сети | Блок модели | Модуль слоев модели нейронной сети, модуль плотного движения, модуль дискриминатора, модуль генератора, модуль детектора ключевых точек, модуль вспомогательных классов, модуль запуска модели, модуль сервисов |
| | Блок API | Модуль API нейронной сети |
| | Блок данных | Модуль конфигурации, модуль датасетов, модуль аугментации |
| Блок серверной части | Блок аутентификации | Модуль бизнес–логики, модуль конфигурации и ошибок, модуль работы с БД, модуль сервисов для аутентификации |
| | Блок анимации изображений | Модуль бизнес-логики, модуль конфигурации и ошибок |
| | Блок оплаты | Модуль бизнес-логики, модуль конфигурации и ошибок |
| | Блок подписок | Модуль бизнес-логики, модуль работы с БД |
| Блок пользовательского интерфейса | | Модуль рендеров страниц, модуль сервисов, модуль конфигурации, модуль ресурсов |

Детализированное описание модулей и их взаимодействие представлено в последующих подразделах. Диаграмма последовательности иллюстрирует процессы регистрации, входа в аккаунт, оплаты и анимации изображения, представлена на чертеже ГУИР.400201.042 РР.2. Диаграмма классов представлена на чертеже ГУИР.400201.042 РР.1, на котором отображены разработанные классы данного проекта.

### 3.2 Модуль слоев модели нейронной сети

Модуль слоев модели отвечает за построение базовых строительных блоков нейросетевой архитектуры, используемой в проекте. Он реализован на языке Python с использованием фреймворка PyTorch и предоставляет универсальные компоненты, необходимые для построения энкодеров, декодеров и архитектур типа «песочные часы» (англ. hourglass). Модуль включает как стандартные сверточные блоки с нормализацией и активацией,

так и специализированные элементы, такие как антиалиасинговая интерполяция.

Все классы в модуле реализованы в виде наследников `torch.nn.Module`, которые можно адаптировать при построении моделей. Они отделены от логики обучения и взаимодействия с внешними компонентами, что соответствует принципам модульности и повторного использования.

**3.2.1** Класс `AntiAliasInterpolation2d` предназначен для понижающего масштабирования входных изображений с применением антиалиасинга – гауссовой фильтрации, которая позволяет сохранить ключевые характеристики сигнала и избежать появления артефактов при уменьшении размера изображения.

Экземпляр класса инициализируется двумя параметрами: количеством каналов `channels` и коэффициентом масштабирования `scale`.

В классе определены следующие переменные экземпляра:

1 `scale` (тип `float`) – коэффициент масштабирования. Значение должно быть $\leq 1$.

2 `groups` (тип `int`) – количество групп в свертке (входных каналов).

3 `ka` и `kb` (тип `int`) – параметры симметричного паддинга по каждому краю, вычисляются на основе размера гауссова ядра.

4 `weight` (тип `torch.Tensor`, регистрируется как буфер) – нормализованное гауссово ядро, развернутое на нужное количество каналов.

Метод `forward` принимает тензор `x` формы «B, C, H, W» (batch размер, каналы, высота, ширина). Если масштабирование не требуется, возвращает `x` без изменений. В противном случае сначала добавляет паддинг, затем применяет 2D-свертку с ранее рассчитанным гауссовым ядром, после чего уменьшает изображение до нужного масштаба с помощью билинейной интерполяции. Возвращается тензор той же размерности по числу каналов, но с уменьшенными пространственными размерами.

**3.2.2** Класс `ResBlock2d` реализует двухслойный остаточный блок, сохраняющий пространственное разрешение входного тензора. Такой тип блока используется в сверточных нейронных сетях для увеличения глубины модели без деградации градиента при обучении, за счет добавления остаточных связей.

При создании экземпляра класса задаются параметры:

– `in_features` (тип `int`) – количество входных/выходных каналов;

– `kernel_size` (тип `int`) – размер сверточного ядра;

– `padding` (тип `int`) – значение паддинга.

Внутри блока определены две сверточные операции (`conv1` и `conv2`) с одинаковыми параметрами, а также определены два слоя пакетной нормализации `BatchNorm2d` (`norm1` и `norm2`). Обе операции работают в пространстве `in_features` каналов, что делает блок совместимым с

остаточной связью.

Метод `forward` принимает входной тензор `x` размерности (B, C, H, W) и выполняет последовательность операций, изображенную на рисунке 3.2. Затем результат суммируется с исходным `x`. Возвращается тензор той же формы, что и вход.

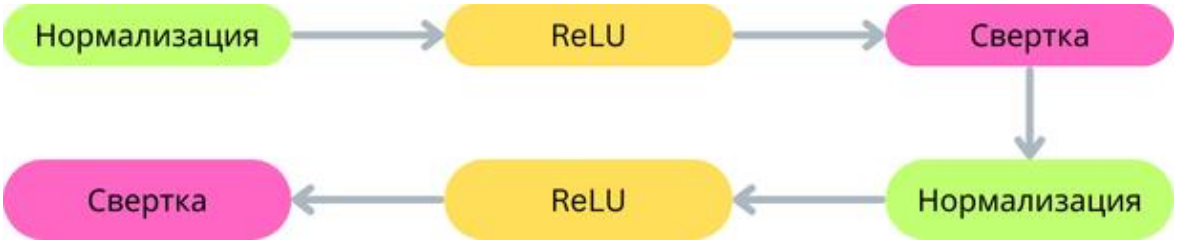


Рисунок 3.2 – Последовательность операции метода `forward` класса `ResBlock2d`

**3.2.3** Класс `UpBlock2d` представляет собой модуль повышения разрешения, который используется в декодирующей части нейронной сети – обычно после сжатия изображения (в энкодере) для восстановления его пространственных размеров. Это типичная часть архитектур типа автоэнкодеров и hourglass-сетей.

При инициализации задаются следующие параметры:

– `in_features` (тип `int`) – количество каналов во входном тензоре;

– `out_features` (тип `int`) – количество каналов на выходе блока;

– `kernel_size` (тип `int`, по умолчанию 3) – размер ядра свертки;

– `padding` (тип `int`, по умолчанию 1) – значение паддинга;

– `groups` (тип `int`, по умолчанию 1) – количество групп для групповой свертки.

Внутри класса определены два слоя: сверточный слой `conv`, выполняющий трансформацию пространственных признаков, и слой пакетной нормализации `norm`, стабилизирующий обучение.

Метод `forward` выполняет следующие действия: сначала входной тензор `x` масштабируется в два раза по высоте и ширине при помощи билинейной интерполяции (`F.interpolate`), затем применяется свертка и нормализация, после чего результат проходит через функцию активации ReLU. Возвращаемый тензор имеет увеличенное разрешение и количество каналов, заданное параметром `out_features`.

**3.2.4** Класс `DownBlock2d` реализует блок понижения разрешения, предназначенный для использования в энкодере нейронной сети. Его задача – постепенно уменьшать пространственное разрешение изображения, одновременно увеличивая глубину признаков. Это позволяет модели извлекать более абстрактные и обобщенные представления входных данных.

В конструкторе задаются следующие параметры:

– in_features (тип int) – количество входных каналов;
– out_features (тип int) – количество выходных каналов;
– kernel_size (тип int, по умолчанию 3) – размер ядра свертки;
– padding (тип int, по умолчанию 1) – значение паддинга;
– groups (тип int, по умолчанию 1) – параметр групповой свертки.

Внутри класса определены три слоя:

– conv – сверточный слой, преобразующий входные признаки в заданное число выходных каналов;

– norm – слой пакетной нормализации (BatchNorm2d), улучшающий стабильность и скорость обучения;

– pool – слой усредняющего пулинга (AvgPool2d), уменьшающий высоту и ширину тензора вдвое.

Метод forward реализует прямое прохождение данных: к входному тензору применяется свертка, затем нормализация и функция активации ReLU, после чего результат проходит через слой пуллинга. Возвращается тензор с уменьшенным в 2 раза пространственным разрешением и преобразованными признаками.

Блок используется в иерархических архитектурах для постепенного сокращения размеров входа и увеличения семантической емкости признаков.

**3.2.5** Класс SameBlock2d реализует базовый сверточный блок, сохраняющий пространственное разрешение входного тензора. Назначение блока – выполнить линейное преобразование входных признаков с возможностью изменения количества каналов, сохранив при этом высоту и ширину тензора.

В конструкторе задаются следующие параметры:

– in_features (тип int) – количество входных каналов;
– out_features (тип int) – количество выходных каналов;
– groups (тип int, по умолчанию 1) – параметр для групповой свертки;
– kernel_size (тип int по умолчанию 3) – размер ядра свертки;
– padding (тип int по умолчанию 1) – значение паддинга.

Внутри блока определены два основных слоя:

– conv – сверточный слой nn.Conv2d, который выполняет преобразование признаков;

– norm – слой пакетной нормализации BatchNorm2d, стабилизирующий обучение и ускоряющий сходимость модели.

Метод forward принимает на вход тензор признаков, к которому последовательно применяются сверточный слой, затем слой нормализации и функция активации ReLU. Возвращается тензор с той же шириной и высотой, но с другим числом каналов, заданным параметром out_features.

**3.2.6** Класс Decoder реализует декодирующую часть hourglass-архитектуры – симметричной нейросетевой структуры, предназначенной для

восстановления пространственного разрешения и детализированной реконструкции признаков, полученных после сжатия энкодером.

Экземпляр класса инициализируется следующими параметрами:

– `block_expansion` (тип `int`) – базовый множитель числа каналов;

– `in_features` (тип `int`) – число входных каналов;

– `num_blocks` (тип `int`, по умолчанию 3) – количество уровней в декодере (глубина);

– `max_features` (тип `int`, по умолчанию 256) – максимальное допустимое число каналов на любом уровне декодера.

Внутри класса создается список блоков `up_blocks`, реализованный через `nn.ModuleList`, каждый из которых представляет собой объект `UpBlock2d`. Эти блоки выполняют поэтапное увеличение пространственного разрешения признаков. Количество каналов на каждом уровне рассчитывается с учетом масштаба `scale`, где глубже расположенные блоки имеют большее число признаков, но оно ограничено сверху значением `max_features`.

Параметр `out_filters` хранит итоговое число каналов после прохождения всех апсемплирующих блоков, и рассчитывается как сумма `block_expansion` и `in_features`.

Метод `forward` принимает список тензоров, сформированных энкодером. Он извлекает самый глубокий из них и последовательно подает его в каждый апсемплирующий (англ. upsampling) блок. После каждого апсемплирования результат объединяется с соответствующим тензором-пропуском (англ. skip connection) через конкатенацию вдоль размерности каналов. Это позволяет сохранить информацию с разных уровней абстракции и повысить качество реконструкции. Финальный тензор возвращается как выход декодера.

Таким образом, `Decoder` отвечает за обратную трансформацию признаков с малым пространственным разрешением обратно в более детализированное изображение с учетом информации, переданной по скип-соединениям из энкодера.

**3.2.7** Класс `Encoder` реализует энкодерную часть hourglass-архитектуры, предназначенную для последовательного извлечения признаков из входного изображения и снижения его пространственного разрешения.

При инициализации класс принимает следующие параметры:

– `block_expansion` (тип `int`) – коэффициент, определяющий базовое количество каналов на каждом уровне;

– `in_features` (тип `int`) – количество входных каналов;

– `num_blocks` (тип `int`, по умолчанию 3) – количество уровней или блоков свертки и понижающей выборки;

– `max_features` (тип `int`, по умолчанию 256) – верхний предел количества признаков (каналов).

Внутри конструктора формируется список модулей `down_blocks` –

последовательность блоков понижающей выборки, реализованных с помощью класса DownBlock2d. Первый блок получает на вход исходное количество каналов (in_features), тогда как последующие блоки получают уже увеличенное число каналов, вычисленное через вспомогательную функцию num_channels, которая масштабирует размерность признаков в зависимости от глубины, но не превышает max_features.

Метод forward принимает входной тензор x и проходит его через каждый блок из down_blocks, последовательно уменьшая пространственное разрешение и увеличивая количество каналов. На каждом этапе результат сохраняется в список outs, включая и исходный тензор. Этот список возвращается в конце и используется, как правило, декодером, чтобы сформировать skip connection, обеспечивающие сохранение пространственной информации.

**3.2.8** Класс Hourglass реализует архитектуру типа hourglass, широко используемую в задачах генерации и обработки изображений, где важно уметь одновременно улавливать как глобальные, так и локальные структуры. Такая архитектура состоит из энкодера и декодера, соединенных через skip connection. При инициализации объект Hourglass принимает следующие параметры:

– block_expansion (тип int) – базовое количество каналов, определяющее ширину сети на первом уровне;

– in_features (тип int) – число каналов входного тензора;

– num_blocks (тип int, по умолчанию 3) – количество уровней вложенности, определяющее глубину модели;

– max_features (тип int, по умолчанию 256) – максимальное число каналов, ограничивающее рост размерности признаков.

Экземпляр класса включает два ключевых компонента:

– encoder – объект класса Encoder, выполняющий последовательное понижение разрешения и извлечение признаков;

– decoder – объект класса Decoder, восстанавливающий пространственное разрешение и объединяющий признаки скип-соединений;

– out_filters – числовое значение, содержащее количество каналов на выходе декодера.

Метод forward принимает входной тензор x, сначала обрабатывает его энкодером, который возвращает список промежуточных представлений. Эти данные затем подаются в декодер, который восстанавливает пространственную структуру и возвращает итоговый результат.

### 3.3 Модуль плотного движения

Данный модуль представлен единственным классов DenseMotionNetwork – нейронной сетью, предназначенной для

предсказания плотного поля движения между изображениями на основе разреженных представлений ключевых точек. Он использует архитектуру hourglass и обучается аппроксимировать деформации объектов путем анализа различий между двумя наборами ключевых точек: из исходного (`kp_source`) и управляющего (`kp_driving`) изображений.

Переменные экземпляра:

– `num_kp` (тип `int`) – количество ключевых точек, на основе которых строится представление движения;

– `scale_factor` (тип `int`) – коэффициент масштабирования входного изображения (для понижения разрешения);

– `kp_variance` (тип `float`) – параметр для гауссового ядра при построении тепловых карт ключевых точек;

– `hourglass` (тип `Hourglass`) – сверточная сеть, извлекающая признаки и предсказывающая маску движения на основе входных признаков.

– `mask` (тип `nn.Conv2d`) – сверточный слой, предсказывающий веса для каждой из возможных деформаций, включая фон;

– `occlusion` (тип `nn.Conv2d` или `None`) – дополнительный выход, предсказывающий карту окклюзий;

– `down` (тип `AntiAliasInterpolation2d` или `None`) – модуль для сглаживающего даунсемплинга (англ. downsampling) входного изображения.

Методы класса:

1 `create_heatmap_representations` генерирует тепловые карты, отражающие разность между ключевыми точками на управляющем и исходном изображениях. Эти карты подаются в основную сеть в качестве признаков движения.

2 `create_sparse_motions` строит «разреженные» движения (аффинные преобразования) для каждой ключевой точки. При наличии якобианов дополнительно учитываются локальные искажения.

3 `_compute_deformation` вычисляет итоговое деформационное поле как взвешенную сумму разреженных движений по предсказанной маске.

4 `create_deformed_source_image` применяет разреженные движения к исходному изображению, создавая множество вариантов деформированного изображения.

5 `forward` – основной метод прямого прохода. Сначала при необходимости понижает разрешение входного изображения. Затем формирует входные признаки: тепловые карты и деформированные изображения. После этого они подаются в hourglass, которая предсказывает маску движения. На выходе возвращает словарь с:

– `sparse_deformed` – набор изображений, деформированных с использованием разреженных движений;

– `mask` – веса, определяющие вклад каждой деформации;

– `occlusion_map` – (опционально) карта окклюзий;

– `deformation` – итоговое поле движения, интерполированное по

маске.

В целом, DenseMotionNetwork предназначена для совместной обработки движений и формы объектов путем объединения разреженных геометрических преобразований и предсказанных масок, что особенно полезно в задачах анимации, переноса мимики и реконструкции движений.

### 3.4 Модуль дискриминатора

**3.4.1** Класс DownBlock2d представляет собой базовый сверточный блок, применяемый в энкодере дискриминатора. Он предназначен для поэтапного уменьшения пространственного разрешения входных признаков и извлечения информации, необходимой для различения реальных и синтезированных данных.

Переменные экземпляра:

– conv (тип nn.Conv2d) – сверточный слой с ядром заданного размера, может быть обернут в спектральную нормализацию при включенном параметре sn;

– norm (тип nn.InstanceNorm2d) – нормализующий слой;

– pool (nn.AvgPool2d) – слой усредняющего пулинга.

Методы forward последовательно применяет к входному тензору свертку, нормализацию, LeakyReLU-активацию и (при необходимости) пулинг. Возвращает обработанный тензор с пониженным разрешением и преобразованными признаками.

**3.4.2** Класс Discriminator реализует патч-базовый дискриминатор, вдохновленный архитектурой Pix2Pix. Его задача – определять, являются ли входные изображения реалистичными на уровне локальных патчей. Модель может быть дополнительно усилена информацией о ключевых точках, если параметр use_kp активен, что делает дискриминатор чувствительным к геометрическому искажению формы объектов.

Переменные экземпляра:

– use_kp (тип bool) определяет, используется ли информация о ключевых точках при формировании входа;

– kp_variance (тип float) – дисперсия для генерации гауссовых тепловых карт ключевых точек;

– down_blocks (тип nn.ModuleList) – последовательность сверточных блоков DownBlock2d, понижающих разрешение и извлекающих признаки;

– conv (тип nn.Conv2d) – финальный сверточный слой, формирующий карту предсказаний размером 1×H×W.

В зависимости от настроек, первый блок получает на вход либо изображение, либо изображение, дополненное тепловыми картами ключевых точек. Каждый следующий блок увеличивает количество признаков и (почти всегда) понижает разрешение. Последний слой – это сверточный слой,

предсказывающий «реалистичность» каждого патча.

Метод forward принимает тензор x и, опционально, словарь kp с координатами ключевых точек. Если use_kp включен, из ключевых точек формируется тепловая карта, которая объединяется с изображением. Далее данные проходят через все сверточные блоки, а на каждом этапе сохраняются промежуточные карты признаков (они могут использоваться, например, в функции многомасштабных потерь). Финальный слой выдает карту предсказаний, содержащую патч-оценки. Возвращается кортеж из списка промежуточных признаков и итоговой карты предсказаний.

**3.4.3** Класс MultiScaleDiscriminator реализует дискриминатор, способный анализировать изображения на нескольких пространственных масштабах одновременно. Такой подход повышает устойчивость к изменениям масштаба объектов и позволяет более точно оценивать реалистичность сгенерированных изображений как в глобальном, так и в локальном контексте. Каждый масштаб обрабатывается собственным экземпляром базового дискриминатора (Discriminator), использующего одни и те же параметры, но разные входные разрешения.

Переменные экземпляра:

– scales (тип Iterable[float]) – список масштабов, на которых будет производиться оценка входных изображений;

– discs (тип nn.ModuleDict) – словарь, сопоставляющий каждому масштабу объект Discriminator.

Метод forward принимает словарь x, где ключи представляют собой строки вида «prediction_<scale>», а значениями являются изображения соответствующего масштаба. Также опционально может быть передан словарь kp с ключевыми точками. Каждый дискриминатор обрабатывает свое изображение, возвращая промежуточные признаки и итоговую карту предсказаний. Результаты собираются в выходной словарь, где каждому масштабу соответствуют ключи вида: «feature_maps_<scale>» и «prediction_map_<scale>».

**3.4.4** Класс DiscriminatorFullModel объединяет дискриминатор, извлечение ключевых точек и расчет потерь в рамках одного модуля. Он предназначен для эффективного обучения в условиях использования нескольких масштабов и мультимодульной архитектуры, упрощая вычисление дискриминаторной ошибки и обеспечивая корректное разделение графов вычислений между генератором и дискриминатором.

Переменные экземпляра:

– kp_extractor (тип nn.Module) – модуль, извлекающий ключевые точки из изображений;

– generator (тип nn.Module) – генератор, создающий предсказанное изображение на основе ключевых точек и исходных кадров;

– discriminator (тип nn.Module) – дискриминатор, который

оценивает реалистичность настоящих и сгенерированных изображений;

– train_params (тип dict) – словарь с параметрами обучения, в том числе с весами потерь;

– scales (тип list[float]) – список пространственных масштабов, используемых в ImagePyramide и дискриминаторе;

– loss_weights (тип dict) – словарь с коэффициентами весов для различных компонентов функции потерь;

– pyramid (тип ImagePyramide) – вспомогательный модуль, создающий масштабную пирамиду изображений для подачи на вход дискриминатору.

Метод forward принимает два словаря: x – с исходными входными данными, и generated – с результатами генератора, включая предсказанные кадры и ключевые точки. На основе них он:

– строит масштабные пирамиды изображений (реальных и сгенерированных) с помощью ImagePyramide;

– отсоединяет ключевые точки от графа генератора;

– пропускает обе пирамиды через дискриминатор, получая карты предсказаний на всех масштабах;

– вычисляет потери по формуле LSGAN (реальным изображениям должно соответствовать значение близкое к 1, сгенерированным – к 0);

– возвращает словарь disc_gan с итоговой потерей дискриминатора.

### 3.5 Модуль генератора

**3.5.1** Класс Vgg19 реализует сверточную нейросеть VGG-19, предобученную на датасете ImageNet. Она используется исключительно для вычисления перцептивной ошибки – меры, оценивающей сходство между изображениями в пространстве признаков промежуточных слоев сети. В этом случае модель не обучается и служит фиксированным экстрактором признаков.

Переменные экземпляра:

1 slices (тип torch.nn.ModuleList) – список из пяти последовательных фрагментов слоев предобученной VGG-19. Эти фрагменты соответствуют выходам слоев relu1_2, relu2_2, relu3_2, relu4_2, relu5_2 и используются для получения промежуточных представлений изображения.

2 mean (тип torch.Tensor) – тензор со средними значениями по каналам RGB, используется для нормализации входного изображения.

3 std (тип torch.Tensor) – тензор со стандартными отклонениями по каналам RGB, также применяется при нормализации.

Метод forward принимает изображение x в виде тензора и возвращает список признаков, извлеченных с разных уровней абстракции в сети VGG. На входе изображение нормализуется с помощью средних значений и стандартных отклонений, соответствующих обучению VGG на ImageNet.

Далее оно последовательно прогоняется через каждый из пяти блоков, накопленных в `slices`. На каждом этапе сохраняются выходы, которые затем возвращаются в виде списка. Эти признаки используются для расчета перцептивной ошибки между сгенерированным и целевым изображением, что способствует более реалистичной генерации, особенно в задачах, где важна структура и текстура.

**3.5.2** Класс `OcclusionAwareGenerator` реализует основной генератор модели, который синтезирует новое изображение, трансформируя исходное изображение в соответствии с движением, заданным ключевыми точками. В архитектуре генератора используются симметричные энкодер-декодерные блоки с пропуском через узкое бутылочное горлышко (англ. bottleneck).

Переменные экземпляра:

– `dense_motion_network` (тип `DenseMotionNetwork`) – модуль, предсказывающий деформацию, маски и, опционально, карты окклюзий на основе ключевых точек;

– `first` (тип `SameBlock2d`) – начальный сверточный блок, расширяющий количество каналов и извлекающий первичные признаки;

– `down_blocks` (тип `nn.ModuleList`) – список нисходящих блоков, уменьшающих пространственное разрешение и увеличивающих количество признаков;

– `bottleneck` (тип `nn.Sequential`) – последовательность residual-блоков, которые обрабатывают глубокое представление признаков;

– `up_blocks` (тип `nn.ModuleList`) – список восходящих блоков, восстанавливающих пространственное разрешение до исходного;

– `final` (тип `nn.Conv2d`) – выходной слой, преобразующий декодированные признаки обратно в изображение заданного числа каналов;

– `estimate_occlusion_map` (тип `bool`) – флаг, определяющий, будет ли `DenseMotionNetwork` предсказывать окклюзионную карту;

– `num_channels` (тип `int`) – число цветовых каналов на выходе.

Статический метод `deform_input` применяет предсказанную деформацию к входному тензору. Если размер тензора не совпадает с размером карты деформаций, последняя интерполируется до нужных размеров. Это позволяет учитывать движение объекта между кадрами, переносимое через ключевые точки.

Метод `forward` принимает исходное изображение и два набора ключевых точек: `kp_driving` и `kp_source`. На первом этапе изображение пропускается через нисходящие блоки и сжимается в пространственном размере. Если подключен `DenseMotionNetwork`, вычисляется карта деформаций, маски и, возможно, карта окклюзий. Признаки исходного изображения и само изображение деформируются соответствующим образом. Признаки проходят через bottleneck-блоки, затем восстанавливаются через `up_blocks`. В финале используется сверточный слой и сигмоида для

получения выходного изображения. Метод возвращает словарь, содержащий предсказанное изображение, а также дополнительные промежуточные карты: `mask`, `sparse_deformed`, `occlusion_map`, `deformed`.

**3.5.3** Класс `GeneratorFullModel` служит объединяющей оболочкой для генератора, дискриминатора и извлекателя ключевых точек с целью централизованного расчета всех соответствующих потерь в процессе обучения. Он реализует полный прямой проход генератора и производит вычисление некоторых потерь.

Переменные экземпляра

– `kp_extractor` (тип `KPDetector`) – модуль для извлечения ключевых точек из изображений;

– `generator` (тип `OcclusionAwareGenerator`) – генератор изображений, который создает предсказанное изображение на основе пары ключевых точек, извлеченных из исходного и управляющего изображений;

– `discriminator` (тип `MultiscalDiscriminator`) – дискриминатор, используется при расчете потерь и для получения промежуточных карт признаков для сопоставления;

– `train_params` (тип `dict`) – словарь, содержащий параметры обучения, включая веса всех типов потерь, список масштабов для пирамиды изображений, а также параметры геометрических трансформаций;

– `scales` (тип `list`) – список масштабов, на которых будет производиться построение пирамиды изображений для оценки качества генерации;

– `disc_scales` (тип `list`) – список масштабов, на которых работает дискриминатор;

– `loss_weights` (тип `dict`) – словарь с весами для каждого типа потерь;

– `pyramid` (тип `ImagePyramide`) – объект `ImagePyramide`, используемый для построения изображений с разными масштабами;

– `vgg` (тип `Vgg19`) – сверточная сеть, задействованная при включенной перцептивной потере.

Метод `forward` принимает словарь `x`, содержащий изображения `source` и `driving`, и реализует полный прямой проход всей связки моделей. На первом этапе извлекаются ключевые точки из обоих изображений. Затем генератор создает предсказание на основе этих ключевых точек. Полученный результат дополняется в словарь `generated`.

Если указаны веса для перцептивной потери, производится прогон реальных и предсказанных изображений через `Vgg19`, с последующим сравнением признаков на каждом уровне. Далее, если включена GAN-потеря, выполняется прогон пирамидных изображений через дискриминатор, и рассчитываются соответствующие ошибки генератора. В случае, если заданы веса для сопоставления признаков, дополнительно сравниваются промежуточные карты признаков с выходов дискриминатора.

При необходимости, выполняется геометрическая трансформация управляющего кадра, с повторным извлечением ключевых точек, и далее оценивается эквивариантность ключевых точек.

Метод возвращает два объекта: словарь `loss_values`, содержащий значения всех рассчитанных потерь, и словарь `generated`, включающий в себя сгенерированное изображение и сопутствующую информацию.

### 3.6 Модуль детектора ключевых точек

Данный модуль представлен единственным классом `KPDetector`, который реализует обнаружение ключевых точек на изображении с возможностью дополнительной оценки локальных линейных преобразований в их окрестности. Он выполняет извлечение координат ключевых точек из карты активаций, а при необходимости определяет и локальную деформацию.

Переменные экземпляра:

– `temperature` (тип `float`) – скаляр, определяющий «резкость» распределения вероятностей в карте ключевых точек;

– `scale_factor` (тип `float`) – коэффициент масштабирования входного изображения;

– `estimate_jacobian` (тип `bool`) – логическое значение, указывающее, нужно ли дополнительно оценивать матрицы Якоби для каждой ключевой точки;

– `num_kp` (тип `int`) – количество извлекаемых ключевых точек;

– `predictor` (тип `Hourglass`) – hourglass-сеть, служащая в качестве экстрактора основных признаков;

– `kp` (тип `nn.Conv2d`) – сверточный слой, предсказывающий карту вероятности положения ключевых точек из признаков, полученных от `predictor`;

– `jacobian` (тип `nn.Conv2d`) – сверточный слой для оценки параметров якобианов;

– `num_jacobian_maps` (тип `int`) – количество карт, из которых будут извлекаться якобианы;

– `down` (тип `AntiAliasInterpolation2d`) – объект сглаженного понижения разрешения, который применяется к входному изображению, если `scale_factor` < 1.

Метод `forward` принимает на вход изображение `x` (тензор формы «B, C, H, W») и возвращает словарь с координатами ключевых точек и, при необходимости, их якобианами.

На первом этапе, если задан `scale_factor`, изображение масштабируется вниз. Затем проходит через `predictor`, после чего из полученной карты признаков предсказывается карта вероятностей для каждой ключевой точки. Эта карта нормализуется с учетом `temperature`. Далее вызывается функция `gaussian2kp`, которая извлекает координаты

ключевых точек из нормализованной карты вероятностей. Если активирована оценка якобианов, то из отдельных сверточных карт собираются матрицы 2×2 для каждой ключевой точки через взвешенное суммирование значений якобианов, где веса – соответствующие значения карты вероятностей.

Результатом метода является словарь, содержащий координаты ключевых точек и, при необходимости, якобианы.

### 3.7 Модуль вспомогательных классов нейронной сети

**3.7.1** Класс `ImagePyramide` реализует построение пирамиды изображений – набора копий исходного изображения, масштабированных с различными коэффициентами. Класс применяет сглаживающее понижение разрешения с помощью соответствующих модулей для каждого масштаба.

Переменные экземпляра:

– `downs` (тип `nn.ModuleDict`) – словарь модулей, где каждому ключу «`scale_<масштаб>`» соответствует экземпляр `AntiAliasInterpolation2d`, понижающий разрешение изображения до указанного масштаба;

– `scale_names` (тип `list`) – список строк с именами ключей из `downs`;

– `output_keys` (тип `list`) – список строк вида «`prediction_<масштаб>`», соответствующий итоговым ключам словаря выходов.

Метод `forward` принимает входное изображение `x` (тензор формы «B, C, H, W») и прогоняет его через все масштабные уровни из `downs`. На выходе возвращается словарь, в котором каждому ключу соответствует результат масштабирования изображения до этого уровня.

**3.7.2** Класс `Transform` реализует геометрическое преобразование изображения на основе аффинной трансформации и, при необходимости, тонкой пластической деформации (англ. TPS – Thin Plate Spline). Основное назначение – создание искусственных преобразований кадров и координат для обучения моделей с учетом эквариантности.

Переменные экземпляра:

– `bs` (тип `int`) – размер батча, используется для генерации индивидуальных параметров трансформации для каждого примера;

– `theta` (тип `list`)– тензор, представляющий параметры аффинного преобразования (смещение, масштаб, поворот);

– `tps` (тип `bool`) – флаг, указывающий, используется ли TPS-преобразование;

– `control_points` – координаты контрольных точек TPS в нормализованной сетке;

– `control_params` – параметры смещения контрольных точек TPS.

Методы класса:

1 `transform_frame` принимает входной батч изображений, строит координатную сетку и применяет к ней заданные трансформации. Возвращает тензор того же размера с трансформированным изображением.

2 `warp_coordinates` применяется к координатам формы `[bs, N, 2]`, преобразуя их с помощью как аффинной, так и TPS-трансформации.

3 `_apply_affine_transform` реализует саму аффинную трансформацию, добавляя фиктивную координату к каждой точке и перемножая ее с матрицей `theta`.

4 `_apply_tps_transform` отвечает за тонкую пластическую деформацию. Считает расстояние от каждой координаты до контрольных точек, применяет радиальную базисную функцию и искажает координаты в соответствии с параметрами `control_params`.

5 `jacobian` вычисляет якобиан преобразования.

### 3.8 Модуль запуска модели

Модуль запуска предоставляет универсальный интерфейс для работы с моделью, позволяя выполнять как обучение, так и инференс в различных режимах.

Поддерживаются два способа запуска: локальный запуск (через Jupyter Notebook или CLI) и запуск в виде веб-сервиса на базе FastAPI. При запуске из командной строки доступно три режима работы (таблица 3.2).

Таблица 3.2 – CLI-режимы работы модели нейронной сети

| Название режима | Описание |
|---|---|
| Режим генерации анимации (англ. animation mode) | Модель принимает одно изображение и последовательность кадров (англ. driving video). На выходе – видео, где исходное изображение «оживает» в соответствии с движением из driving video. |
| Режим проверки качества модели (англ. reconstruction mode) | В этом случае и исходное изображение, и видео берутся из одной и той же последовательности. Это позволяет оценить, насколько точно модель может реконструировать движение |
| Режим обучения модели на пользовательском датасете (англ. training mode) | Позволяет настроить параметры обучения, логирования, сохранения чекпоинтов и запуска обучения как с нуля, так и с предобученной модели |

Все режимы CLI запускаются с помощью Python-скрипта и поддерживают настройку через YAML-конфигурации или аргументы командной строки, что делает запуск гибким и масштабируемым.

**3.8.1** Класс `BaseRunner` – это базовый абстрактный класс, определяющий общий интерфейс для всех сценариев запуска модели. Предназначен для переопределения в классах-наследниках, каждый из которых реализует конкретную логику выполнения.

Данный класс обеспечивает базовую инициализацию: загружает конфигурации, модельные компоненты и организует доступ к параметрам запуска. Через метод `run` задается точка входа в основной процесс, которая реализуется в подклассах.

Переменные экземпляра:

– `args` (тип `Namespace`) – объект с параметрами командной строки, содержащий конфигурацию запуска;

– `log` (тип `Logger`) – объект логгера для записи различных событий;

– `device_ids` (тип `list[int]`) – список идентификаторов CUDA-устройств, на которых будет выполняться модель;

– `verbose` (тип `bool`) – флаг расширенного вывода;

– `config` (тип `dict`) – словарь с параметрами модели;

– `log_dir` (тип `str`) – путь к директории для сохранения логов, чекпоинтов и визуализаций;

– `generator` (тип `OcclusionAwareGenerator`) – генераторная часть модели;

– `discriminator` (тип `MultiscaleDiscriminator`) – дискриминаторная часть модели;

– `kp_detector` (тип `KPDetector`) – модуль извлечения ключевых точек из изображений.

**3.8.2** Класс `AnimationRunner` – конкретная реализация базового класса `BaseRunner`, предназначенная для запуска модели в режиме анимации.

Наследует также все переменные из `BaseRunner`. Переменная экземпляра `animation_service` (тип `AnimationService`) – сервисный компонент, инкапсулирующий логику создания анимаций. Отвечает за связывание ключевых точек между кадрами и генерацию предсказаний.

Метод `run` – основной метод запуска, который загружает датасет (тип `FramesDataset`) в режиме инференса (`is_train=False`) и запускает процесс анимации, передавая все необходимые параметры в `AnimationService.animate`.

**3.8.3** Класс `ReconstructionRunner` – класс, реализующий запуск модели в режиме реконструкции. Наследует абстрактный базовый класс `BaseRunner` и служит для оценки качества восстановления исходного изображения моделью. Не содержит собственных переменных, помимо унаследованных от `BaseRunner`.

Метод `run` – основной метод запуска модели нейронной сети. Загружает

датасет через FramesDataset в режиме инференса (is_train=False) и вызывает метод reconstruction из ReconstructionService, передавая в него все необходимые компоненты: конфигурацию, модели, чекпоинт, директорию логов и данные.

**3.8.4** Класс TrainingRunner – класс, реализующий запуск модели в режиме обучения. Наследует абстрактный базовый класс BaseRunner и используется для тренировки нейросетевой архитектуры на основе предоставленного датасета и конфигурации. Дополнительных переменных, кроме унаследованных от BaseRunner, не содержит.

Методы run – метод запуска процесса обучения. Он создает объект датасета FramesDataset в режиме обучения (is_train=True) с параметрами из конфигурации. Далее инициирует процесс обучения, вызывая статический метод train из TrainingService, передавая ему все необходимые компоненты.

### 3.9 Модуль сервисов нейронной сети

**3.9.1** Класс AnimationService представляет собой сервисный компонент, реализующий процесс генерации анимации на основе модели детекции ключевых точек и генератора изображений. Основная цель класса – преобразовать неподвижное изображение в видео, имитируя движение из управляющей последовательности кадров.

У класса AnimationService нет явно инициализируемых переменных экземпляра, поскольку он реализован преимущественно через @staticmethod и @classmethod.

Методы класса:

1 make_animation является классовым методом, предназначенным для генерации видео из одного изображения и управляющего видео. Он принимает исходное изображение и список кадров видео, выполняет преобразование входных данных в тензоры, извлекает ключевые точки и последовательно применяет генератор для каждого кадра. В результате возвращает список сгенерированных кадров в формате NumPy-массивов.

2 Метод animate реализует полный пайплайн генерации анимаций для датасета. Внутри происходит загрузка моделей из чекпоинта, разворачивание их на GPU при наличии, построение датасета пар и его оборачивание в DataLoader. Затем по каждой паре осуществляется прогон модели. Результаты визуализируются с помощью VisualizationService.

3 Метод normalize_kp нормализует ключевые точки из текущего кадра управляющего видео относительно исходного и первого управляющего кадра. Поддерживается опция относительного смещения и якобианов, а также масштабирование на основе оценки движения.

4 Методы _image_to_tensor, _video_to_tensor и

_tensor_to_image являются вспомогательными и служат для преобразования входных NumPy-данных в тензоры PyTorch и обратно, что обеспечивает совместимость с моделью.

5 Метод _calculate_movement_scale вычисляет коэффициент масштабирования движения между исходными ключевыми точками и начальными управляющими на основе объемов выпуклых оболочек.

**3.9.2** Класс VideoAnimationService – это высокоуровневый сервис для генерации анимационных видеороликов из одного исходного изображения и управляющего видео, используя модели генерации и детекции ключевых точек.

Переменные экземпляра:

– config_path (тип str) – путь к YAML-конфигурации модели;

– checkpoint_path (тип str) – путь к файлу, содержащему веса предобученной модели;

– source_image_path (тип str) – путь к изображению, которое будет анимироваться;

– driving_video_path (тип str) – путь к видео, по движениям которого будет создана анимация;

– result_video_path (тип str) – путь, по которому будет сохранено результирующее видео;

– relative (тип bool) – флаг, указывающий на использование относительного перемещения ключевых точек;

– adapt_scale (тип bool) – флаг, который отвечает за масштабирование движения для лучшего соответствия объектам;

– find_best (тип bool) – флаг для автоматического поиска наилучшего кадра для старта анимации;

– best_frame (тип int) – явное указание на индекс лучшего кадра;

– cpu (тип bool) – принудительное использование CPU вместо GPU;

– log (тип Logger) – логгер, используемый для записи статуса и ошибок выполнения;

– generator (тип OcclusionAwareGenerator) – модель генератора изображений;

– kp_detector (тип KPDetector) – модель детектора ключевых точек.

Методы класса:

1 _validate_paths – вспомогательный метод, проверяющий существование указанных файлов.

2 _log_init логирует ключевые параметры конфигурации запуска.

3 run – основной метод, запускающий процесс анимации. Загружает входные данные, вызывает генерацию с помощью AnimationService и сохраняет результат.

4 _generate_animations – выполняет генерацию кадров. Если

включен поиск лучшего кадра, разбивает видео на две части и генерирует анимацию отдельно вперед и назад, объединяя результат.

5 `_save_video` сохраняет сгенерированные кадры в видеофайл.

**3.9.3** Класс `ModelService` – класс, который предназначен для управления жизненным циклом моделей генерации, используемых в системе анимации. Он позволяет инкапсулировать загрузку конфигураций, создавать директории логов, инициализировать модели для обучения и загрузку моделей для инференса.

Переменные экземпляра:

– `config_path` (тип `str`) – путь к YAML-файлу конфигурации;

– `checkpoint_path` (тип `str` или `None`) – путь к файлу чекпоинта;

– `log_dir` (тип `str`) – базовая директория для сохранения логов и копий конфигураций;

– `cpu` (тип `bool`) – флаг, указывающий принудительно использовать CPU вместо GPU;

– `verbose` (тип `bool`) – флаг, который позволяет управлять расширенным выводом;

– `config` (тип `dict`) – словарь с параметрами, загруженными из конфигурационного файла;

– `device` (тип `torch.device`) – выбранное устройство для выполнения операций (CPU или CUDA).

Методы класса:

1 `_load_config` – метод, который загружает YAML-конфигурацию из указанного пути и возвращает ее в виде словаря.

2 `_prepare_log_dir` – метод, определяющий рабочую директорию. Если передан чекпоинт, рабочей директорией становится его родительская папка. Иначе создается новая папка в директории, и в нее копируется конфигурационный файл.

3 `init_training_models` инициализирует и возвращает три модели: генератор, дискриминатор и детектор ключевых точек с параметрами, указанными в конфигурации.

4 `load_eval_models` загружает и возвращает модели генератора и детектора ключевых точек в режиме `eval`, подготавливая их к инференсу. Загружаются веса из указанного чекпоинта, и при использовании CUDA оборачиваются в `DataParallelWithCallback`.

**3.9.4** Класс `LoggingService` отвечает за логирование процесса обучения модели, визуализацию результатов, сохранение и загрузку чекпоинтов.

Переменные экземпляра:

– `loss_list` (тип `list`) – список накопленных потерь за эпоху;

– `cpk_dir` (тип `str`) – директория для сохранения чекпоинтов модели;

– visualizations_dir (тип str) – директория, в которую сохраняются визуализации реконструкции;

– log_file (тип file) – файловый дескриптор файла лога;

– zfill_num (тип int) – число символов для дополнения нулями в номерах эпох;

– visualizer (тип VisualizationService) – экземпляр класса визуализации, используется для сохранения изображений;

– checkpoint_freq (тип int) – частота сохранения чекпоинтов в эпохах;

– epoch (тип int) – текущий номер эпохи;

– best_loss (тип float) – значение минимальной потери;

– names (тип list или None) – список имен метрик потерь;

– models (тип dict) – словарь с моделями, которые необходимо сериализовать.

Методы класса:

1 log_scores вычисляет среднее значение накопленных потерь по каждой метрике, форматирует строку с результатами и сохраняет ее в лог-файл. После этого список loss_list очищается.

2 visualize_rec сохраняет изображение визуализации восстановления по данным входа и выхода.

3 save_cpk сохраняет состояние всех моделей и текущую эпоху в виде чекпоинта.

4 load_cpk – статический метод для загрузки чекпоинта. Может восстанавливать веса генератора, дискриминатора, детектора ключевых точек и их оптимизаторов.

5 __enter__ и __exit__ – методы для контекстного менеджера. При выходе из контекста автоматически сохраняется текущий чекпоинт и закрывается лог-файл.

6 log_iter сохраняет потери за итерацию в loss_list.

7 log_epoch обновляет текущую эпоху и сохраняет чекпоинт, если эпоха кратна checkpoint_freq.

8 setup_logger – статический метод для настройки и возвращения объекта Logger.

**3.9.5** Класс ReconstructionService предназначен для выполнения реконструкции видео на основе обученных моделей генерации и детекции ключевых точек. Он используется для оценки качества восстановления видео, сравнивая каждый кадр с исходным, а также для сохранения визуальных результатов в виде изображений и видео.

Класс не содержит переменных экземпляра – вся логика реализована в статическом методе, и состояние не сохраняется между вызовами.

Метод reconstruction – основной и единственный метод класса, реализующий процесс реконструкции видео. Метод принимает словарь

конфигурации, модели генерации и детекции ключевых точек, путь к чекпоинту, директорию для логирования и датасет.

В процессе реконструкции метод проходит по кадрам каждого видео. Первый кадр используется как исходный, и каждый следующий реконструируется с помощью модели генерации, используя ключевые точки текущего и исходного кадра. Также на каждом шаге сохраняется визуализация результата и считается ошибка реконструкции.

После обработки видео весь ряд реконструированных кадров сохраняется как одно горизонтальное изображение. Последовательность визуализаций сохраняется в видеофайл. В лог выводится среднее значение ошибки реконструкции по всем видео.

**3.9.6** Класс `TrainingService` предназначен для организации процесса обучения моделей генерации, детекции ключевых точек и дискриминации.

Класс не содержит переменных экземпляра – все обучение реализуется в статическом методе `train`, который принимает словарь конфигурации, модели (генератор, дискриминатор, детектор ключевых точек), путь до чекпоинта, директорию для логирования, датасет и список идентификаторов устройств (GPU), если обучение происходит на нескольких устройствах.

Если конфигурация указывает на повторение датасета, то датасет разворачивается соответствующим образом. Создаются «полные» обертки моделей генератора и дискриминатора, в которые входят и сами модели, и детектор ключевых точек. Эти обертки реализуют логику прямого и обратного проходов, потери и обучение. При наличии CUDA-машин, они оборачиваются в `DataParallelWithCallback`.

Метод не возвращает никаких значений, но по завершении сохраняет состояние моделей и выводит информацию о ходе обучения.

**3.9.7** Класс `VisualizationService` отвечает за визуализацию изображений с наложением ключевых точек и различных промежуточных результатов модели.

Переменные экземпляра:

– `kp_size` (тип `int`) – размер рисуемых ключевых точек;

– `draw_border` (тип `bool`) – флаг, указывающий, нужно ли рисовать белую рамку вокруг изображений;

– `colormap` – таблица цветов, используемая для назначения уникальных цветов ключевым точкам, по умолчанию – «`gist_rainbow`».

Методы класса:

1 `draw_image_with_kp` принимает изображение и массив нормализованных ключевых точек, преобразует координаты ключевых точек в пиксели, а затем рисует на изображении цветные кружки вокруг ключевых точек. Метод возвращает изображение с наложенными ключевыми точками.

2 `create_image_column_with_kp` накладывает ключевые точки на

каждый кадр из набора изображений и формирует вертикальную колонку из этих изображений. Аналогично, `create_image_column` объединяет изображения в колонку, при необходимости добавляя рамку.

3 `create_image_grid` собирает горизонтальную сетку из колонок изображений с ключевыми точками или без них, принимая произвольное количество таких колонок.

4 `visualize` принимает исходное изображение, движущиеся кадры и словарь с выходами модели, в том числе ключевые точки и промежуточные изображения. Он формирует набор визуализаций – исходное изображение с ключевыми точками, трансформированные кадры, деформированные изображения, итоговые предсказания, карты окклюзии и отдельные деформации, объединяет их в сетку и возвращает итоговое изображение в формате `uint8` для отображения или сохранения.

### 3.10 Модуль API нейронной сети

Данный модуль отвечает за конфигурацию, обработку исключений и эндпоинты API для взаимодействия с нейросетевой моделью, реализованной через FastAPI.

Основные классы и компоненты модуля:

– `APIModeEnum` – перечисление режимов работы API;

– `LogLevelEnum` – перечисление уровней логирования;

– `APIConfig` – класс конфигурации API, загружающий параметры из файла окружения;

– `MLModelException` – базовое исключение для всех ошибок, связанных с ML-моделью;

– `exception_handlers` (тип `dict`) – словарь с обработчиками исключений для кастомных исключений и ошибок валидации;

– `router` (тип `APIRouter`) – роутер FastAPI с префиксом «`/fomm`» и тегом «`First Order Motion Model`», через который определяется конечная точка для анимации.

Методы и функции модуля:

1 `handle_exception` обрабатывает исключения, сопоставляя их с предопределенными обработчиками. Если исключение не распознано, возвращает ошибку 500. Если API не в продакшн режиме, в ответ добавляется подробная трассировка ошибки. Возвращает сообщение и HTTP-статус.

2 `app_exception_handler` – асинхронный обработчик для FastAPI, который вызывает `handle_exception` и возвращает `JSONResponse` с ошибкой.

3 `stream_and_cleanup` – генератор для потоковой передачи содержимого файла, после чего удаляет файл, обеспечивая очистку временных ресурсов.

4 `animate_image_by_video` – POST-эндпоинт, принимающий изображение и видео, сохраняющий их во временные файлы, запускающий

сервис анимации VideoAnimationService, который генерирует анимированное видео. Возвращает видео как потоковую HTTP-ответ.

### 3.11 Модуль конфигурации нейронной сети

Данный модуль представляет собой конфигурационный файл, который используется для управления параметрами обучения и архитектурой модели нейронной сети, основанной на First Order Motion Model. Он содержит все необходимые настройки: от загрузки и аугментации данных, до параметров генератора, дискриминатора, ключевых точек, а также управления обучением и визуализацией. Благодаря этому конфигу можно легко воспроизводить различные сценарии работы и настраивать поведение модели без изменения кода.

**3.11.1** Словарь dataset_params (тип dict) – основной блок, описывающий входные данные. Имеет следующие переменные:

– root_dir (тип str) – каталог с изображениями;

– frame_shape (тип list[int]) – размер кадра;

– id_sampling (тип bool) – флаг, по которому выборка производится по идентификатору субъекта, если True;

– pairs_list (тип str) – путь к CSV-файлу с парами «изображение-видео», необходимыми для обучения;

– augmentation_params (тип dict) – параметры аугментации;

– flip_param (тип dict) – включает в себя горизонтальное отражение изображения и реверс кадров по времени;

– jitter_param (тип dict) – параметр, отвечающий за цветовые искажения.

**3.11.2** Словарь model_params (тип dict) отвечает за настройки всех модулей модели: общие параметры, генератор, детектор ключевых точек и дискриминатор.

Словарь common_params (тип dict) в свою очередь имеет следующие переменные:

– num_kp (тип int) – количество ключевых точек, используемых моделью;

– num_channels (тип int) – число цветовых каналов;

– estimate_jacobian (тип bool) – флаг, который отвечает за то, будет ли модель предсказывать якобианы движения.

Словарь kp_detector_params (тип dict) включает настройки температурного коэффициента (temperature), количество блоков, расширение каналов (block_expansion), масштаб (scale_factor), и ограничение по признакам (max_features).

Словарь generator_params (тип dict) задает архитектуру

генератора: блоки, расширения, число признаков, глубину. Флаг `estimate_occlusion_map` включает генерацию карты окклюзий. Вложенный блок `dense_motion_params` описывает аналогичные параметры для блока, отвечающего за плотное движение.

Словарь `discriminator_params` (тип `dict`) описывает дискриминатор: масштабирование (`scales`), расширение блоков, количество признаков, нормализация (флаг `sn`).

**3.11.3** Словарь `train_params` (тип `dict`) представляет собой блок с гиперпараметрами обучения:

– `num_epochs` (тип `int`) – количество эпох обучения;

– `num_repeats` (тип `int`) – количество повторений каждого видео;

– `epoch_milestones` (тип `list[int]`) – точки, на которых возможно изменение learning rate или логики обучения;

– `lr_generator`, `lr_discriminator`, `lr_kp_detector` (тип `float`) – отдельные скорости обучения для компонентов модели;

– `batch_size` (тип `int`) – размер батча;

– `scales` (тип `list[float]`) – масштабирование входов;

– `checkpoint_freq` (тип `int`) – период сохранения модели в эпохах;

– `transform_params` (тип `dict`) – словарь с параметрами деформаций;

– `loss_weights` (тип `dict`) – веса компонентов функции потерь.

**3.11.4** Словарь `animate_params` представляет собой конфигурацию для генерации анимированных видео:

– `num_pairs` (тип `int`) – количество обрабатываемых пар «изображение-видео»;

– `format` (тип `str`) – формат итогового видео;

– `normalization_params` (тип `dict`) – словарь, который определяет, использовать ли относительное движение, якобиан и адаптировать ли масштаб движения.

**3.11.5** Словарь `reconstruction_params` (тип `dict`) описывает параметры генерации видео при реконструкции:

– `num_videos` (тип `int`) – количество реконструируемых видео;

– `format` (тип `str`) – формат выходного видеофайла, например «.mp4».

**3.11.6** Словарь `visualizer_params` (тип `dict`) хранит в себе параметры визуализации ключевых точек:

– `kp_size` (тип `int`) – размер отображаемых точек;

– `draw_border` (тип `bool`) – флаг, в котором указывается, требуется ли прорисовка рамки вокруг изображения;

– `colormap` (тип `str`) – название цветовой карты.

### 3.12 Модуль датасетов

Модуль датасетов отвечает за подготовку и организацию данных, используемых при обучении и инференсе нейронной сети. Он включает в себя реализацию классов, обеспечивающих чтение, трансформацию, выборку и структурирование видеоданных в форме, подходящей для подачи в модель. Также предусмотрены механизмы повторного прохода по датасету, генерации пар «изображение-видео», и обработки данных в многопроцессорной среде с поддержкой обратных вызовов для `DataParallel`.

**3.12.1** Класс `DatasetRepeater` используется для логического увеличения объема исходного датасета за счет многократного прохода по нему.

Переменные экземпляра:

– `dataset` (тип `Dataset`) – исходный датасет;

– `num_repeats` (тип `int`, по умолчанию 100) – количество полных проходов по исходному датасету, которые должен эмулировать `DatasetRepeater`.

Методы класса:

1 `__len__` возвращает общее количество элементов в расширенном датасете, равное произведению длины исходного датасета на число повторов.

2 `__getitem__` возвращает элемент исходного датасета с индексом `idx % len(dataset)`, что позволяет зацикливать выборку данных.

`DatasetRepeater` не изменяет данные, он только переопределяет методы `__len__` и `__getitem__`, обеспечивая поведение, идентичное многократному склеиванию датасета.

**3.12.2** Класс `FramesDataset` представляет собой универсальный датасет для работы с видео, подготовленных в различных форматах.

Переменные экземпляра:

– `root_dir` (тип `str`) – путь к директории с видеоданными;

– `frame_shape` (тип `tuple[int, int, int]`, по умолчанию (256, 256, 3)) – форма кадров, к которой приводится видео при чтении;

– `id_sampling` (тип `bool`, по умолчанию `False`) – включает особый режим выборки, при котором кадры выбираются из одного и того же объекта/идентификатора;

– `is_train` (тип `bool`, по умолчанию `True`) – флаг, определяющий, используется ли датасет для обучения или валидации;

– `random_seed` (тип `int`, по умолчанию 0) – фиксирует случайность при случайной разбивке на train- или test-выборку;

– `pairs_list` (тип `str` или `None`) – путь к CSV-файлу, содержащему пары видео для загрузки в `PairedDataset`;

– `augmentation_params` (тип `dict` или `None`) – параметры

аугментации, передаваемые в AllAugmentationTransform.

Методы класса:

1 __len__ возвращает количество видеофрагментов в текущем подмножестве.

2 __getitem__ возвращает словарь с данными.

Если каталог содержит train/ и test/, используется явная разбивка. Иначе применяется случайное разделение train_test_split.

**3.12.3** Класс PairedDataset представляет собой пары данных, составленные на основе базового датасета, который в свою очередь используется в задачах, где необходимо сформировать обучающие примеры в виде пар, что особенно полезно в задачах генерации, сравнения или привязки объектов.

Переменные экземпляра:

– initial_dataset (тип Dataset) – базовый датасет, на основе которого формируются пары;

– number_of_pairs (тип int) – максимальное число пар, которые необходимо сформировать;

– seed (тип int) – значение для инициализации генератора случайных чисел, которые позволяет обеспечить воспроизводимость при случайной генерации пар;

– pairs (тип list[tuple[int, int]]) – список индексов пар, где каждая пара – это два индекса элементов из initial_dataset;

Методы класса:

1 __len__ возвращает общее число пар в датасете.

2 __getitem__ по заданному индексу возвращает словарь, содержащий два подсловаря: поля, имеющие префикс driving_ и поля, имеющие префикс source_.

Если pairs_list отсутствует, пары генерируются случайным образом с помощью координатной сетки np.mgrid, откуда затем случайным образом выбирается необходимое количество пар. Если pairs_list указан, он читается как CSV-файл, и фильтруется по доступным названиям видео из initial_dataset.

**3.12.4** Класс CallbackContext является контейнером для хранения общего состояния между копиями одного и того же подмодуля при использовании DataParallel или DistributedDataParallel. Объекты этого класса создаются один раз для каждой уникальной подструктуры модуля и передаются во все копии через __data_parallel_replicate__.

Класс DataParallelWithCallback – это модифицированная версия класса DataParallel, которая позволяет вызывать обратные вызовы __data_parallel_replicate__ после создания реплик.

### 3.13 Модуль аугментации

Модуль аугментации отвечает за применение случайных преобразований к видеоклипам или изображениям, используемым в обучении моделей компьютерного зрения. Эти аугментации повышают устойчивость модели за счет внесения контролируемого шума в данные.

**3.13.1** Класс `ColorJitter` применяется для случайной цветовой аугментации изображений или видеоклипов, а именно изменения яркости, контрастности, насыщенности и оттенка.

Переменные экземпляра:

– `brightness` (тип `float`) – максимальная степень изменения яркости;

– `contrast` (тип `float`) – максимальная степень изменения контрастности изображения;

– `saturation` (тип `float`) – максимальная степень изменения насыщенности цвета;

– `hue` (тип `float`) – максимальный сдвиг оттенка.

Методы класса:

1 `__call__` позволяет использовать экземпляр класса как вызываемый объект. Он принимает список изображений и применяет к каждому элементу заданную цветовую аугментацию. Метод возвращает новый список изображений, преобразованных с учетом случайно выбранных параметров яркости, контрастности, насыщенности и оттенка.

2 `get_params` – статический метод, который генерирует случайные значения для всех указанных параметров аугментации.

3 `apply_color_jitter` является основным рабочим методом и реализует саму логику цветовой аугментации. Он сначала получает случайные параметры с помощью `get_params`, затем формирует список преобразований и случайно перемешивает их порядок. Далее эти преобразования последовательно применяются к каждому изображению в клипе. Метод автоматически определяет тип входного и использует соответствующие преобразования.

4 `apply_pipeline` – вспомогательный статический метод, применяющий последовательность трансформаций к одному изображению. Используется внутри `apply_color_jitter`.

**3.13.2** Класс `RandomCrop` используется для выполнения случайного кадрирования видео- или клип-данных, представленных в виде списка изображений. Кадрирование выполняется синхронно для всех кадров клипа – то есть, вырезается один и тот же участок из каждого изображения.

Экземпляр класса имеет переменную `size` (тип `tuple[int, int]`) – размер обрезки. Если при инициализации передано одно целое число, он трактуется как квадратная обрезка, и преобразуется во внутреннее

представление (`size, size`).

Метод `__call__` – основной метод, который делает объект вызываемым и применяет к переданному клипу операцию случайного кадрирования. На вход подается список изображений, представляющих последовательность кадров. Метод сначала проверяет формат изображений и определяет размеры оригинала. Если изображения меньше требуемого размера, они сначала дополняются до нужного размера с помощью `pad_clip`. Затем случайным образом выбирается левый верхний угол обрезки, и применяется обрезка одинакового участка ко всем изображениям с помощью `crop_clip`. Метод возвращает список кадрированных изображений в том же формате, что и входные данные.

**3.13.3** Класс `RandomFlip` выполняет случайное отражение видеоклипа во времени и/или по горизонтали.

Переменные экземпляра:

– `time_flip` (тип `bool`) – указывает, нужно ли выполнять отражение во времени;

– `horizontal_flip` (тип `bool`) – указывает, нужно ли применять горизонтальное отражение к каждому кадру.

Метод `__call__` делает объект вызываемым и применяет случайные отражения к переданному клипу. Входной параметр `clip` – это список изображений, где каждое изображение представляет один кадр в формате NumPy-массива. Результатом работы метода является модифицированный список изображений в том же формате, что и входной.

**3.13.4** Класс `RandomResize` используется для случайного масштабирования видеоклипа в пределах заданного диапазона соотношений сторон.

Переменные экземпляра:

– `ratio` (тип `tuple[float]`) – кортеж из двух чисел, определяющий диапазон масштабирования;

– `interpolation` (тип `str`) – метод интерполяции, используемый при изменении размера.

Метод `__call__` делает объект вызываемым и применяет масштабирование ко всем кадрам в переданном клипе. Входной аргумент `clip` – это список изображений в формате NumPy-массивов либо объектов PIL. Сначала определяется оригинальный размер кадра, затем случайным образом выбирается коэффициент масштабирования. Новый размер рассчитывается как произведение исходной ширины и высоты на этот коэффициент. После этого вызывается функция `resize_clip`, которая применяет изменение размера ко всему клипу.

Результатом метода `__call__` является список кадров, масштабированных до измененного размера, в том же формате, что и входной клип.

**3.13.5** Класс `RandomRotation` реализует аугментацию, при которой каждый кадр видеоклипа поворачивается на случайный угол в заданном диапазоне.

Экземпляр класса имеет переменную `degrees` (тип `tuple[float, float]`) – диапазон углов, в пределах которого случайным образом выбирается угол поворота. Если передано одно целое число, оно интерпретируется как максимальное отклонение в обе стороны. Если передается кортеж, он должен содержать два значения – минимальный и максимальный угол поворота.

Метод `__call__` применяется к клипу – списку изображений в формате NumPy-массивов или объектов PIL. Сначала случайным образом выбирается угол поворота из заданного диапазона. Затем поворот применяется ко всем кадрам. Для NumPy используется `skimage.transform.rotate`, при этом сохраняется диапазон значений исходного изображения. Для PIL используется встроенный метод `rotate`. Метод возвращает список повернутых изображений в том же формате, что и входной клип.

**3.13.6** Класс `AllAugmentationTransform` представляет собой композицию различных аугментаций, применяемых к видеоклипу последовательно.

Переменная экземпляра `transforms` (тип `list[callable]`) – список инициализированных объектов аугментаций, применяемых к каждому клипу. Формируется динамически в зависимости от переданных параметров.

При инициализации экземпляра можно передать конфигурации для каждой из поддерживаемых трансформаций:

– `resize_param` (тип `dict`) – параметры для `RandomResize`;

– `rotation_param` (тип `dict`) – параметры для `RandomRotation`;

– `flip_param` (тип `dict`) – параметры для `RandomFlip`;

– `crop_param` (тип `dict`) – параметры для `RandomCrop`;

– `jitter_param` (тип `dict`) – параметры для `ColorJitter`.

Если какой-либо из параметров не передан, соответствующая трансформация не включается в пайплайн.

Метод `__call__` применяет каждую из включенных трансформаций поочередно к переданному видеоклипу. Метод возвращает клип, прошедший через все аугментации.

### 3.14 Общие элементы серверной части

Модуль общих компонентов содержит вспомогательные классы, перечисления, конфигурации и абстракции, которые обеспечивают единообразие поведения и взаимодействия различных частей серверного приложения. Эти элементы не завязаны на конкретную предметную область, но активно используются во всех слоях архитектуры: от контроллеров до бизнес-логики и хранилищ данных.

Класс APIModeEnum представляет собой перечисление возможных режимов запуска API-сервера: LOCAL, DEV, STAGE и PROD. Он позволяет централизованно определять режим работы приложения и использовать его в конфигурациях и условиях логики, например, при определении допустимых источников CORS-запросов.

Аналогично, класс LogLevelEnum определяет допустимые уровни логирования: INFO, DEBUG, WARNING, ERROR, CRITICAL. Эти значения соответствуют уровням логов Python и используются в настройке логирования приложения, обеспечивая фильтрацию и форматирование сообщений в зависимости от окружения.

Классы RequestModel и ResponseModel наследуются от pydantic.BaseModel и используются как родительские структуры для всех входных и выходных данных API. Они обеспечивают валидацию, сериализацию и десериализацию данных, гарантируя строгую типизацию и соответствие структуре запросов и ответов. Пустые базовые классы позволяют единообразно описывать интерфейсы, а также расширять их при необходимости в конкретных эндпоинтах.

Классы ResponseFailure и ResponseSuccess инкапсулируют логику формирования HTTP-ответов на основе результата выполнения бизнес-логики. Класс ResponseSuccess используется для формирования успешных ответов, он принимает объект ResponseModel (или список таких объектов), сериализует его и возвращает в виде JSONResponse с соответствующим кодом. Класс ResponseFailure служит для обработки и формирования ошибок, возникающих в процессе выполнения. Он использует внешний обработчик исключений handle_exception, возвращая согласованный JSON-ответ с кодом ошибки и сообщением.

Интерфейс IDBRepository представляет собой асинхронный контекстный менеджер, реализующий шаблон управления сессией взаимодействия с базой данных. Внутри него создается асинхронная сессия SQLAlchemy через sessionmaker, автоматически происходит commit или rollback в зависимости от наличия ошибок при выходе из контекста. При возникновении проблем подключения генерируется исключение ConnectionError, что упрощает отладку и контроль устойчивости подключения к базе данных.

IUseCase – абстрактный базовый класс, задающий интерфейс для всех бизнес-кейсов (англ. use case). В нем определен единственный абстрактный метод execute, который должен быть реализован в конкретных реализациях. Метод принимает произвольные аргументы и возвращает один из двух типов: ResponseSuccess или ResponseFailure. Такой подход обеспечивает единообразие исполнения и возвращаемых значений на уровне бизнес-логики.

Класс APIConfig предназначен для загрузки параметров API-сервера из переменных окружения. Он использует pydantic.BaseSettings и автоматически считывает значения из соответствующего файла,

расположенного в каталоге env/. Конфигурация содержит параметры базового и frontend-URL, режима запуска, уровня логирования, секретного ключа и списка допустимых источников запросов. Через специальное свойство allowed_origins осуществляется логика динамического добавления localhost в разрешенные источники при запуске в «небоевых» режимах (LOCAL, DEV, STAGE).

DBConfig выполняет аналогичную функцию, но для подключения к базе данных. Он загружает параметры подключения (хост, порт, имя БД, пользователь, пароль и драйвер) и формирует итоговую строку подключения db_conn_url, учитывая безопасность хранения пароля с помощью типа SecretStr. Это позволяет централизованно и безопасно управлять конфигурацией доступа к СУБД.

### 3.15 Модуль бизнес-логики для аутентификации на сервере

Модуль аутентификации реализует ключевые элементы бизнес-логики, необходимые для идентификации пользователей, генерации и обработки токенов, а также передачи информации о текущем пользователе между слоями приложения.

**3.15.1** Класс TokenType представляет собой строковое перечисление (тип StrEnum), определяющее два типа токенов: access и refresh. Это позволяет явно различать назначение токена в логике генерации, валидации и обновления.

UserAuthInfo – структура данных, предназначенная для хранения минимальной информации, извлекаемой из токена авторизации. Ключевое поле sub_id представляет собой уникальный идентификатор пользователя.

Модель UserRead используется для представления информации о пользователе в ответах API. В текущей реализации она содержит единственное поле – username, представляющее имя пользователя. Параметр model_config = ConfigDict(from_attributes=True) позволяет использовать эту модель напрямую с ORM-объектами, автоматически извлекая значения из атрибутов SQLAlchemy-моделей.

**3.15.2** Класс UserLoginUseCase реализует бизнес-логику процесса входа пользователя в систему. Он отвечает за проверку учетных данных и генерацию токенов при успешной аутентификации.

Переменные экземпляра класса:

– auth_service (тип AuthService) – объект сервиса, отвечающего за проверку пользовательских данных;

– token_service (тип TokenService) – сервис, реализующий генерацию JWT-токенов (access и refresh), а также сроков их действия;

– auth_config (тип AuthConfig) – конфигурационный объект, предоставляющий настройки, необходимые для генерации токенов и

параметров аутентификации.

Вложенные модели:

1 Класс `Request` – модель входных данных, ожидаемая от клиента. Содержит два обязательных поля: `username` (тип `str`) – имя пользователя и `password` (тип `str`) – пароль пользователя.

2 Класс `Response` – модель выходных данных, отправляемая в случае успешной аутентификации: `token_name` (тип `str`) – тип токена (обычно «`Bearer`»), `access_token` (тип `str`) – JWT-токен доступа, `refresh_token` (тип `str`) – JWT-токен обновления, `access_expires_in` (тип `int`) – срок жизни access-токена в секундах, `refresh_expires_in` (тип `int`) – срок жизни refresh-токена в секундах.

Метод `execute` реализует основную логику кейса. Он принимает объект запроса типа `Request`, проверяет логин и пароль пользователя через `auth_service`, и при успехе вызывает `token_service` для генерации токенов. Возвращается объект `ResponseSuccess`, содержащий необходимые поля для авторизации.

**3.15.3** Класс `UserSignUpUseCase` реализует процесс регистрации нового пользователя в системе. Он отвечает за проверку уникальности имени пользователя и создание записи о новом пользователе в базе данных.

Переменная экземпляра класса `user_repository` (тип `UserRepository`) является объектом репозитория для взаимодействия с хранилищем данных пользователей.

Вложенный класс `Request` (наследуется от `RequestModel`) – модель входных данных, принимаемых для регистрации (имя пользователя и пароль).

Вложенный класс `Response` – модель выходных данных, возвращаемых после успешной регистрации. В данном случае не содержит полей, так как достаточно лишь подтвердить успешное выполнение.

Метод `execute` реализует основную логику кейса регистрации. Он принимает объект запроса типа `Request` и выполняет следующие шаги:

1 Открывает асинхронный контекст с репозиторием пользователей.

2 Проверяет, существует ли уже пользователь с указанным именем.

3 Если имя уникально, вызывается метод создания нового пользователя с переданным именем и паролем.

4 При успешном создании возвращается ответ `ResponseSuccess` с HTTP-статусом 201 Created.

5 В случае любой ошибки формируется ответ `ResponseFailure` с сообщением об ошибке и соответствующим статусом.

### 3.16 Модуль работы с БД для аутентификации на сервере

Данный модуль отвечает за работу с таблицей пользователей в базе данных, обеспечивая создание новых пользователей и получение информации

о них по имени пользователя.

Класс `Base` – базовый класс, являющийся корнем для всех моделей SQLAlchemy в проекте. Он наследуется от `DeclarativeBase` и служит точкой привязки метаданных и базовой функциональности ORM.

Класс `User` представляет таблицу пользователей в базе данных с названием «user». Модель используется для хранения аутентификационных данных пользователей. Основные поля:

– `id` (тип `int`) – уникальный идентификатор пользователя, первичный ключ, автоинкрементируемый;

– `username` (тип `str`) – имя пользователя, уникальное, обязательное к заполнению, индексируемое для быстрого поиска;

– `hashed_password` (тип `str`) – хеш пароля пользователя, обязательное поле.

Класс `UserRepository` – класс репозитория для работы с данными пользователей, реализующий интерфейс `IDBRepository`. Предоставляет методы для создания пользователя и получения пользователя по имени. Использует асинхронный сеанс SQLAlchemy для взаимодействия с базой.

Метод `create_user` создает нового пользователя с указанным именем и хеширует пароль с помощью `PasswordService`. Пользователь добавляется в текущую сессию для последующего коммита.

Метод `get_user_by_username` асинхронно извлекает пользователя из базы по имени. Если пользователь с таким именем найден, возвращается объект `User`, иначе `None`.

### 3.17 Модуль сервисов для аутентификации на сервере

Модуль реализует логику, связанную с аутентификацией пользователей, включая проверку учетных данных и работу с данными пользователей. Он инкапсулирует взаимодействие с репозиториями и служит промежуточным слоем между контроллерами (или use case) и слоями доступа к данным.

**3.17.1** Класс `AuthService` отвечает за аутентификацию пользователей, предоставляя методы для проверки подлинности учетных данных.

Переменные экземпляра:

– `user_repository` (тип `UserRepository`) – репозиторий для работы с пользователями в базе данных;

– `password_service` (тип `PasswordService`) – сервис для хеширования и проверки паролей.

Метод `authenticate_user` – это асинхронный метод, который принимает имя пользователя и пароль в открытом виде. Он пытается получить пользователя из базы данных через репозиторий. Если пользователь не найден или переданный пароль не совпадает с сохраненным хешем, возвращает `None`.

В случае успешной проверки возвращает объект `UserRead`.

**3.17.2** Класс `TokenService` отвечает за создание и валидацию JWT-токенов.

Переменная экземпляра класса `auth_config` (тип `AuthConfig`) – объект конфигурации с параметрами, такими как секретный ключ, алгоритм подписи, время жизни токенов и издатель.

Метод `generate_tokens` – это асинхронный метод генерации пары токенов (access и refresh). Внутри рассчитывается время жизни для каждого токена на основе настроек, затем вызывается вспомогательный метод `_create_token`, который формирует JWT с необходимыми данными. Возвращает словарь с токенами и временем их действия в секундах.

Метод `verify_token` служит для валидации переданного JWT. Использует секретный ключ и алгоритм подписи из конфигурации. Проверяет срок действия токена. При успешной проверке возвращает полезную нагрузку токена в виде словаря.

Метод `_create_token` – вспомогательный метод, который создает JWT с заданными данными и временем действия.

**3.17.3** Класс `JWTBearer` реализует класс для аутентификации запросов на основе JWT-токенов. Класс наследуется от `HTTPBearer` FastAPI и служит для извлечения и валидации токена из заголовка Authorization. `JWTBearer` интегрируется с FastAPI как класс-зависимость и обеспечивает удобную и безопасную аутентификацию запросов на основе JWT.

Основной метод `__call__` вызывается при обработке запроса. Он получает токен, переданный клиентом, и проверяет его валидность с помощью сервиса `TokenService`. В случае успешной проверки создается объект `UserAuthInfo` с идентификатором пользователя.

**3.17.4** Класс `PasswordService` реализует сервис для хеширования и проверки паролей с использованием библиотеки passlib и алгоритма bcrypt.

Класс содержит два статических метода:

1 `hash_password` принимает открытый текст пароля и возвращает его безопасный хеш. Для хеширования используется bcrypt, который является надежным и устойчивым к атакам методом.

2 `verify_password` принимает открытый текст пароля и ранее сохраненный хеш, проверяет их соответствие.

### 3.18 Модуль конфигурации и ошибок аутентификации на сервере

Данный модуль содержит настройки и определения исключений, связанных с аутентификацией пользователя.

Класс конфигурации `AuthConfig` основан на

pydantic.BaseSettings и загружает параметры аутентификации из файла окружения. Включает следующие параметры:

– SECRET_KEY – секретный ключ для подписи JWT-токенов;

– ALGORITHM – алгоритм шифрования токенов (например, HS256);

– ACCESS_TOKEN_EXPIRE_MINUTES – время жизни access-токена в минутах;

– REFRESH_TOKEN_EXPIRE_DAYS – время жизни refresh-токена в днях;

– TOKEN_ISSUER – издатель токена.

Исключения, связанные с аутентификацией:

1 AuthException – базовый класс для всех исключений, связанных с аутентификацией и авторизацией.

2 InvalidCredentialsException – исключение, которое вызывается при неверных данных пользователя (логин/пароль).

3 UserAlreadyExistsException – ошибка, возникающая при попытке зарегистрировать пользователя с уже существующим именем.

4 ExpiredTokenException – ошибка, возникающая при использовании просроченного токена.

5 InvalidTokenException – исключение, которое возникает при неверном или некорректном токене.

### 3.19 Модуль бизнес-логики для анимации изображений на сервере

Класс AnimateImageByVideoUseCase реализует бизнес-логику по анимации изображения на основе видео. Он обрабатывает запросы от пользователя, взаимодействует с ML-сервисом для выполнения анимации и следит за обновлением информации о подписке пользователя, уменьшая количество доступных запросов.

Переменные экземпляра:

– ml_engine_config (тип MLEngineConfig) – содержит конфигурационные параметры для обращения к ML-сервису;

– user_repository (тип UserRepository) – репозиторий для взаимодействия с пользовательскими данными;

– subscription_repository (тип SubscriptionRepository) – репозиторий, отвечающий за работу с подписками.

Метод execute – основной метод, реализующий бизнес-логику. На вход принимает объект request, содержащий загруженные пользователем изображение и видео, а также объект user, включающий в себя sub_id авторизованного пользователя. Метод сначала читает файлы и подготавливает их к отправке. Затем он извлекает пользователя из БД и определяет его user_id. После этого происходит обращение к ML-сервису через HTTP-запрос, куда передаются изображение и видео. В случае успешного ответа метод обновляет количество оставшихся запросов у пользователя, уменьшая

их на единицу. Наконец, возвращается потоковый ответ типа `StreamingResponse`, содержащий анимированное видео.

### 3.20 Модуль конфигурации и ошибок анимации на сервере

Класс `MLEngineConfig` представляет собой конфигурационный компонент, отвечающий за параметры подключения к внешнему ML-сервису, выполняющему задачу анимации изображения. Он автоматически загружает переменные из env-файла и обеспечивает удобный доступ к ним внутри других компонентов системы.

Переменные экземпляра класса:

– `BASE_URL` (тип `str`) – базовый URL ML-сервиса;

– `ML_ENGINE_KEY` (тип `str`) – секретный API-ключ, необходимый для авторизации запросов к ML-сервису;

– `ML_ENGINE_KEY_HEADER` (тип `str`, по умолчанию «`X-ML-Engine-Key`») – имя HTTP-заголовка, в который помещается API-ключ при выполнении запроса.

Класс `MLEngineException` – это базовый класс исключений, связанных с нештатными ситуациями при взаимодействии с ML-сервисом. Он служит родительским для потенциальных специализированных исключений.

### 3.21 Модуль бизнес-логики для оплаты на сервере

**3.21.1** Класс `StripeRepository` представляет собой обертку над SDK Stripe и инкапсулирует бизнес–логику, связанную с получением и созданием продуктов и сессий оплаты. Он используется для асинхронного взаимодействия с API Stripe внутри серверной части приложения, обеспечивая удобный и безопасный интерфейс для работы с платежной системой.

Экземпляры класса не создаются напрямую – все методы класса являются методами класса (`@classmethod`) и вызываются без необходимости инстанцирования. Конфигурация Stripe предполагается настроенной глобально вне данного класса.

Метод `get_products` получает список всех активных продуктов в Stripe. Возвращает объект `ListObject`, содержащий элементы `Product`, каждый из которых включает данные о товаре и его цене. В запрос также добавляется параметр `expand=["data.default_price"]`, чтобы сразу получить вложенную информацию о цене.

Метод `get_product` извлекает информацию о конкретном продукте по его `product_id`.

Метод `get_checkout_session` получает объект сессии оплаты (тип `Session`) по ее `session_id`.

Метод `create_checkout_session` создает новую сессию оплаты Stripe, используя переданный словарь параметров `params`. Возвращает

объект Session, содержащий URL для перенаправления и другую служебную информацию.

**3.21.2** Класс CreateCheckoutSessionUseCase реализует бизнес-логику создания платежной сессии в Stripe через сервер. Он используется при оформлении пользователем подписки или разовой покупки через интеграцию Stripe Checkout. Является частью слоя use case в архитектуре приложения.

Переменные экземпляра класса:

– redirect_url (тип str) – строка, содержащая frontend-URL, на который будет перенаправлен пользователь после успешной или отмененной оплаты;

– stripe_repository (тип StripeRepository) – репозиторий, инкапсулирующий все обращения к API Stripe.

Метод execute – основной метод, запускающий выполнение бизнес-логики. Принимает объект Request, содержащий пользователя (user) и идентификатор цены в Stripe (price_id). Сначала формирует словарь параметров params, включающий информацию о способах оплаты, списке товаров, а также URL-адресах для перенаправления после оплаты. Затем вызывает метод create_checkout_session у StripeRepository, чтобы создать сессию оплаты. В случае успеха возвращает объект ResponseSuccess с информацией о сессии в виде словаря.

**3.21.3** Класс CreatePaymentUseCase реализует бизнес-логику подтверждения успешной оплаты через Stripe и обработки подписки пользователя на сервере. Он вызывается после того, как пользователь завершает процесс оплаты, и выполняет обновление или создание записи о подписке, а также учитывает доступное количество запросов.

Переменные экземпляра класса:

– redirect_url (тип str) – строка с URL-адресом клиентского интерфейса, куда происходит перенаправление после обработки платежа;

– stripe_repository (тип StripeRepository) – репозиторий для взаимодействия с API Stripe;

– subscription_repository (тип SubscriptionRepository) – репозиторий для работы с данными о подписках пользователей в базе данных. Используется для обновления или создания записей о подписке;

– user_repository (тип UserRepository) – репозиторий, предоставляющий доступ к информации о пользователях.

Метод execute принимает объект Request, содержащий:

– user – информация об авторизованном пользователе;

– session_id – идентификатор платежной сессии Stripe;

– product_id – идентификатор продукта Stripe, купленного пользователем.

Метод выполняет следующие действия:

1 Получает продукт из Stripe и извлекает количество запросов из его метаданных.

2 Получает платежную сессию по ее идентификатору и проверяет статус оплаты.

3 Получает пользователя из базы данных по имени.

4 Проверяет наличие подписки в базе и, если подписка существует, обновляет количество запросов, если нет – создает новую.

Возвращает ответ с количеством оставшихся запросов и ID платежной сессии.

В случае ошибок Stripe или бизнес-правил (например, неоплаченной сессии или отсутствия данных о количестве запросов) возвращается объект `ResponseFailure` с описанием проблемы.

**3.21.4** Класс `GetCheckoutSessionUseCase` – это бизнес-логика, отвечающая за получение информации о сессии оплаты Stripe по ее идентификатору.

Переменная экземпляра класса `stripe_repository` (тип `StripeRepository`) – репозиторий, инкапсулирующий работу с API Stripe. Используется для получения объекта платежной сессии по ее ID.

Методы `execute` принимает объект `Request`, содержащий:

– `user` – авторизованный пользователь;

– `session_id` – строковый идентификатор платежной сессии в Stripe.

Метод выполняет запрос к Stripe через `StripeRepository` и получает объект сессии. Если операция проходит успешно, возвращается объект `ResponseSuccess`, оборачивающий словарь сессии `checkout_session`. В случае ошибок возвращается объект `ResponseFailure` с описанием исключения.

### 3.22 Модуль конфигурации и ошибок для оплаты на сервере

Данный модуль отвечает за загрузку конфигурации доступа к платежной системе Stripe, а также определяет кастомные исключения, связанные с ошибками обработки платежей.

Класс `StripeConfig` – конфигурационный класс, основанный на BaseSettings, предназначенный для хранения и загрузки параметров Stripe из env-файла.

Переменные экземпляра класса:

– `PUBLIC_KEY` (тип `str`) – публичный ключ Stripe;

– `SECRET_KEY` (тип `str`) – секретный ключ Stripe.

После инициализации класса `StripeConfig`, ключ `SECRET_KEY` устанавливается глобально в `stripe.api_key`, что позволяет использовать библиотеку Stripe без необходимости явно передавать ключ в каждом запросе.

Класс `StripeException` – базовый класс исключений, используемый для обработки всех ошибок, связанных с продуктами или логикой Stripe. Класс

StripePaymentException наследуется от StripeException. Данное исключение выбрасывается в случае проблем с обработкой платежа, например, если платежная сессия была создана, но не завершена или не прошла проверку.

### 3.23 Модуль бизнес-логики для подписок на сервере

**3.23.1** Класс Subscription – это Pydantic-модель, представляющая бизнес-сущность подписки в системе. Используется для формирования ответа о текущем статусе подписки пользователя.

Переменные экземпляра класса:

– subscription_id (тип int) – уникальный идентификатор подписки;

– user_id (тип int) – идентификатор пользователя, которому принадлежит подписка;

– remaining_queries (тип int) – количество оставшихся запросов, доступных пользователю в рамках подписки.

Класс GetCurrentSubscriptionUseCase реализует интерфейс IUseCase и предназначен для обработки запроса, связанного с получением информации о текущей подписке авторизованного пользователя.

Переменные экземпляра класса:

– user_repository (тип UserRepository) – репозиторий для работы с данными пользователя;

– subscription_repository (тип SubscriptionRepository) – репозиторий для доступа к данным подписки пользователя.

Метод execute обрабатывает входной запрос, извлекает информацию о пользователе и, если подписка существует, возвращает ее данные в виде объекта Subscription. Принимает объект Request, содержащий объект user (тип UserAuthInfo) – информацию об авторизованном пользователе.

**3.23.2** Класс GetLimitedSubscriptionsUseCase реализует интерфейс IUseCase и отвечает за получение и возврат списка подписок в ограниченном формате вместе с публичным ключом Stripe.

Переменные экземпляра класса:

– stripe_repository (тип StripeRepository) – репозиторий для асинхронного взаимодействия с API Stripe, через который происходит запрос списка продуктов;

– stripe_config (тип StripeConfig) – объект конфигурации, содержащий настройки Stripe.

Метод execute принимает объект Request, включающий данные об авторизованном пользователе. Выполняется асинхронный вызов к Stripe через репозиторий для получения активных продуктов. Полученный список продуктов вместе с публичным ключом Stripe упаковывается в модель

Response, которая затем оборачивается в ResponseSuccess и возвращается как JSON-ответ.

### 3.24 Модуль работы с БД для подписок на сервере

Данный модуль предназначен для управления сущностями подписок в базе данных. Он содержит модель данных для подписки и репозиторий для выполнения основных операций с записями подписок.

Класс Subscription описывает структуру таблицы «subscription» в базе данных с помощью SQLAlchemy ORM. Он наследуется от базового класса Base и содержит три основных поля:

– id (тип int) – уникальный идентификатор записи, первичный ключ с автоинкрементом;

– user_id (тип int) – внешний ключ, указывающий на пользователя в таблице «user», обязательное поле;

– remaining_queries (тип int) – количество оставшихся запросов в подписке, обязательное для заполнения.

Класс SubscriptionRepository – репозиторий, который отвечает за доступ и изменение данных подписок в базе. Он реализует интерфейс IDBRepository, используя асинхронные методы для взаимодействия с сессией базы данных. Основная переменная экземпляра – это async_session, обеспечивающая асинхронное выполнение запросов к базе.

Метод get_one_by_user_id асинхронно выполняет запрос к базе для получения одной подписки, связанной с указанным пользователем. Возвращает объект Subscription либо None, если подписка не найдена.

Метод add_one создает новую запись подписки с заданным количеством оставшихся запросов для пользователя с указанным user_id. Добавляет запись в сессию и вызывает flush для сохранения изменений, возвращая созданный объект.

Метод update_one обновляет существующую подписку, увеличивая количество оставшихся запросов на переданное значение. После обновления вызывает flush и возвращает обновленный объект.

### 3.25 Модуль рендеров страниц пользовательского интерфейса

**3.25.1** Класс AuthRender отвечает за отображение форм авторизации и регистрации в пользовательском веб-интерфейсе на базе Streamlit.

Переменные экземпляра:

– templates (тип dict) – словарь с HTML-шаблонами, используемыми для визуального оформления форм авторизации и регистрации;

– auth_service (тип AuthService) – экземпляр сервиса авторизации, предоставляющего методы login и register для

взаимодействия с серверной частью;

– auth_config (тип AuthConfig) – объект конфигурации, из которого подгружаются, например, условия использования сервиса.

В st.session_state также инициализируются ключи login_error, register_error, register_success для отображения статуса операций пользователю.

Метод _init_session_state – это статический метод, задающий значения по умолчанию для состояния Streamlit-сессии.

Метод render_login_form отображает форму авторизации. Включает текстовые поля для имени пользователя и пароля, кнопку входа и обработку результата входа через auth_service.login. В случае ошибки отображает соответствующее сообщение из состояния сессии.

Метод render_register_form отображает форму регистрации. Предлагает поля ввода имени, пароля и подтверждения пароля. Также содержит флажок для согласия с условиями использования и выпадающее окно для их просмотра. При успешной регистрации перезагружает страницу, иначе показывает сообщения об ошибках или успехе, в зависимости от состояния сессии.

**3.25.2** Класс HomeRender отвечает за визуальное отображение и пользовательское взаимодействие с основной страницей генерации анимации. Он обеспечивает загрузку входных файлов, отображение результатов генерации, а также обработку кнопок действий и ограничений, связанных с подпиской.

Переменные экземпляра:

– templates (тип dict) – словарь HTML-шаблонов для вставки стилизованных компонентов страницы;

– animation_service (тип AnimationService) – сервисный класс, обрабатывающий генерацию анимации на основе входных файлов;

– home_config (тип HomeConfig) – объект конфигурации, содержащий допустимые расширения для изображений и видео;

– query_balance (тип int) – количество доступных генераций, получаемое из PaymentService.

CookieService.controller используется для сохранения количества доступных генераций в cookie.

Через _init_session_state инициализируются следующие ключи:

– animation_result хранит результат генерации (видео);

– source_image – байтовое представление загруженного изображения;

– driving_video – байтовое представление загруженного видео.

Методы класса:

1 render_subscription_required отображает предупреждение о необходимости подписки.

2 render_image_upload отображает компонент загрузки изображения.

3 render_video_upload отображает компонент загрузки видеофайла.

4 render_generate_button отображает кнопку запуска генерации анимации.

5 _process_generation_request выполняет отправку загруженных данных в AnimationService и обрабатывает ответ. В случае успеха сохраняет результат и инициирует перезапуск страницы.

6 render_download_button отображает кнопку для скачивания результата генерации.

**3.25.3** Класс PricingRender отвечает за отображение страницы тарифов, управление выбором подписки, интеграцию с платежной системой Stripe, а также за обработку успешных или отмененных платежей.

Переменные экземпляра:

– templates (тип dict) – словарь HTML-шаблонов для визуальных компонентов страницы;

– auth_service (тип AuthService) – сервис аутентификации, используемый для защищенных API-запросов;

– payment_service (тип PaymentService) – сервис обработки бизнес-логики после успешной оплаты;

– cookie (тип CookieController) – контроллер управления cookie для хранения информации о выбранном плане, цене и сессии;

– api_config (тип APIConfig) – конфигурация API backend-сервера;

– pricing_config (тип PricingConfig) – конфигурация логики отображения и поведения страницы с тарифами;

– stripe_public_key (тип str) – публичный ключ Stripe, полученный с сервера;

– products (тип list) – список тарифных планов, полученных с сервера Stripe.

Методы класса:

1 _init_session_state инициализирует session_state с ключами: selected_product_id (ID выбранного пользователем продукта) и selected_price_id (ID соответствующей цены).

2 _handle_payment_callback обрабатывает query-параметры, полученные после перехода со Stripe. При success=true вызывает метод make_payment и перезагружает страницу. При canceled=true выводит предупреждение и также выполняет перезагрузку.

3 _load_products выполняет авторизованный GET-запрос к API для получения списка тарифов и публичного ключа Stripe.

4 _select_product сохраняет выбор пользователя в session_state и cookie.

5 render_plan_card рендерит карточку тарифного плана с названием, ценой, скидкой и списком фич. Активная карточка визуально выделяется.

6 render_subscribe_button рендерит кнопку подписки для выбранного плана.

### 3.26 Модуль сервисов пользовательского интерфейса

**3.26.1** Класс AnimationService предназначен для взаимодействия с серверной частью анимации. Он используется в пользовательском интерфейсе для отправки исходного изображения и управляющего видео на сервер, где происходит генерация итоговой анимации.

Переменные экземпляра класса:

– api_url (тип str) – строка с базовым backend-URL сервера;

– auth_service (тип AuthService) – экземпляр сервиса авторизации, через который отправляются все запросы.

Метод generate_animation предназначен для генерации анимации. Он принимает словарь files, содержащий два ключа: source_image и driving_video, каждый из которых представляет собой кортеж с именем файла, содержимым в байтах и MIME-типом. Метод выполняет POST-запрос на сервер, используя авторизованный метод запроса через AuthService. В случае успешного выполнения запроса метод возвращает словарь с флагом success = True, байтовым содержимым анимационного видео в ключе animation_data, а также сообщением об успешной обработке. В случае ошибки возвращается словарь с success = False, пустыми данными и текстом ошибки в ключе message.

**3.26.2** Класс AuthService отвечает за обработку логики аутентификации пользователей в пользовательском интерфейсе. Он обеспечивает вход, регистрацию, выход из аккаунта, а также выполнение авторизованных HTTP-запросов к защищенным маршрутам API.

Переменная экземпляра класса api_config (тип APIConfig) – конфигурационный объект, содержащий базовый URL для всех запросов, связанных с авторизацией.

Методы класса:

1 check_auth проверяет, авторизован ли пользователь, анализируя данные в cookie. Если токены и имя пользователя найдены, они записываются в st.session_state. В противном случае авторизация считается неуспешной.

2 login отправляет POST-запрос на сервер с указанными учетными данными. При успешной авторизации сохраняет access- и refresh-токены, а также имя пользователя в сессионное состояние и cookie. В случае ошибки возвращает False и сохраняет сообщение об ошибке в

st.session_state.login_error.

3 register выполняет регистрацию нового пользователя. Сначала проверяет, совпадают ли пароли, после чего отправляет POST-запрос на сервер. При успехе отображается сообщение о том, что регистрация завершена. В случае ошибки сохраняет сообщение в st.session_state.register_error.

4 logout очищает токены авторизации из cookie и обнуляет авторизационное состояние пользователя в Streamlit. После выполнения перезагружает интерфейс с помощью st.rerun.

5 make_authenticated_request используется для отправки авторизованных HTTP-запросов с access-токеном в заголовке. Если сервер возвращает статус 401, это считается признаком истечения сессии, и токены очищаются.

**3.26.3** Класс CookieService представляет собой сервис-обертку над контроллером cookie, предоставляемым библиотекой streamlit_cookies_controller. Он используется для управления данными авторизации и другими пользовательскими параметрами в cookie. Основное назначение – установка, получение и удаление cookie-данных, а также подготовка заголовков авторизации для HTTP-запросов.

Переменная экземпляра класса controller (тип CookieController) – экземпляр класса из сторонней библиотеки, обеспечивающий взаимодействие с cookie в Streamlit-приложении.

Методы класса:

1 set_auth_cookies сохраняет в cookie словарь с авторизационными данными, полученными после успешного входа в систему.

2 get_auth_cookies извлекает из cookie сохраненные авторизационные данные. Если данные отсутствуют, возвращает None.

3 clear_auth_cookies удаляет из cookie как авторизационные данные, так и связанные данные, такие как текущий баланс запросов.

4 get_auth_headers формирует словарь заголовков для авторизованных запросов. Если в cookie найден access_token, добавляется заголовок Authorization: Bearer <токен>. Если токен отсутствует, возвращается пустой словарь.

**3.26.4** Класс PaymentService реализует бизнес-логику, связанную с оплатой и подписками. Используется для получения текущего баланса запросов пользователя, а также для подтверждения и обработки успешных платежей.

Переменные класса:

– api_config (тип APIConfig) – конфигурационный объект, содержащий базовый URL для API-запросов;

– pricing_config (тип PricingConfig) – объект с настройками, связанными с отображением и поведением страницы тарифов, включая

задержки сообщений после оплаты.

Методы класса:

1 `get_query_balance` выполняет GET-запрос к серверу на получение текущей активной подписки пользователя и возвращает количество оставшихся анимационных запросов. Если запрос неудачен, отображает сообщение об ошибке и выполняет выход из аккаунта.

2 `make_payment` используется для подтверждения успешной покупки. Выполняет POST-запрос с `session_id` и `product_id`, полученными из cookie, на эндпоинт, обрабатывающий завершение транзакции. В случае успеха обновляет cookie с актуальным значением оставшихся запросов.

**3.26.5** Класс `ResourceService` отвечает за загрузку статических ресурсов, необходимых для визуального оформления и работы пользовательского интерфейса приложения.

Методы класса:

1 `load_styles` загружает и применяет CSS-файл из каталога `assets/styles`, соответствующий указанному имени. Содержимое стиля инжектируется в Streamlit через `st.markdown` с флагом `unsafe_allow_html=True`, что позволяет применять кастомные стили поверх стандартного оформления.

2 `load_template` предназначен для загрузки HTML-шаблонов, используемых на различных страницах приложения. Файл читается из каталога `templates/` и возвращается в виде строки, готовой к вставке в `st.markdown`.

3 `load_image` загружает изображение из каталога `assets/images` и возвращает его в виде строки, закодированной в формате base64.

### 3.27 Модуль конфигурации пользовательского интерфейса

Данный модуль содержит конфигурационные классы, определяющие ключевые параметры интерфейса и поведения приложения. Все классы реализованы на базе `pydantic_settings.BaseSettings`, что обеспечивает простую интеграцию с переменными окружения, централизованную настройку и удобную валидацию параметров.

Класс `APIConfig` служит для хранения основных URL-адресов, используемых приложением при взаимодействии с серверной частью.

Переменные экземпляра:

– `BASE_URL` (тип `str`) – базовый адрес API backend-сервера;

– `TERMS_OF_SERVICE_URL` (тип `str`) – URL страницы с условиями обслуживания.

Класс `AuthConfig` формирует текст пользовательского соглашения, отображаемого при регистрации пользователя. В его составе – разметка Markdown с основными пунктами соглашения и ссылкой на полную версию

условий. Переменная экземпляра данного класса TERMS_OF_SERVICE (тип str) – строка, содержащая текст пользовательского соглашения, включая правила использования, возрастные ограничения и ссылку на подробную информацию.

Класс HomeConfig содержит настройки, связанные с загрузкой медиафайлов и отображением сообщений на главной странице.

Переменные экземпляра:

– ALLOWED_IMAGE_TYPES (тип list) – список допустимых форматов изображений, которые можно загружать пользователю;

– ALLOWED_VIDEO_TYPES (тип list) – список допустимых форматов видео, применяемых для анимации;

– MESSAGE_DELAY (тип int) – длительность паузы (в секундах) перед обновлением интерфейса или отображением временных сообщений.

Класс PricingConfig управляет задержками в сообщениях, связанных с тарифными планами и оплатой. Единственной переменной экземпляра класса является MESSAGE_DELAY (тип int) – количество секунд, в течение которых пользователь видит информационные сообщения.

### 3.28 Модуль ресурсов пользовательского интерфейса

Модуль ресурсов обеспечивает визуальное и структурное оформление веб-интерфейса приложения. Он содержит файлы стилей CSS, изображения, а также HTML-шаблоны, которые используются для динамической генерации интерфейсных компонентов на разных страницах. Ресурсы этого модуля загружаются с помощью сервиса ResourceService, который инкапсулирует работу с файловой системой и предоставляет методы для подключения стилей, изображений и HTML-фрагментов.

Каталог assets/ содержит статические ресурсы – изображения и стили, используемые в разных частях интерфейса:

– images/ содержит изображения, используемые в интерфейсе;

– styles/ содержит CSS-файлы, стилизующие определенные разделы интерфейса.

Каталог templates/ содержит HTML-шаблоны, используемые при рендеринге интерфейса в Streamlit:

– about/ содержит единственный шаблон содержимого страницы «About»;

– auth/ содержит шаблоны, используемые для интерфейса аутентификации;

– home/ включает HTML-шаблоны, используемые на главной странице;

– pricing/ содержит шаблоны, связанные с отображением и выбором тарифных планов.

# 4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

В данном разделе рассматриваются основные этапы разработки программных модулей, реализующих функциональность дипломного проекта. Описание охватывает выбор инструментов и технологий, применяемых при реализации, подходы к проектированию архитектуры, а также построение и обучение модели машинного обучения. Помимо этого, в рамках раздела представлена разработка страниц пользовательского интерфейса.

Листинг кода приведен в приложении А. Полные исходные коды находятся на диске в архивах «Animatica-backend.zip», «Animatica-ml-engine.zip» и «Animatica-frontend.zip».

## 4.1 Инструменты и подходы к разработке

В процессе разработки программной системы используются современные практики проектирования, инструменты и стандарты, направленные на повышение качества кода, его читаемости и устойчивости, а также последующее сопровождение.

Большинство компонентов системы реализуются с использованием классов, что способствует логическому разделению ответственности, повторному использованию кода и удобству тестирования. Принципы инкапсуляции, наследования и полиморфизма активно применяются в построении архитектуры.

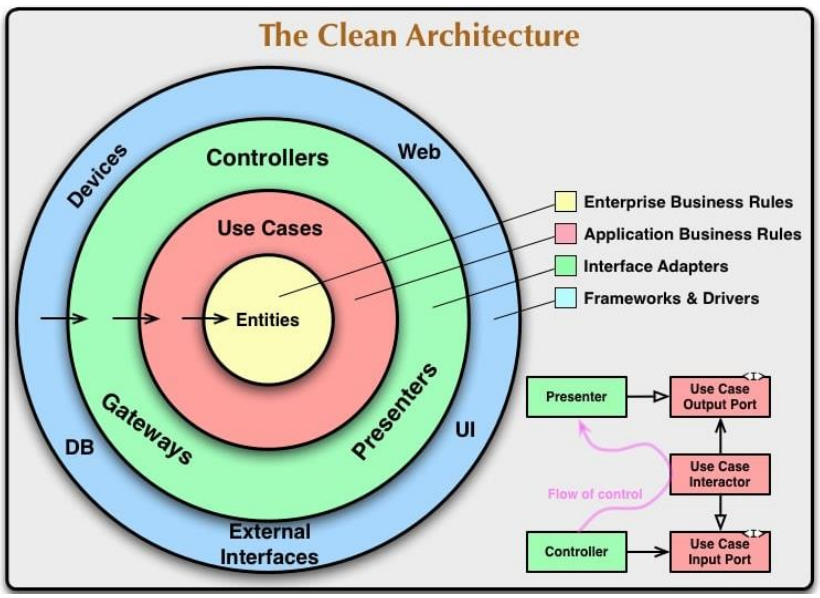


Рисунок 4.1 – Принципы Clean Architecture приложения [22]

Архитектурная организация системы основана на принципах чистой архитектуры (англ. Clean Architecture), которая акцентирует внимание на разделении ответственности и слабой связанности компонентов (рисунок 4.1). Такой подход позволяет изолировать бизнес-логику от внешних зависимостей,

благодаря чему становится возможным тестирование, модификация и масштабирование отдельных частей приложения без необходимости переписывать всю систему целиком.

Во всех Python-модулях соблюдаются правила оформления кода в соответствии с официальным стилевым руководством Python – PEP 8 [23]. Это повышает читаемость и стандартизирует структуру проекта.

Проект полностью развивается под управлением распределенной системы контроля версий Git и размещен на платформе GitHub. Использование Git обеспечивает хранение полной истории изменений кода, что упрощает откат к предыдущим версиям и восстановление после ошибок. На GitHub организована командная работа: разработчики создают отдельные ветки для новых функций, после чего через Pull Request проводится код-ревью. Это помогает отлавливать ошибки и улучшать качество кода до слияния в основную ветку.

GitHub также предоставляет средства для совместной работы: ведение задач, вики, документация и управление релизами. Благодаря этому команда может эффективно координировать работу, распределять задачи и отслеживать прогресс. Централизованное хранилище кода гарантирует, что все участники работают с одной и той же актуальной версией проекта, а истории коммитов достаточно для подробного анализа любых изменений.

В серверной части настроены процессы непрерывной интеграции и доставки (CI/CD) с помощью GitHub Actions [24]. При каждом коммите в репозиторий автоматически запускается конвейер сборки, который последовательно выполняет следующие шаги:

– проверка и форматирование кода;

– запуск тестов.

Такой CI/CD-подход позволяет мгновенно выявлять ошибки еще до ручного тестирования и гарантирует, что каждый коммит может быть развернут. Автоматизация проверки качества кода и развертывания ускоряет выпуск новых версий приложения и снижает риск человеческих ошибок. Кроме того, интеграция CI/CD с GitHub обеспечивает прозрачность процессов: при неудачных проверках команда получает оперативные уведомления о причинах сбоев.

Для управления изменениями схемы базы данных PostgreSQL используется система миграций Alembic [25]. При модификации моделей данных в коде Alembic автоматически генерирует скрипты миграций, которые применяют соответствующие изменения в структуре базы. Это гарантирует синхронизацию между моделью данных в приложении и фактической схемой БД. Основные преимущества применения Alembic:

– версионирование схемы;

– упрощение развертывания;

– возможности откатов изменений.

Использование Alembic упрощает поддержку и развитие базы данных в команде, исключает рассинхронизацию между кодом и данными, и повышает надежность процесса обновления.

Для обеспечения единого стиля и минимизации технического долга применяются следующие инструменты:

– isort [26] – для автоматической сортировки импортов;

– black [27] – для автоформатирования кода;

– pre-commit [28] – для запуска линтеров и автоформатирования при каждом коммите.

Конфигурация этих инструментов хранится в файле `pyproject.toml`:

```
[tool.black]
line-length = 120

[tool.isort]
multi_line_output = 3
include_trailing_comma = true
```

Здесь `line-length = 120` задает максимальную длину строки, а параметры `multi_line_output` и `include_trailing_comma` контролируют форматирование длинных импортов. Автоформатирование гарантирует единый стиль кода во всем проекте, устраняет споры по оформлению и облегчает код-ревью. Благодаря этому разработчики могут сосредоточиться на логике, а не на синтаксисе. Кроме того, единственные правила форматирования в одном файле конфигурации упрощают настройку среды разработки.

Использование всех вышеперечисленных подходов и инструментов способствует созданию устойчивой, поддерживаемой и масштабируемой программной системы, соответствующей современным стандартам индустрии.

### 4.2 Реализация модели нейронной сети

Алгоритм обучения модели приведен на чертеже ГУИР.400201.042 ПД.

**4.2.1** Класс `DenseMotionNetwork` реализует преобразование разреженных движений, заданных ключевыми точками, в плотное поле деформации изображения.

```
class DenseMotionNetwork(nn.Module):
    def __init__(
        self,
        block_expansion,
        num_blocks,
        max_features,
        num_kp,
        num_channels,
        estimate_occlusion_map=False,
        scale_factor=1,
        kp_variance=0.01
```

```
    ):
        ...
        in_features = (num_kp + 1) * (num_channels + 1)
        self.hourglass = Hourglass(
            block_expansion,
            in_features,
            num_blocks,
            max_features
        )
        self.mask = nn.Conv2d(
            self.hourglass.out_filters,
            num_kp + 1,
            kernel_size=7,
            padding=3
        )
        self.occlusion = (
            nn.Conv2d(...)
            if estimate_occlusion_map else None
        )
        self.down = AntiAliasInterpolation2d(
            num_channels, scale_factor
        ) if scale_factor != 1 else None
```

На этапе инициализации создаются:

– `hourglass` – сверточная сеть, извлекающая признаки из изображения;

– `mask` – слой, предсказывающий маску вкладов от каждой ключевой точки и фона;

– `occlusion` – выход, предсказывающий карту окклюзий;

– `down` – слой антиалиасинга для уменьшения разрешения входа.

Для создания гауссовых тепловых карт на основе координат ключевых точек применяется метод:

```
def create_heatmap_representations(
    self, source_image, kp_driving, kp_source
):
    ...
    gaussian_driving = kp2gaussian(kp_driving, ...)
    gaussian_source = kp2gaussian(kp_source, ...)
    heatmaps = torch.cat(
        [
            torch.zeros_like(gaussian_driving[:, :1]),
            gaussian_driving - gaussian_source
        ],
        dim=1
    )
    return heatmaps.unsqueeze(2)
```

Основная идея – получить разницу между положениями ключевых точек на управляющем и исходном изображениях.

Создается сетка координат, отражающая движение каждого пикселя, ассоциированного с ключевой точкой, из положения в `kp_driving` в `kp_source`:

```
def create_sparse_motions(
    self, source_image, kp_driving, kp_source
):
    ...
    identity_grid = make_coordinate_grid((h, w), ...)
    coordinate_grid = (
        identity_grid - kp_driving["value"].view(...)
    )
    if "jacobian" in kp_driving:
        jacobian = torch.matmul(
            kp_source["jacobian"],
            torch.inverse(kp_driving["jacobian"])
        )
        coordinate_grid = torch.einsum(
            "bkhwij,bkhwj->bkhwi",
            jacobian,
            coordinate_grid
        )
    driving_to_source = (
        coordinate_grid + kp_source["value"].view(...)
    )
    return torch.cat([
        identity_grid.expand(...), driving_to_source
    ], dim=1)
```

При наличии якобиана осуществляется дополнительное аффинное преобразование.

Изображение деформируется для каждой ключевой точки с использованием функций `grid_sample`:

```
def create_deformed_source_image(
    self, source_image, sparse_motions
):
    ...
    source_expanded = source_image.unsqueeze(1).expand(...)
    motions_flat = sparse_motions.view(...)
    deformed = F.grid_sample(
        source_expanded, motions_flat, align_corners=True
    )
    return deformed.view(...)
```

Полученные деформированные изображения будут использоваться далее при построении признаков.

На этапе прямого прохода все промежуточные представления (тепловые карты и деформированные изображения) объединяются и подаются в hourglass-сеть. Также предсказывается маска весов, определяющая, какие

области изображения и какие движения имеют наибольшее значение. Вычисляется итоговое плотное поле деформации. При необходимости генерируется карта окклюзий.

Далее представлен описанный метод:

```
def forward(self, source_image, kp_driving, kp_source):
    if self.down:
        source_image = self.down(source_image)

    heatmaps = self.create_heatmap_representations(
        source_image, kp_driving, kp_source
    )
    sparse_motions = self.create_sparse_motions(
        source_image, kp_driving, kp_source
    )
    deformed_source = self.create_deformed_source_image(
        source_image, sparse_motions
    )

    hourglass_input = torch.cat(
        [heatmaps, deformed_source],
        dim=2,
    ).view(bs, -1, h, w)
    prediction = self.hourglass(hourglass_input)
    mask = F.softmax(self.mask(prediction), dim=1)

    deformation = (
        sparse_motions.permute(0, 1, 4, 2, 3)
        * mask.unsqueeze(2)
    ).sum(dim=1)
    deformation = deformation.permute(0, 2, 3, 1)
```

**4.2.2** Компонент `KPDetector` отвечает за обнаружение ключевых точек на изображении, а также (опционально) за оценку якобиана в их окрестности, что позволяет учитывать локальные аффинные преобразования. Это важный этап при построении управляемой анимации объектов.

В качестве backbone-сети используется модуль `Hourglass`, извлекающий пространственные признаки из входного изображения. Далее применяется сверточный слой `kp`, формирующий тепловые карты для каждой ключевой точки:

```
class KPDetector(nn.Module):
    def __init__(
        ...,
        estimate_jacobian=False,
        single_jacobian_map=False,
        ...
    ):
        ...
        self.predictor = Hourglass(...)
```

```
        self.kp = nn.Conv2d(
            ...,
            out_channels=num_kp,
            kernel_size=7,
            padding=pad
        )
```

Если включена опция `estimate_jacobian`, дополнительно создается сверточный слой `jacobian`, который предсказывает для каждой ключевой точки 2×2 матрицу якобиана:

```
if estimate_jacobian:
    self.jacobian = nn.Conv2d(
        ..., out_channels=4 * self.num_jacobian_maps, ...
    )
    self.jacobian.weight.data.zero_()
    self.jacobian.bias.data.copy_(...)
```

При использовании `single_jacobian_map=True`, оценка выполняется глобально, а не отдельно для каждой точки. Для повышения стабильности обучения слой инициализируется как единичное преобразование.

При прямом проходе входное изображение, при необходимости, сначала уменьшается через антиалиасинг (`AntiAliasInterpolation2d`), после чего проходит через сверточный предсказатель признаков. Далее полученные карты пропускаются через слой `kp`, формируя необработанные тепловые карты ключевых точек:

```
def forward(self, x):
    if self.down:
        x = self.down(x)

    feature_map = self.predictor(x)
    heatmap = self.kp(feature_map)
```

Постобработка тепловых карт проводится следующим образом:

```
heatmap_flat = heatmap.view(B, K, -1)
heatmap = F.softmax(
    heatmap_flat / self.temperature, dim=2
).view(B, K, H, W)
out = gaussian2kp(heatmap)
```

Сформированные тепловые карты нормализуются функцией softmax по пространству (высота × ширина), что позволяет интерпретировать их как вероятностные распределения. Температура регулирует «резкость» распределения. После нормализации тепловые карты передаются в функцию `gaussian2kp`, которая извлекает координаты ключевых точек (взвешенное

среднее значений тепловой карты) и оформляет результат в словарь.

Оценка якобиана проводится следующим образом:

```
if self.jacobian:
    jacobian_map = self.jacobian(feature_map)
    jacobian_map = jacobian_map.view(
        B, self.num_jacobian_maps, 4, H, W
    )
    weighted = (
        heatmap.unsqueeze(2) * jacobian_map
    ).view(B, K, 4, -1)

    jacobian = weighted.sum(dim=-1).view(B, K, 2, 2)
    out["jacobian"] = jacobian
```

Предсказанная карта якобианов взвешивается с учетом значений тепловых карт, после чего по каждой карте берется сумма, представляющая финальную матрицу преобразования для соответствующей ключевой точки. Результат добавляется к словарю выхода под ключом «`jacobian`».

**4.2.3** Класс `OcclusionAwareGenerator` реализует генераторную сеть, которая преобразует исходное изображение в соответствии с движением, заданным положением ключевых точек. В отличие от классических архитектур, здесь дополнительно учитываются окклюзии, что позволяет более реалистично моделировать деформации объектов.

На входе инициализируется модуль `DenseMotionNetwork`, который отвечает за расчет поля деформаций и, при необходимости, карты окклюзий:

```
self.dense_motion_network = DenseMotionNetwork(...)
```

Он использует информацию о ключевых точках исходного и управляющего изображений для моделирования движения.

Архитектура сети симметрична и включает:

– сверточный блок, сохраняющий размерность изображения;

– энкодер;

– серии остаточных блоков (англ. bottleneck), обеспечивающие глубокую переработку признаков без потери информации;

– декодер;

– сверточный слой, возвращающий изображение с числом каналов, равным исходному.

Это описано в следующем коде:

```
self.first = SameBlock2d(...)
self.down_blocks = nn.ModuleList([...])
self.bottleneck = nn.Sequential([...])
self.up_blocks = nn.ModuleList([...])
self.final = nn.Conv2d(...)
```

Метод deform_input применяет поле деформации к признаковым картам или изображению, используя билинейную интерполяцию с функцией grid_sample:

```
def deform_input(inp, deformation):
    ...
    return F.grid_sample(inp,
        deformation, align_corners=True)
```

Это позволяет гибко изменять форму объекта на основе движения ключевых точек.

При прямом проходе сначала из исходного изображения извлекаются признаки с помощью серии сверточных блоков, последовательно уменьшая разрешение:

```
out = self.first(source_image)
for i in range(len(self.down_blocks)):
    out = self.down_blocks[i](out)
```

Далее на основе ключевых точек и изображения рассчитывается деформация, которую затем применяют к признаковым картам.

```
dense_motion = self.dense_motion_network(...)
...
out = self.deform_input(out, deformation)
```

Если включена оценка карты окклюзий, то она умножается на признаковые карты, подавляя области с отсутствующей информацией. Также дополнительно деформируется исходное изображение – этот результат сохраняется как «deformed».

Закодированные и трансформированные признаки проходят через остаточные блоки и восстанавливаются в изображение с помощью декодера:

```
out = self.bottleneck(out)
for up_block in self.up_blocks:
    out = up_block(out)
out = self.final(out)
out = F.sigmoid(out)
```

Финальный слой и sigmoid возвращают изображение с пикселями в диапазоне [0, 1]. Результат сохраняется в выходной словарь под ключом «prediction».

Метод forward возвращает словарь с несколькими ключами:

– «prediction» – итоговое изображение, сформированное генератором;

– «deformed» – исходное изображение, деформированное согласно движениям;

– «mask» и «sparse_deformed» – дополнительные данные от DenseMotionNetwork;

– «occlusion_map» – карта окклюзий, если используется.

**4.2.4** Класс MultiScaleDiscriminator – это дискриминаторная модель, предназначенная для обработки изображений на нескольких пространственных масштабах. Вместо того чтобы анализировать изображение только на одном масштабе, MultiScaleDiscriminator принимает входы на нескольких масштабах и пропускает каждую версию изображения через отдельный экземпляр базового дискриминатора Discriminator:

```
def __init__(self, scales=(), **kwargs):
```

Каждому масштабу сопоставляется своя независимая копия дискриминатора:

```
self.discs = nn.ModuleDict({...})
```

Метод forward принимает:

– x – словарь изображений разного масштаба;

– kp – ключевые точки.

Далее для каждого масштаба извлекается изображение по ключу «prediction_<scale>». Оно обрабатывается соответствующим Discriminator, после чего сохраняются карты признаков и карты предсказаний:

```
feature_maps, prediction_map = disc(x[key], kp)

out_dict[f"feature_maps_{scale}"] = feature_maps
out_dict[f"prediction_map_{scale}"] = prediction_map
```

В свою очередь класс Discriminator реализует патч-базовый дискриминатор, то есть модель, которая не принимает решение о «реальности» всего изображения целиком, а делает это локально по участкам. Такой подход получил широкое распространение в задачах, связанных с генерацией изображений, поскольку позволяет эффективно оценивать структурную достоверность на уровне текстур, краев и локальных паттернов.

Если включен режим use_kp, то перед подачей на сверточные слои изображение расширяется путем конкатенации с тепловыми картами, генерируемыми из координат ключевых точек с помощью функции kp2gaussian:

```
if self.use_kp:
    heatmap = kp2gaussian(kp, x.shape[2:], self.kp_variance)
    x = torch.cat([x, heatmap], dim=1)
```

Слои сверточной обработки реализованы через `DownBlock2d`:

```
for i in range(num_blocks):
    self.down_blocks.append(...)
```

Первый блок адаптирован под входное количество каналов (возможно расширенное за счет тепловых карт).

Финальный слой – это одномерная свертка с ядром 1×1, выдающая карту вероятностей. При включенной спектральной нормализации он оборачивается через `nn.utils.spectral_norm`:

```
self.conv = nn.utils.spectral_norm(conv) if sn else conv
```

### 4.3 Структура сервера на FastAPI

В рамках дипломного проекта была разработана серверная часть приложения на основе фреймворка FastAPI. Архитектура проекта построена с учетом принципов чистой архитектуры и разделения ответственности, что обеспечивает модульность, расширяемость и удобство поддержки кода. Структура проекта приведена далее:

```
Animatica-backend/
│
├── app/
│   ├── alembic/
│   ├── src/
│   ├── domain/
│   │   ├── entities/
│   │   ├── interfaces/
│   │   └── usecases/
│   │
│   ├── infrastructure/
│   │   ├── api/
│   │   │   └── v0/
│   │   ├── configs/
│   │   ├── exceptions/
│   │   ├── models/
│   │   ├── repositories/
│   │   ├── services/
│   │   └── main.py
│   ├── alembic.ini
│   └── README.md
│
└── env/
```

Описание компонентов:

– `domain/entities/` содержит основные бизнес-сущности, которые описывают предметную область приложения;

– `domain/interfaces/` включает абстракции и интерфейсы для инверсии зависимостей, что позволяет отделить логику от конкретных

реализаций;

– `domain/usecases/` реализует бизнес-логику и сценарии использования, обеспечивая последовательность действий и правил приложения;

– `infrastructure/api/v0/` реализован REST API с использованием FastAPI, здесь находятся маршруты и контроллеры для обработки HTTP-запросов;

– `infrastructure/configs/` хранит настройки приложения, включая параметры подключения к базе данных и внешним сервисам;

– `infrastructure/exceptions/` определяет собственные классы исключений для более точной обработки ошибок;

– `infrastructure/models/` содержит ORM-модели, которые отражают структуру базы данных;

– `infrastructure/repositories/` реализует логику доступа к данным, инкапсулируя операции с базой;

– `infrastructure/services/` предоставляет сервисы для работы с бизнес–логикой и взаимодействия с репозиториями;

– `main.py` – точка запуска приложения, где инициализируется FastAPI, подключаются маршруты и запускается сервер.

Такой подход к организации проекта обеспечивает четкое разделение уровней ответственности и упрощает сопровождение кода, а также тестирование отдельных модулей.

### 4.4 Описание страниц пользовательского интерфейса

Пользовательский интерфейс построен на основе многостраничной архитектуры: каждая страница отвечает за отдельный компонент пользовательского взаимодействия. Ниже приведен основной код, регистрирующий страницы приложения:

```
auth_page = st.Page(
    page="views/auth.py",
    title="Authentication",
    icon=":material/login:",
)

home_page = st.Page(
    page="views/home.py",
    title="Home",
    icon=":material/home:",
)

about_page = st.Page(
    page="views/about.py",
    title="About",
    icon=":material/info:",
)
```

```
pricing_page = st.Page(
    page="views/pricing.py",
    title="Pricing",
    icon=":material/payments:",
)
```

Каждая страница создается с помощью объекта `st.Page`, в котором указываются:

– путь к файлу с реализацией логики страницы (`page`);

– заголовок страницы (`title`);

– иконка страницы (`icon`).

Таким образом, обеспечивается четкая структура пользовательского интерфейса, а навигация между страницами становится удобной и понятной. Streamlit автоматически подгружает содержимое каждой страницы на основе заданного пути и отображает ее в меню навигации.

**4.4.1** Страница аутентификации предназначена для авторизации и регистрации пользователей. Она визуально разделена на две колонки – форма входа и форма регистрации. Это реализовано с помощью стандартного компонента `st.columns`.

В начале модуля подключаются необходимые ресурсы:

```
ResourceService.load_styles("auth.css")

templates = {
    "main_container":
ResourceService.load_template("auth/main_container.html"),
    "auth_header":
ResourceService.load_template("auth/header.html"),
    "login_column":
ResourceService.load_template("auth/login_column.html"),
    "register_column":
ResourceService.load_template("auth/register_column.html"),
}
```

Стили и HTML-шаблоны загружаются с помощью сервисного класса `ResourceService`, что позволяет отделить внешний вид интерфейса от логики.

Перед отображением форм выполняется проверка, авторизован ли пользователь:

```
if st.session_state.get("logged_in", False):
    st.success(
        f"Welcome back, {st.session_state.username}!"
    )
    st.rerun()
```

Если пользователь уже вошел в систему, отображается приветствие и

происходит перезапуск интерфейса.

Основная разметка страницы представлена с помощью HTML-шаблонов:

```
st.markdown(
    templates["main_container"], unsafe_allow_html=True
)
st.markdown(
    templates["auth_header"], unsafe_allow_html=True
)
```

Затем реализуется разделение на две колонки:

```
col1, col2 = st.columns(2)
with col1:
    render.render_login_form()
with col2:
    render.render_register_form()
```

Объект `render` класса `AuthRender` отвечает за отрисовку форм. Он инкапсулирует соответствующую логику и шаблоны, обеспечивая чистоту основного кода.

В завершение, закрывается контейнерный HTML-блок:

```
st.markdown("</div>", unsafe_allow_html=True)
```

**4.4.2** Главная страница пользовательского интерфейса предназначена для загрузки исходных изображений и видео, а также запуска генерации анимации. Она содержит визуальные и логические блоки, отвечающие за: проверку подписки пользователя, отображение форм загрузки изображения и видео, запуск генерации и отображение результата с возможностью загрузки.

В начале страницы подключаются модули рендеринга и сервисы и загружаются необходимые CSS-стили и HTML-шаблоны:

```
ResourceService.load_styles("home.css")

templates = {
    "subscription": ResourceService.load_template(
        "home/subscription_required.html"
    ),
    "driving_video": ResourceService.load_template(
        "home/driving_video.html"
    ),
    "header": ResourceService.load_template(
        "home/generation_header.html"
    ),
}
```

Перед тем как отобразить интерфейс генерации, производится проверка

состояния подписки пользователя:

```
query_balance = CookieService.controller.get(
    "query_balance"
)
if (query_balance is None or query_balance == 0)
    and not st.session_state.animation_result:
    render.render_subscription_required()
```

Если у пользователя нет активной подписки и не был ранее получен результат генерации, отображается шаблон с уведомлением о необходимости оформить подписку.

При наличии подписки или сгенерированного результата, пользователю отображается основной интерфейс (сначала – заголовок страницы):

```
st.markdown(templates["header"], unsafe_allow_html=True)
```

Затем реализуется двухколоночная структура:

```
col1, col2 = st.columns(2)
with col1:
    render.render_image_upload()
with col2:
    render.render_video_upload()
```

В первой колонке осуществляется загрузка изображения, во второй – загрузка видео. Эти функции инкапсулированы в методы `render_image_upload` и `render_video_upload` соответственно.

Ниже размещена кнопка запуска генерации:

```
st.markdown("---")
render.render_generate_button()
```

Если в сессии уже хранится результат генерации, он будет отображен в виде видео и доступен для загрузки:

```
if st.session_state.animation_result:
    st.markdown(templates["header"], unsafe_allow_html=True)
    st.video(st.session_state.animation_result)
    render.render_download_button()
```

**4.4.3** Страница оплаты «Pricing» отвечает за отображение и выбор тарифных планов пользователем. Она содержит визуальные элементы, формирующие карточки подписок, и обеспечивает переход к оформлению подписки. Основной акцент сделан на визуальную структуру, читаемость и взаимодействие с бэкендом через классы-рендеры и сервисы.

В начале модуля подключаются необходимые классы и стили, затем

загружаются CSS-стили и HTML-шаблоны:

```
ResourceService.load_styles("pricing.css")
templates = {
    "header": ResourceService.load_template(
        "pricing/header.html"
    ),
    "plan_card": ResourceService.load_template(
        "pricing/plan_card.html"
    ),
    "selection": ResourceService.load_template(
        "pricing/selection_message.html"
    ),
    "checkout_button": ResourceService.load_template(
        "pricing/checkout_button.html"
    ),
}
```

В верхней части страницы размещается заголовок, определенный в отдельном HTML-шаблоне:

```
st.markdown(templates["header"], unsafe_allow_html=True)
```

Для управления отображением тарифных планов используется объект `PricingRender`, который обрабатывает список продуктов и их визуализацию:

```
render = PricingRender(templates)
```

Если список тарифных планов не загружен, пользователю выводится предупреждение, после чего выполняется выход из учетной записи:

```
if not render.products:
    st.warning("No subscription plans available.")
    time.sleep(3)
    AuthService.logout()
```

Карточки подписок отображаются в виде горизонтального ряда, каждая – в отдельной колонке:

```
with st.container():
    cols = st.columns(len(render.products))
    for col, product in zip(cols, render.products):
        render.render_plan_card(product, col)
```

Каждый продукт из списка `render.products` визуализируется с помощью метода `render_plan_card`, который использует соответствующий HTML-шаблон.

Если пользователь выбрал подписку, соответствующая информация берется из состояния сессии, и отображается кнопка для перехода к оформлению:

```
if st.session_state.selected_product_id
    and st.session_state.selected_price_id:
    selected_product = next((
        p for p in render.products
        if p["id"] == st.session_state.selected_product_id
    ), None)
    if selected_product:
        render.render_subscribe_button(selected_product)
```

**4.4.4** Информационная страница «About» содержит сведения о разработчике проекта и служит информационным блоком интерфейса. Она оформлена в едином стиле с остальными страницами приложения и использует шаблонный подход для отображения контента.

В начале скрипта подключаются необходимые библиотеки и сервисы. Далее формируется контекст – словарь, содержащий данные, подставляемые в HTML-шаблон страницы. В него включены ссылки на контактные данные, имя разработчика и краткая информация о нем:

```
context = {
    "image_base64": ResourceService.load_image(
        "animatica_about.svg"
    ),
    "email_link": "mailto:prostolex2004@mail.ru",
    "telegram_link": "https://t.me/keffirchk",
    "linkedin_link":    "https://www.linkedin.com/in/alexey-
klimovich-30744b249/",
    "developer_name": "Klimovich Alexey",
    "developer_info":  "4th-year  Computer  Science  student
specializing in Computing Machines, Systems, and Networks",
}
```

Затем выполняется загрузка CSS-стилей и HTML-шаблона, отвечающих за визуальное представление:

```
ResourceService.load_styles("about.css")
template = ResourceService.load_template("about/about.html")
```

Шаблон содержит HTML-разметку, в которую автоматически подставляются значения из словаря `context`.

С помощью метода `st.markdown` происходит отрисовка страницы:

```
st.markdown(
    template.format(**context), unsafe_allow_html=True
)
```

# 5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

В данном разделе рассмотрено тестирование разработанной системы, включающей в себя нейросетевую модель анимации изображений, backend-сервер и веб-интерфейс для взаимодействия с пользователем. Проведение испытаний является важным этапом разработки, позволяющим выявить ошибки до ввода системы в эксплуатацию, проверить соответствие реализованного функционала заявленным требованиям, а также оценить производительность, устойчивость и удобство использования программного продукта.

## 5.1 Конфигурация тестовой среды

Для проведения испытаний разработанного программного средства была подготовлена тестовая среда, включающая программные и аппаратные компоненты, обеспечивающие корректную работу всех модулей системы. Конфигурация этой среды представлена в таблицах 5.1 и 5.2.

Таблица 5.1 – Аппаратная конфигурация тестовой среды

| Компонент | Описание |
|---|---|
| Процессор | Intel(R) Core (TM) i5-1036G1 |
| Оперативная память | 8 ГБ DDR4 |
| Видеокарта (дискретная) | Нет |
| Сетевое подключение | Прием – 115 Мбит/с, передача – 72 Мбит/с |

Таблица 5.2 – Программная конфигурация тестовой среды

| Компонент | Версия |
|---|---|
| Операционная система | Windows 10 Домашняя |
| Python | 3.11 |
| FastAPI | 0.115.12 |
| Streamlit | 1.44.1 |
| PyTorch | 2.6.0 |
| Docker Desktop | 4.40.0 |
| Docker Engine | 28.0.4 |
| Docker Compose | v2.34.0-desktop.1 |
| PostgreSQL | 15-alpine |
| Stripe | 12.0.0 |
| Браузер | Google Chrome 136.0.7103.114 |

## 5.1 Тестирование модели нейронной сети

Для обеспечения корректной работы модели анимации изображений была проведена ручная проверка единственного эндпоинта API, реализующего функциональность генерации видео на основе статического

изображения и управляющего видеофайла. Тестирование проводилось в ручном режиме с использованием инструмента Swagger UI, автоматически сгенерированного на основе схем FastAPI (рисунок 5.1).


Рисунок 5.1 – Документация сервиса ML-модели от Swagger

**5.1.1** Модель была развернута в среде FastAPI с доступом к POST-эндпоинту `/api/video`.

В ходе тестирования был отправлен POST-запрос с двумя файлами:

– изображение формата PNG размером 164 КБ;

– видеофайл формата MP4 размером 14,9 МБ.

После успешной обработки запроса сервер возвратил потоковое видео, сформированное на основе переданных данных. Время обработки запроса составило 7 минут 17,262 секунды.

Была подтверждена корректная работа механизма генерации, отсутствие внутренних исключений и корректное освобождение временных файлов после завершения процесса.

**5.1.2** С целью ограничения доступа к модели реализована проверка API-ключа через специальный HTTP-заголовок `X-ML-Engine-Key`. Для проверки этого механизма были проведены тесты, приведенные в таблице 5.3.

Таблица 5.3 – Тестовые сценарии для проверки аутентификации ML-модели

| Сценарий | Ожидаемый результат | Полученный результат |
|---|---|---|
| В заголовках указан корректный ключ доступа к API | Доступ к эндпоинту предоставляется, результат генерируется | Доступ к эндпоинту предоставляется, результат генерируется |
| Отсутствует заголовок или некорректный ключ доступа | Возвращается ошибка с кодом 403 Forbidden с сообщением об ошибке авторизации | Возвращается ошибка с кодом 403 Forbidden с сообщением об ошибке авторизации |

**5.1.3** Для объективной оценки качества обученной модели был проведен тест в режиме реконструкции, при котором в качестве источника и управляющего видео использовался один и тот же видеоряд. Это позволяет модели воспроизвести исходный видеопоток и измерить точность восстановления. В рамках этого этапа были рассчитаны следующие метрики:

– L1 – отражает степень расхождения между оригинальными и сгенерированными изображениями по пикселям;

– AKD (англ. Average Keypoint Distance) – оценивает точность восстановления положения ключевых точек;

– AED (англ. Average Euclidean Distance) – позволяет определить визуальное сходство анимации с оригиналом на основе расстояний в пространстве признаков.

Полученные значения метрик позволяют судить о степени реалистичности, стабильности и точности работы нейронной сети (таблица 5.4).

Таблица 5.4 – Полученные метрики ML-модели

| Метрика | Значение |
|---|---|
| L1 | 0,043 |
| AKD | 1,294 |
| AED | 0,140 |

Тестирование показало, что модель стабильно обрабатывает запросы, корректно работает с API-защитой и способна генерировать реалистичные анимации. Режим реконструкции подтвердил функциональную состоятельность и допустимый уровень искажения.

## 5.2 Тестирование backend-сервера

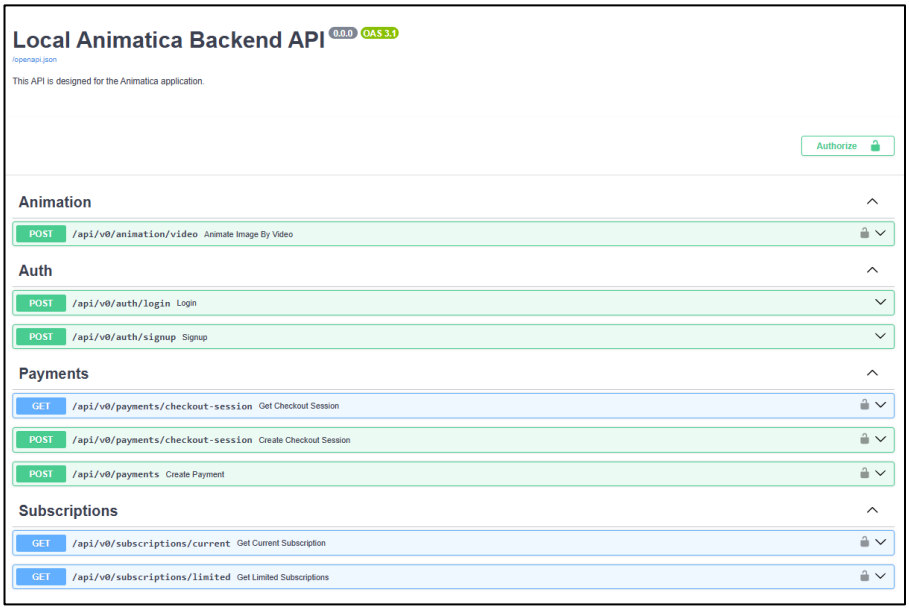


Рисунок 5.2 – Документация backend-сервера от Swagger

Backend-сервер системы реализует REST API с использованием фреймворка FastAPI. Для тестирования API-интерфейса применялся встроенный инструмент Swagger UI, доступный по адресу `/docs`, что обеспечило удобное ручное тестирование каждого маршрута (рисунок 5.2). Описание маршрутов приведено в таблице 5.5.

Таблица 5.5 – Описание маршрутов backend-сервера

| Номер тестирования | Группа | Маршрут | Задачи |
|---|---|---|---|
| 1 | Animation | `/api/v0/animation/video` | Генерация анимации |
| 2 | Auth | `/api/v0/auth/login` | Вход в аккаунт |
| 3 | | `/api/v0/auth/signup` | Регистрация пользователя |
| 4 | Payments | `/api/v0/payments/checkout-session` | Получении сессии оплаты |
| 5 | | `/api/v0/payments/checkout-session` | Создание сессии оплаты |
| 6 | | `/api/v0/payments` | Подтверждение оплаты |
| 7 | Subscriptions | `/api/v0/subscriptions/current` | Получение подписки пользователя |
| 8 | | `/api/v0/subscriptions/limited` | Получение всех подписок |

Для большинства маршрутов были проведены тесты, приведенные в таблице 5.6.

Таблица 5.6 – Тестирование маршрутов backend-сервера

| Номер тестирования | Сценарий | Ожидаемый результат | Полученный результат |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 1 | Загружено изображение и видео, пользователь авторизован | Ответ с кодом 200, содержащий потоковое видео | Соответствует ожидаемому результату |
| 2 | Введены некорректные данные | Ответ с кодом 401 и сообщением «Invalid credentials» | Соответствует ожидаемому результату |
| | Введены корректные данные | Ответ с кодом 200, содержащий информацию о JWT-токенах | Соответствует ожидаемому результату |
| 3 | Введены корректные данные | Ответ с кодом 200 | Соответствует ожидаемому результату |
| | Введены данные с именем пользователя, который уже есть в системе | Ответ с кодом 409 и сообщением «User already exists: Username 'user' already exists» | Соответствует ожидаемому результату |

Продолжение таблицы 5.6

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 3 | Введенный пароль меньше 8 символов | Ответ с кодом 422 и сообщением «Value should have at least 8 items…» | Соответствует ожидаемому результату |
| 7 | Пользователь авторизован | Ответ с кодом 200, содержащий с информацию о подписке | Соответствует ожидаемому результату |
| | Пользователь не авторизован | Ответ с кодом 403 | Соответствует ожидаемому результату |
| 8 | Пользователь авторизован | Ответ с кодом 200, содержащий информацию о доступных подписках | Соответствует ожидаемому результату |
| | Пользователь не авторизован | Ответ с кодом 403 | Соответствует ожидаемому результату |

Маршруты из группы «Payments» были протестированы через пользовательский интерфейс.

### 5.3 Тестирование проекта через пользовательский интерфейс

Для обеспечения удобства использования и верификации бизнес-логики, проект включает веб-интерфейс, разработанный с использованием современных frontend-технологий. В процессе тестирования пользовательского интерфейса была выполнена проверка всех ключевых сценариев использования приложения с точки зрения конечного пользователя.

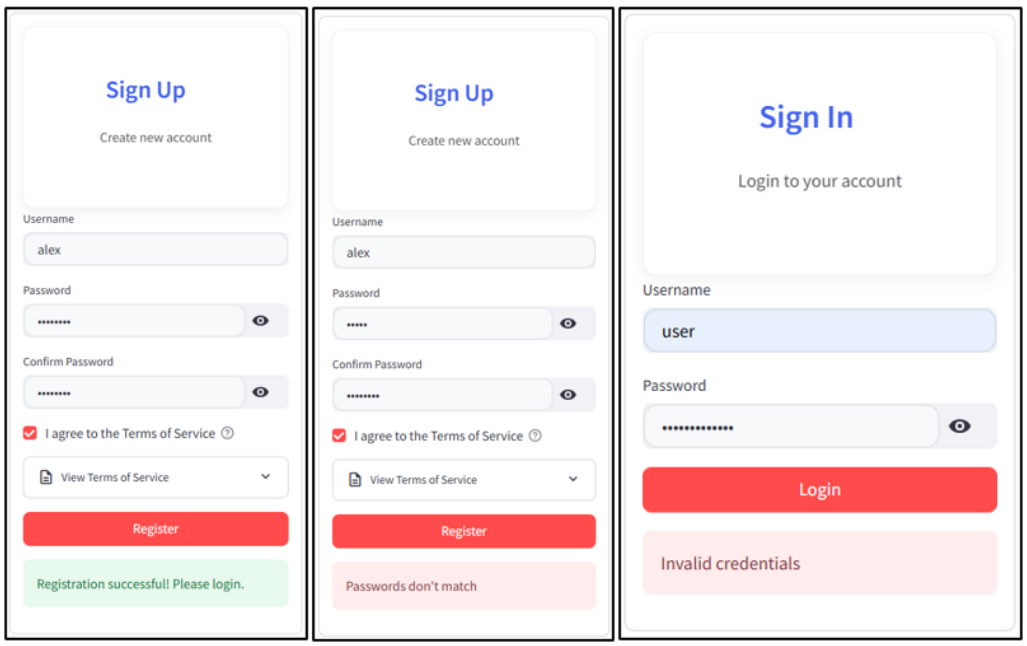


Рисунок 5.3 – Тестирование регистрации и авторизации через пользовательский интерфейс

Сначала была протестирована регистрация и авторизация пользователя с корректными и некорректными данными (рисунок 5.3).

После успешного входа в аккаунт интерфейс позволяет пользователю загрузить необходимые медиафайлы для анимации, если на балансе достаточно попыток генерации. При попытке использования анимации без активной подписки интерфейс корректно перенаправляет пользователя к оплате.

Проверено ограничение по форматам файлов (рисунок 5.4) и максимальный объем файла (рисунок 5.5).

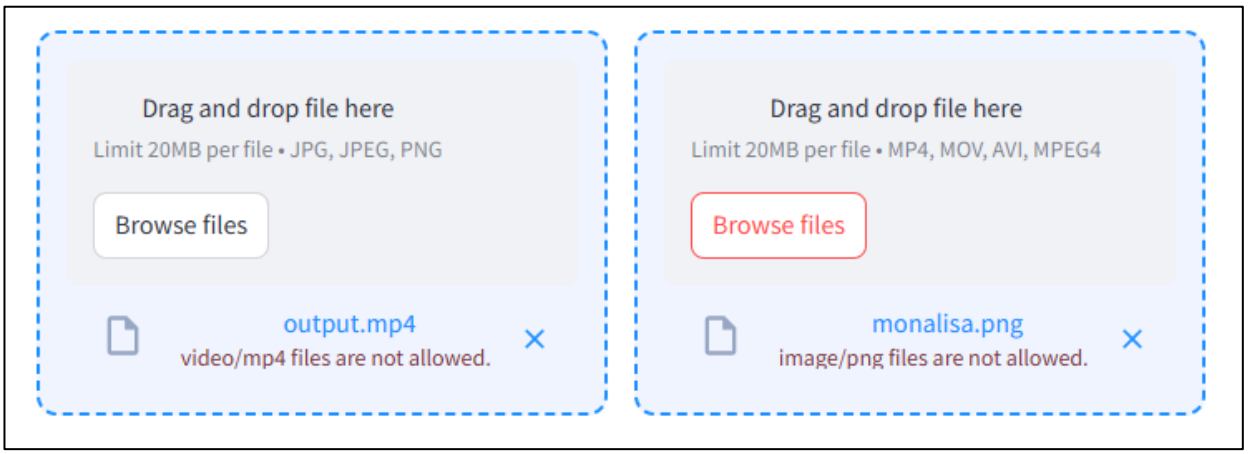


Рисунок 5.4 – Тестирование загрузки файлов неправильных форматов

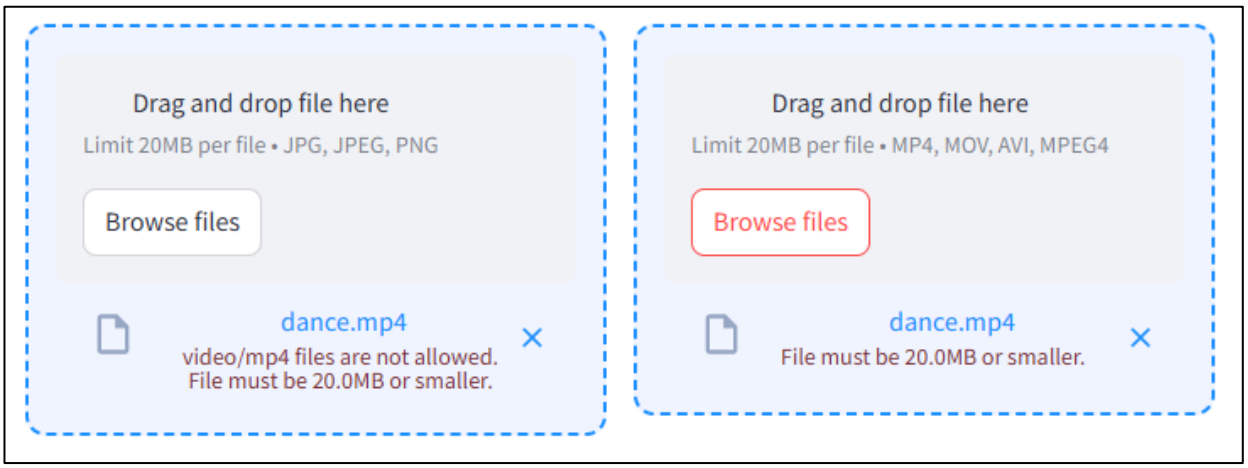


Рисунок 5.5 – Тестирование загрузки слишком больших файлов

После загрузки данных и нажатия кнопки запуска анимации, frontend отправляет запрос к backend-эндпоинту `/api/v0/animation/video`.

Далее проверен успешный запуск генерации, корректная работа индикатора загрузки, и возврат результата в виде видеофайла. Результат отображается в виде встроенного видеоплеера с возможностью повторного воспроизведения и скачивания.

Также проверено отображение информации о текущей подписке,

ограничениях и возможности перехода к оплате. Тестировалась интеграция с платежным шлюзом и возврат обратно на сайт с подтверждением успешной оплаты или отмены оплаты (рисунок 5.6).

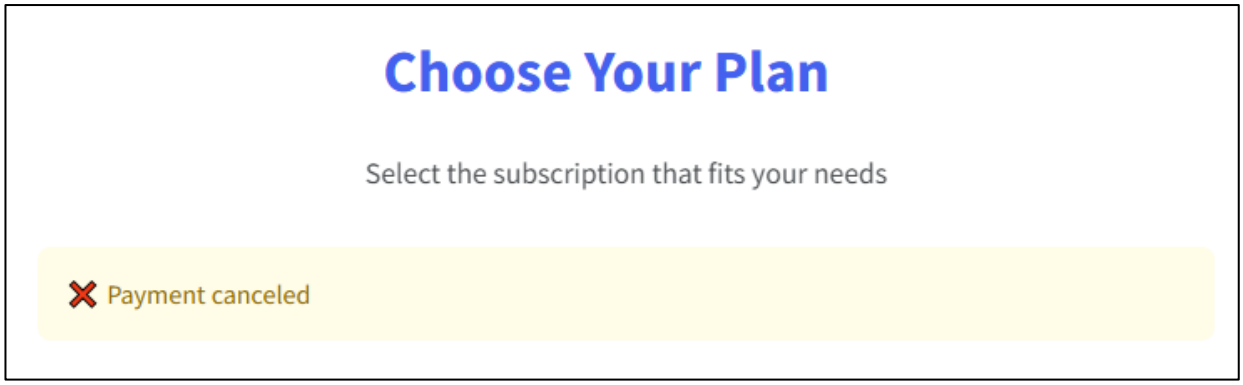


Рисунок 5.6 – Отклонение оплаты

Также были симулированы отключения сети. Тогда в случае со страницей планов подписок будет отображаться соответствующее сообщение (рисунок 5.7).

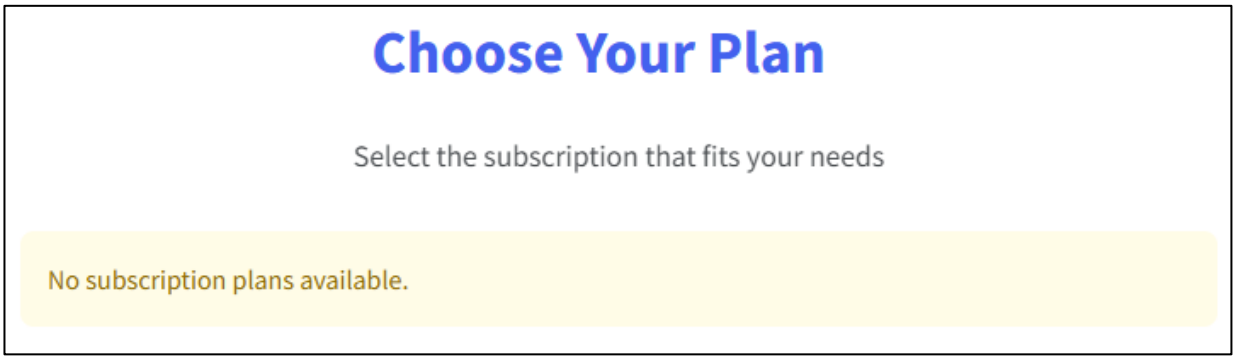


Рисунок 5.7 – Обработка ошибки получения планов подписок

Таким образом, интерфейс корректно информирует пользователя о сбоях, не «зависает» и не вызывает неотработанных исключений.

# 6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## 6.1 Системные требования

Для корректной работы разработанного программного средства необходимо обеспечить соответствующие технические и программные параметры как для backend-сервера, так и для клиентской части.

Минимальные технические характеристики сервера (FastAPI и модель FOMM):

– операционная система: Ubuntu 20.04+ / Windows 10/11 (64–разрядная версия);

– процессор: Intel Core i5 (4 ядра) или AMD Ryzen 5 (аналогичный);

– оперативная память: минимум 8 ГБ, рекомендуется 16 ГБ и более;

– GPU (опционально): NVIDIA GeForce RTX 1080 или выше;

– свободное место на диске: от 20 ГБ;

– Python: версия 3.11 или выше;

– Docker: для развертывания в изолированном окружении;

– СУБД: PostgreSQL.

Минимальное программное обеспечение клиента (веб-приложение на Streamlit и Stripe):

– операционная система: Windows 10/11, Linux, macOS;

– браузер: Google Chrome 100+, Mozilla Firefox 100+, Microsoft Edge 100+;

– интернет–соединение: скорость ≥ 10 Мбит/с;

– экран: разрешение от 1280×720, рекомендуется Full HD и выше;

– Stripe (оплата): поддержка HTTPS, cookie.

## 6.2 Запуск проекта

**6.2.1** Для запуска и тестирования модели нейронной сети локально необходимо настроить виртуальное окружение. Такое окружение можно создать командой (в случае запуска на операционной системе (ОС) типа Linux/MacOS):

```
python3 –m venv venv
```

Следующей командой активируем окружение:

```
source venv/bin/activate
```

В случае запуска проекта на ОС Windows последовательность действий схожа:

```
python –m venv venv

source venv/Scripts/activate
```

Далее необходимо установить необходимые зависимости и библиотеки, которые используются в модели нейронной сети. Для этого в проекте подготовлен файл `requirements.txt`. Выполним команду:

```
pip install -r app/requirements.txt
```

В модели используются некоторые переменные окружения, которые нужно задать в файле `env/ml_engine.env`:

```
ML_ENGINE_KEY=your-secret-key
API_MODE=local
LOG_LEVEL=debug
```

Далее необходимо поместить веса с обученной моделью в каталог `app/data/checkpoints/`.

Как отмечалось в подразделе 3.8, запуск модели можно выполнять несколькими способами. Для локального тестирования можно воспользоваться терминалом с командной строкой, где возможны следующие команды:

```
python run_model.py --mode train --configs config.yaml
python run_model.py --mode reconstruction --configs config.yaml --checkpoint path/to/ckpt
python run_model.py --mode animate --configs config.yaml --checkpoint path/to/ckpt
```

Во всех случаях необходимо указывать путь к конфигурационному файлу. Первая команда служит для запуска модели в режиме обучения, вторая – в режиме реконструкции, третья – в режиме анимации. Во втором и третьем случае кроме указания пути к конфигурационному файлу, необходимо указать путь к весам модели.

Для запуска модели как сервиса на FastAPI необходимо перейти в каталог `app/` и воспользоваться командой:

```
uvicorn src.run_server:app --host 0.0.0.0 --port 90 --reload
```

Таким образом сервис с моделью запуститься на локальной машине и будет доступен по адресу `http://localhost:90`. По адресу `http://localhost:90/docs` можно получить документацию о API для сервера с ML-моделью (рисунок 5.1).

Кроме перечисленных способов запуска, проект имеет `Dockerfile`, что позволяет создать образ с сервером ML-модели и запустить его независимо в Docker-контейнере. Для сборки образа необходимо воспользоваться командой:

```
docker build . --tag animatica-ml-engine
```

Следующей командой создается и запускается контейнер с именем «ml-engine» на основе только что созданного образа:

```
docker run --name ml-engine -p 9080:90 animatica-ml-engine
```

Теперь сервер с ML-моделью будет доступен по адресу `http://localhost:9080`.

**6.2.2** Для запуска сервера для данного проекта необходимо, как и в случае с ML-моделью, подготовить виртуальное окружение. Это делается аналогично (Linux/MacOS):

```
python3 -m venv venv
source venv/bin/activate
```

Или в случае запуска на Windows:

```
python -m venv venv
source venv/Scripts/activate
```

Далее устанавливаются необходимые зависимости:

```
pip install -r app/requirements.txt
```

После этого необходимо настроить переменные окружения.
Файл `env/api.env`:

```
API_BASE_URL=http://localhost:8080
API_MODE=local
LOG_LEVEL=debug
FRONTEND_BASE_URL=http://localhost:8501
SESSION_SECRET_KEY=your_session_secret_key
LOCALHOST_CLIENT_ORIGIN=http://localhost:5173
ALLOWED_ORIGINS=localhost
```

Файл `env/auth.env`:

```
SECRET_KEY=your-secret-key
ALGORITHM=HS256
ACCESS_TOKEN_EXPIRE_MINUTES=30
REFRESH_TOKEN_EXPIRE_DAYS=30
TOKEN_ISSUER=Animatica
```

Файл `env/db.env`:

```
DB_HOST=db
DB_PORT=5432
DB_NAME=animatica
```

```
DB_USER=postgres
DB_PASS=postgres
```

Файл `env/ml-engine.env`:

```
ML_ENGINE_BASE_URL=http://ml-engine:90
ML_ENGINE_KEY=your-secret-key
ML_ENGINE_KEY_HEADER=X-ML-Engine-Key
```

Файл `env/stripe.env`:

```
PUBLIC_KEY=your-public-key
SECRET_KEY=your-secret-key
```

Запустить работоспособный сервер с помощью uvicorn, как для ML-модели, не получится, так как в данном случае присутствует зависимость в видео БД и самой ML-модели. Поэтому для удобства пользования был написан файл `docker-compose.yml`, с помощью которого запускается главный сервер, база данных и сервис с ML-моделью. Выполним команду:

```
docker-compose up -d --build
```

Завершить работу запущенных контейнеров можно командой:

```
docker-compsose down
```

**6.2.3** Последний компонент системы – пользовательский интерфейс, настройка которого производится аналогично предыдущим компонентам: создание виртуального окружение, установка зависимостей, создание env-файлов.

Для данного компонента создается один env-файл с переменными окружения – `env/api.env`:

```
API_URL=http://localhost:8080
TERMS_OF_SERVICE_URL=https://example.com/terms
```

Перейдем в каталог `app/` и запустим модуль пользовательского интерфейса командой:

```
streamlit run src/main.py
```

По умолчанию запущенное веб-приложение будет доступно по адресу `http://localhost:8501`.

Также предусмотрен `Dockerfile`, с помощью которого можно собрать соответствующий образ и создать контейнер с пользовательским интерфейсом. Для этого применяются следующие команды:

```
docker build -f Dockerfile -t animatica-frontend .
docker run -p 8501:8501 --name ui animatica-frontend
```

### 6.3 Инструкция пользования

При переходе на веб-сайт приложения пользователь попадает на страницу регистрации/входа в аккаунт (рисунок 6.1).

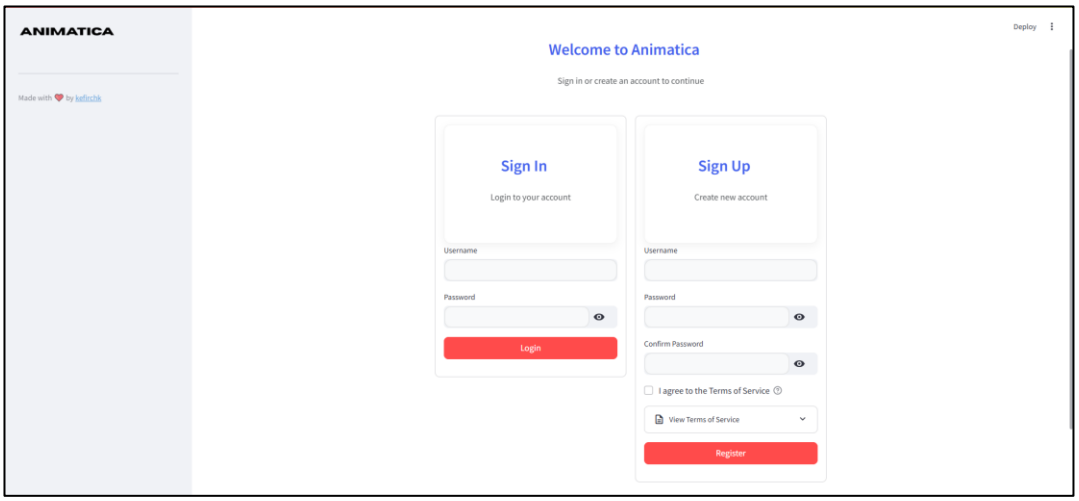


Рисунок 6.1 – Страница регистрации/входа в аккаунт

Здесь пользователю необходимо предварительно зарегистрироваться, указав имя пользователя и пароль. После подтверждения пароля и при согласии с условиями пользования данного веб-приложения будет создан аккаунт пользователя. Это значит, что теперь можно ввести свои данные в окно входа в аккаунт и войти в систему.

После этого пользователь попадает на главную страницу, где производится генерация анимации изображений (рисунок 6.2).

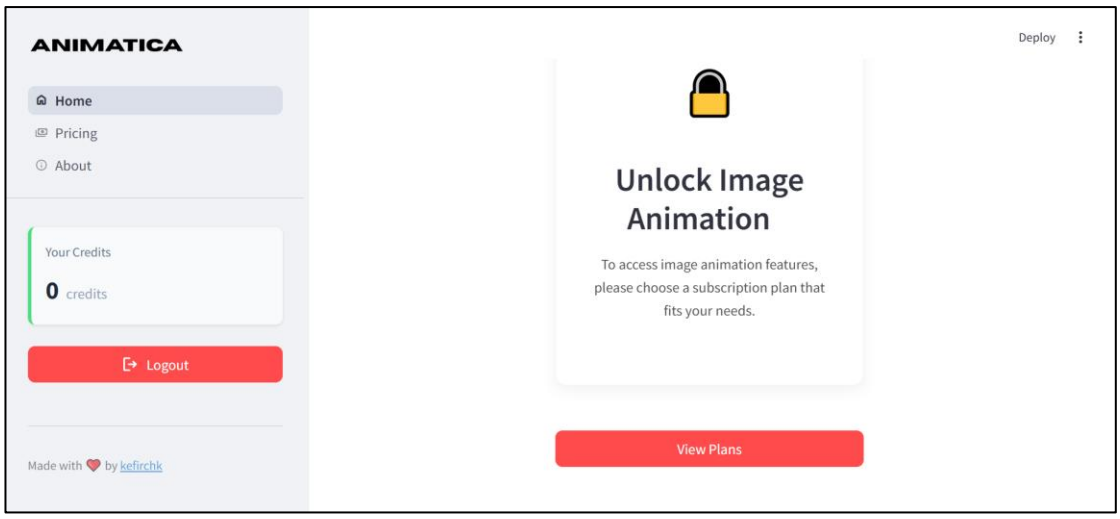


Рисунок 6.2 – Главная страница с заблокированной возможностью генерации анимации

По умолчанию страница заблокирована для генерации анимации, так как у пользователя после создания аккаунта нет попыток генерации анимации (англ. credits). Это можно исправить, перейдя на вкладку «Pricing» или кликнув на кнопку «View Plans». Откроется соответствующая вкладка (рисунок 6.3).

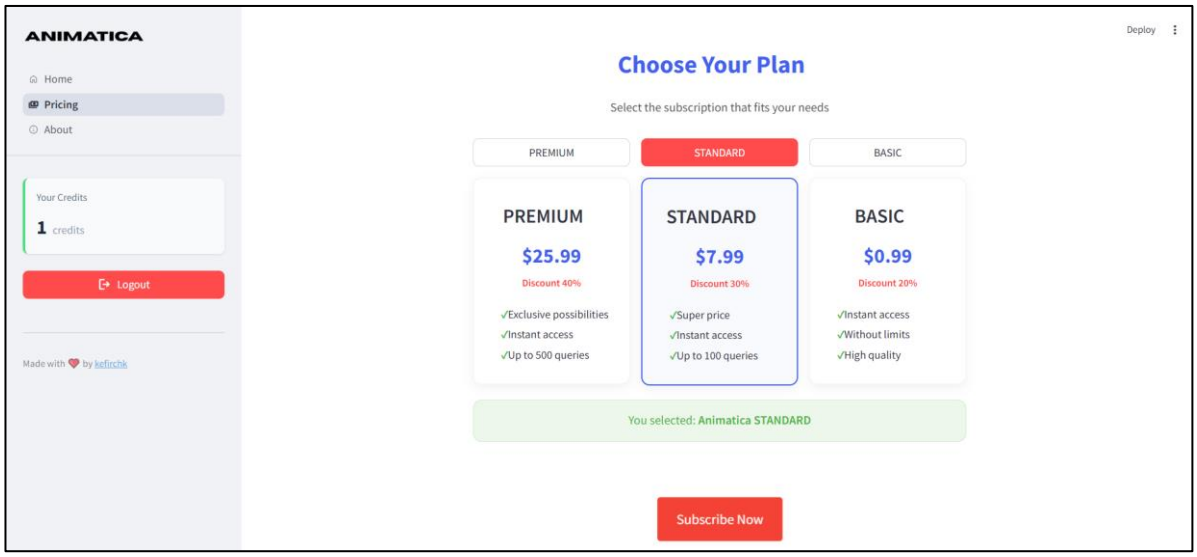


Рисунок 6.3 – Страница оплаты с доступными планами подписки

Пользователь может ознакомиться с возможными планами подписки и выбрать подходящий. Далее необходимо нажать на кнопку «Subscribe Now», после чего произойдет перенаправление на страницу оплаты выбранной подписки от Stripe (рисунок 6.4). В данном случае был выбран план подписки под названием «BASIC».

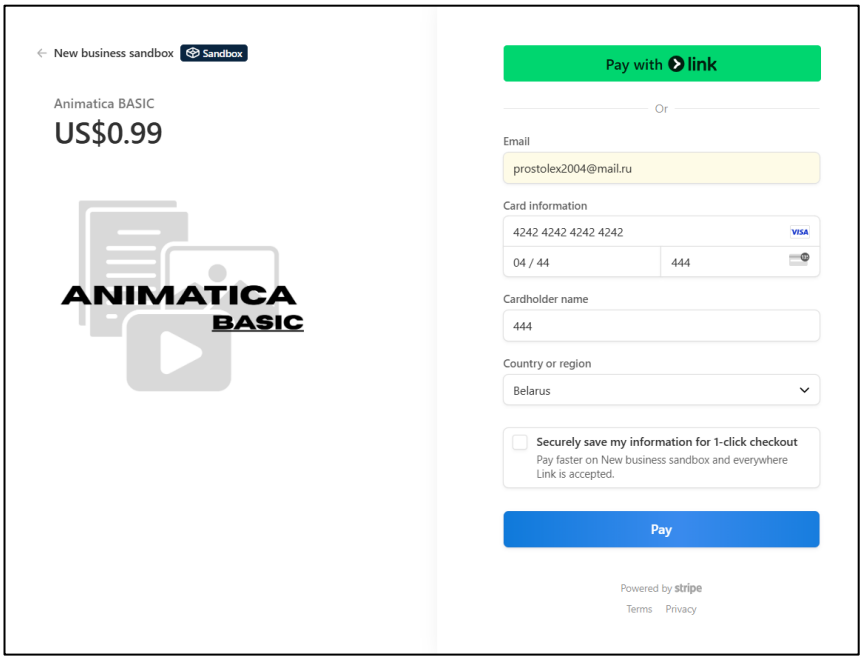


Рисунок 6.4 – Страница оплаты выбранной подписки от Stripe

Введя личные данные (email, card information, cardholder name, country or region), необходимо нажать на кнопку «Pay». Произойдет списание денежных средств и оплата плана подписки.

Затем произойдет перенаправление назад на страницу разработанного веб-приложения, на вкладку «Pricing», где пользователь может увидеть уведомление об успехе или неудачи проведенной операции (рисунок 6.5).

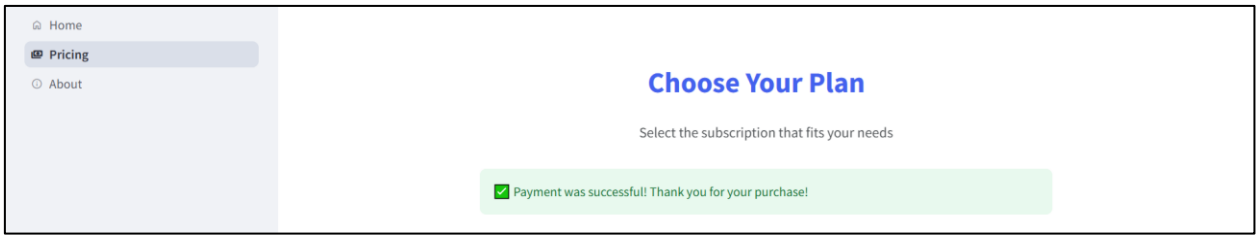


Рисунок 6.5 – Уведомление об успехе оплаты плана подписки

Теперь на балансе появился 1 credit. Это значит, что пользователь может воспользоваться сервисом генерации анимации 1 раз. Перейдя на вкладку «Home» можно заметить, что теперь никакой блокировки нет (рисунок 6.6).

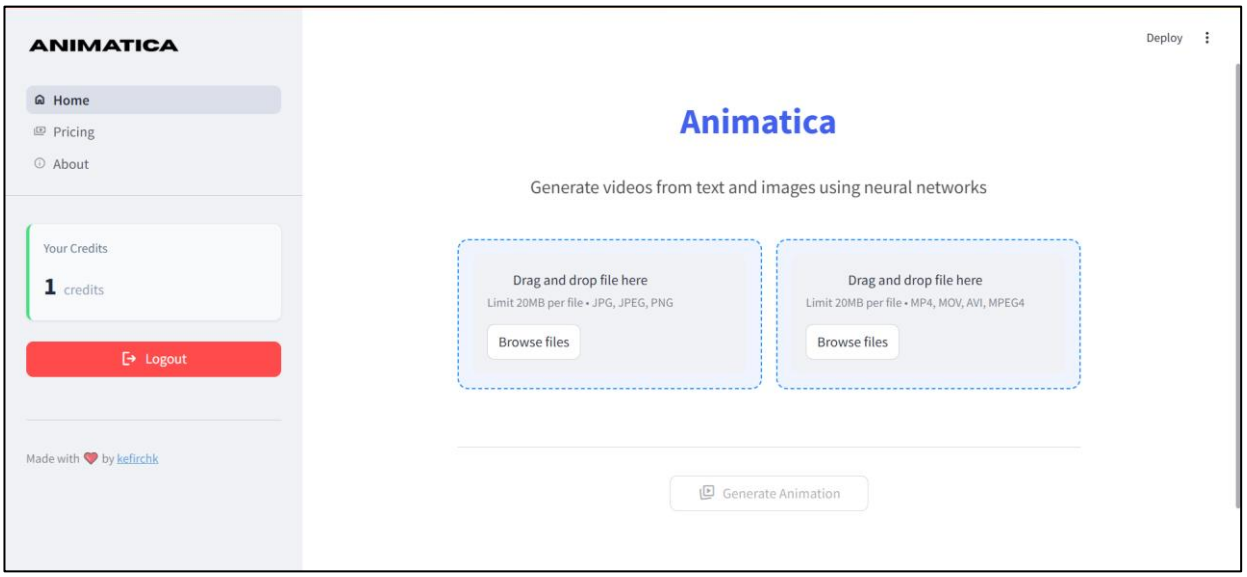


Рисунок 6.6 – Главная страница с возможностью генерации анимации

Страница разбита на две колонке, где в левой можно загрузить исходное изображение для анимации, а в правой – управляющее видео. Присутствует ограничение в 20 Мбайт для загрузки входных данных. Также есть ограничение на форматы загружаемых файлов: для изображения – только форматы JPG, JPEG или PNG, для видео – форматы MP4, AVI, MPEG4.

Загруженные пользователем файлы также визуально отображаются, видео можно запустить для просмотра (рисунок 6.7). Остается нажать на кнопку «Generate Animation» и дождаться завершения генерации анимации изображения (рисунок 6.8). Дождавшись окончания данного процесса на

странице появиться сгенерированное видео, которое можно просмотреть и скачать (рисунок 6.9).

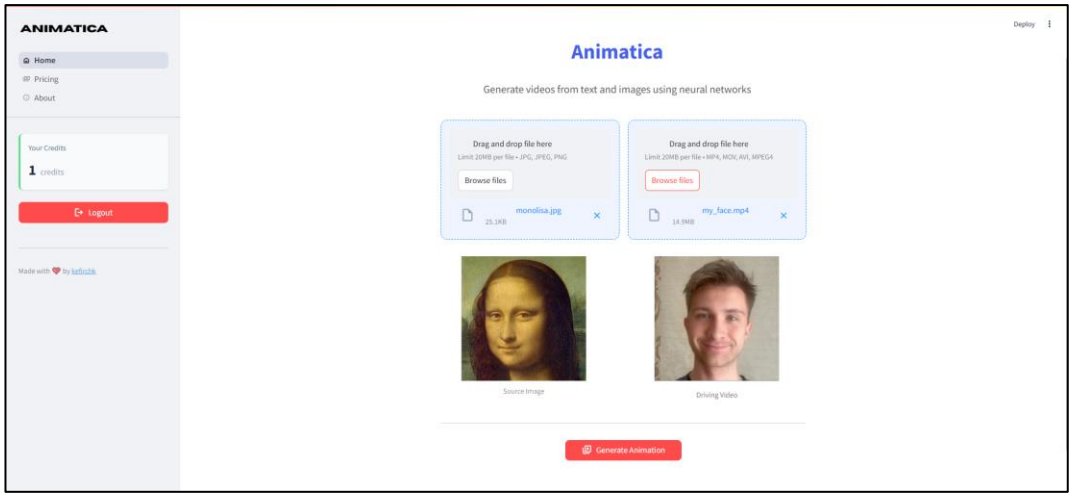


Рисунок 6.7 – Главная страница с загруженными файлами для генерации

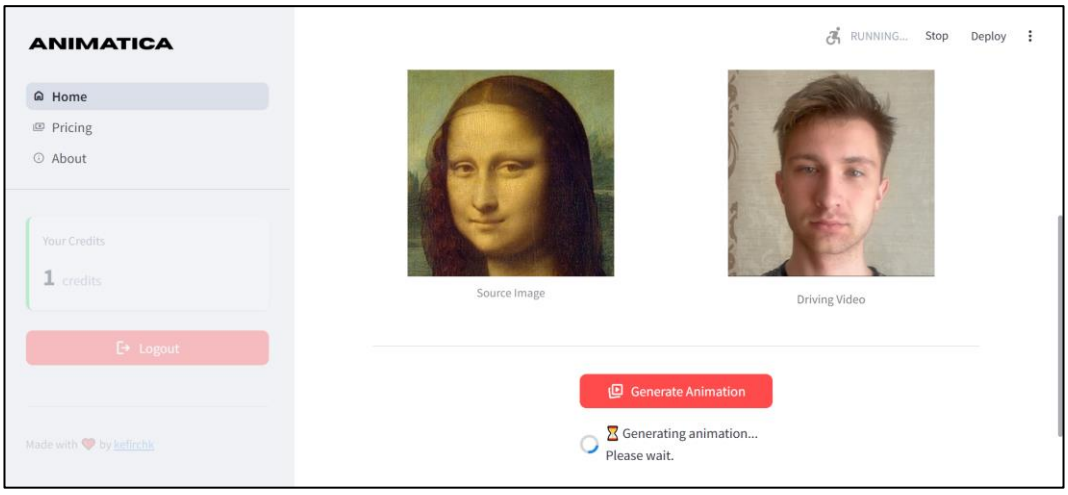


Рисунок 6.8 – Процесс генерации анимации

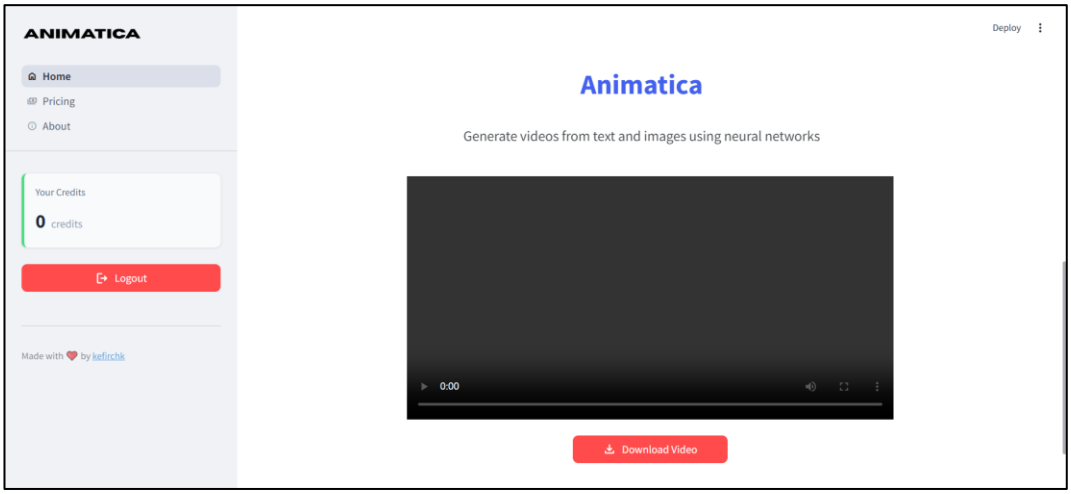


Рисунок 6.9 – Завершенный процесс генерации анимации

Для более детального ознакомления с продуктом предусмотрена страница «About», где можно узнать различные факты о проекте, его особенности. Здесь доступна информация о команде разработки и некоторая контактная информация, которой можно воспользоваться как для отправки каких-либо отзывов о продукте, так и в случае каких-то возникших ошибок (рисунок 6.10).

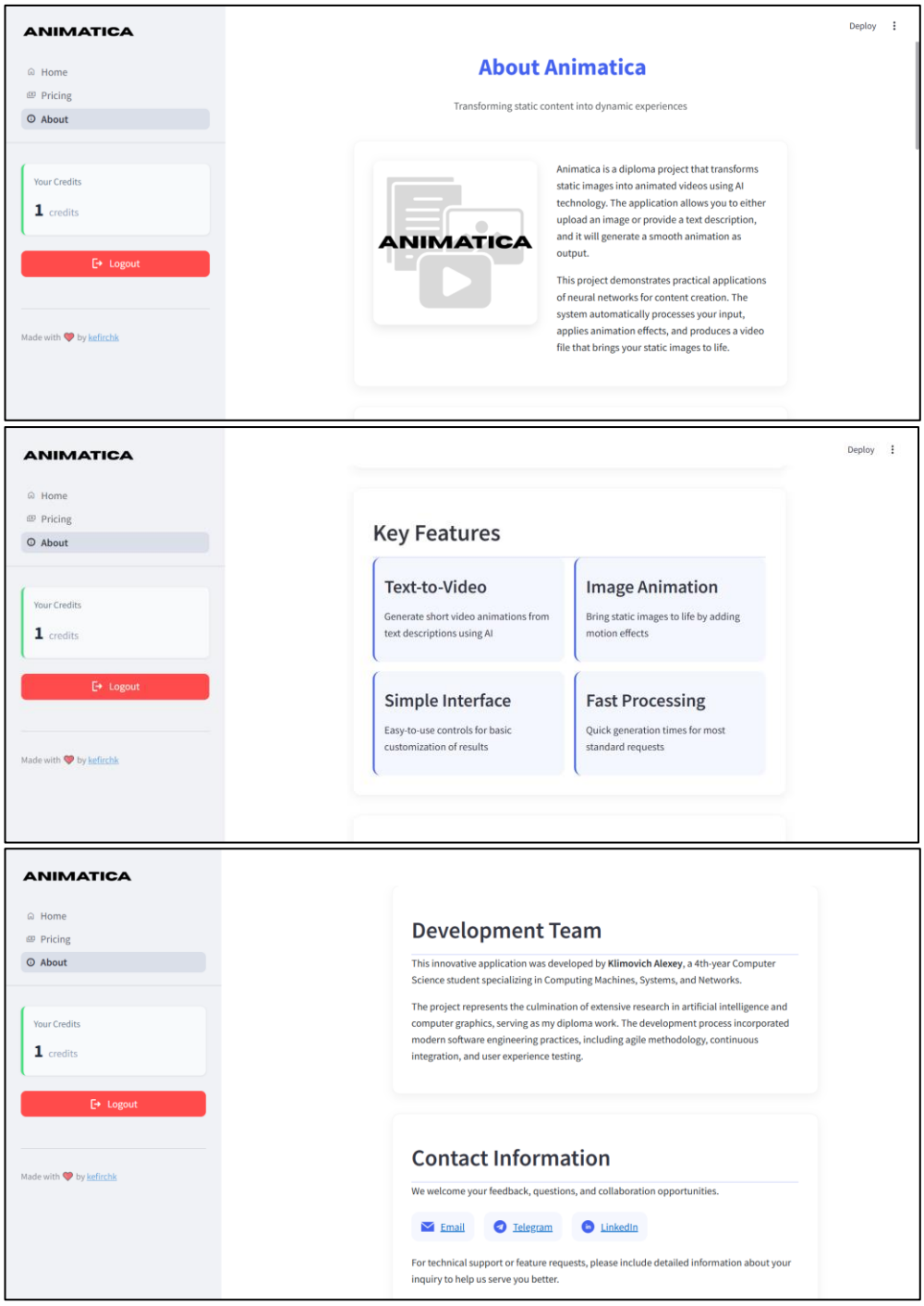


Рисунок 6.10 – Информационная страница

На боковой панели есть ссылка на страницу GitHub, где можно найти данный проект в открытом доступе.

# 7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И РЕАЛИЗАЦИИ НА РЫНКЕ ПРОГРАММНОГО СРЕДСТВА ДЛЯ АНИМАЦИИ ИЗОБРАЖЕНИЙ НА ОСНОВЕ НЕЙРОННЫХ СЕТЕЙ

## 7.1 Характеристика программного средства, разрабатываемого для реализации на рынке

В условиях стремительного развития технологий искусственного интеллекта и растущего спроса на мультимедийный контент возрастает потребность в инструментах, которые позволяют автоматизировать и ускорить процессы создания визуальных материалов. Разрабатываемое программное средство предназначено для анимации статичных изображений с использованием нейронных сетей, что делает возможным получение реалистичных видеопоследовательностей из одного изображения без необходимости ручной покадровой анимации.

Основной задачей проекта является упрощение и ускорение процесса анимации графических объектов за счет применения современных методов машинного обучения. Пользователь может загрузить изображение и получить видеоролик, в котором объект будет динамически «оживлен» – например, сымитированы движения лица, повороты головы или мимика. Это существенно снижает затраты времени и ресурсов по сравнению с традиционными методами анимации, особенно в сферах, где скорость производства контента критически важна.

Программное средство обладает широким спектром потенциального применения, включая следующие направления:

– рекламная индустрия – автоматическое создание визуального контента для социальных сетей, digital-кампаний и маркетинговых материалов;

– виртуальные ассистенты и цифровые аватары;

– образование и онлайн-курсы;

– геймдев и анимация;

– медиа и журналистика.

Ключевые функции программного средства включают:

– обработку изображений с поддержкой популярных форматов;

– генерацию видеороликов с анимацией на основе нейросетевой интерпретации изображения;

– экспорт полученных видеоматериалов в формате MP4 и других популярных расширениях;

– интеграцию с веб-интерфейсом для удобной загрузки изображений и управления процессом генерации;

– хранение и учет пользовательских запросов, управление тарифами и балансом генераций.

Приложение реализуется в виде веб-сервиса с современным, интуитивно понятным интерфейсом. Предусмотрена система подписки с различными тарифами – от базового до профессионального, что обеспечивает доступность

как для индивидуальных пользователей, так и для команд или организаций, заинтересованных в масштабировании видеоконтента.

## 7.2 Расчет инвестиций в разработку программного средства для его реализации на рынке

**7.2.1** Расходы на основную заработную плату рассчитываются на основе состава и численности команды, месячного оклада каждого сотрудника, объема выполненных ими задач, а также с учетом размера премиальных. Такой расчет может быть выполнен с использованием следующей формулы:

$$З_о = К_{пр} \cdot \sum_{i=1}^{n} З_{чi} \cdot t_i, \quad (7.1)$$

где $К_{пр}$ – коэффициент премий и иных стимулирующих выплат;

$n$ – категории исполнителей, занятых разработкой;

$З_{чi}$ – часовой оклад плата исполнителя $i$-й категории, р.;

$t_i$ – трудоемкость работ, выполняемых исполнителем $i$-й категории, ч.

Часовая ставка каждого сотрудника рассчитывается путем деления его месячного оклада на количество рабочих часов в месяц. В расчетах принимается, что месячное количество рабочих часов составляет 160. Расчет затрат на оплату труда разработчиков приведен в таблице 7.1.

Таблица 7.1 – Расчет затрат на основную заработную плату разработчиков

| Категория исполнителя | Месячный оклад, р. | Часовой оклад, р. | Трудоемкость работ, ч. | Итого, р. |
|---|---|---|---|---|
| Инженер по машинному обучению | 3800 | 23,75 | 160 | 3800 |
| Инженер-программист | 3500 | 21,88 | 160 | 3500 |
| Тестировщик | 2200 | 13,75 | 96 | 1320 |
| Проектный менеджер | 2150 | 13,44 | 40 | 537,60 |
| Итого | | | | 9158 |
| Премия и иные стимулирующие выплаты, 20% | | | | 1831,60 |
| Всего затрат на основную заработную плату разработчиков | | | | 10989,60 |

В команде проекта ключевую роль играет инженер по машинному обучению, который занимается разработкой и настройкой нейросетей. Его задача заключается в создании и обучении генеративных моделей, отвечающих за преобразование изображений и текста в видео. Специалист по машинному обучению настраивает архитектуру моделей, подбирает гиперпараметры и оптимизирует их работу, обеспечивая высокую производительность и качество генерируемого контента.

Важным участником процесса является инженер-программист, который

сосредоточен на разработке интерфейса и серверной части приложения. Он интегрирует обученные модели в веб-платформу, разрабатывает пользовательские интерфейсы и обеспечивает стабильное функционирование системы. Этот специалист также отвечает за настройку инфраструктуры для поддержания приложения в онлайн-режиме, обеспечивая его масштабируемость и доступность.

Для обеспечения качества и надежности проекта привлекается тестировщик, который проверяет работоспособность всех компонентов системы. Он проводит функциональные тесты на корректность выполнения всех операций, выявляет и устраняет ошибки, а также проверяет приложение на устойчивость к нагрузкам, чтобы гарантировать стабильную работу при высоких пользовательских запросах.

Организацию и координацию работы команды осуществляет проектный менеджер, который управляет ресурсами и следит за соблюдением сроков. Его задача также заключается в контроле качества выполнения задач, управлении рисками и взаимодействии с клиентами. Менеджер собирает обратную связь, вносит коррективы в проект по мере необходимости и поддерживает бесперебойное сотрудничество между членами команды.

Данные о заработных платах перечисленных специалистов были взяты из источников [29 – 32].

**7.2.2** Для расчета расходов на дополнительную заработную плату разработчиков используем следующую формулу:

$$З_д = \frac{З_о \cdot Н_д}{100}, \quad (7.2)$$

где $Н_д$ – норматив дополнительной заработной платы (равен 15%).

**7.2.3** Размер отчислений на социальное обеспечение рассчитывается на основе ставки отчислений, которая, согласно действующему законодательству на март 2025 года, составляет 35%.

Определим величину отчислений с помощью следующей формулы:

$$Р_{соц} = \frac{(З_о + З_д) \cdot Н_{соц}}{100}, \quad (7.3)$$

где $Н_{соц}$ – норматив отчислений в ФСЗН.

**7.2.4** Прочие расходы вычисляются с учетом норматива, который для данного расчета принимается равным 30%.

С учетом этого рассчитаем прочие расходы по следующей формуле:

$$Р_{пр} = \frac{З_о \cdot Н_{пр}}{100}, \quad (7.4)$$

где $Н_{пр}$ – норматив прочих расходов.

**7.2.5** Для расчета расходов на реализацию требуется учитывать норматив данных затрат, который в данном случае принимается равным 3%. Расчет этих расходов производится по следующей формуле:

$$Р_р = \frac{З_о \cdot Н_р}{100}, \quad (7.5)$$

где $Н_р$ – норматив расходов на реализацию.

**7.2.6** Общую сумму затрат можно определить как сумму ранее рассчитанных расходов, включая основную заработную плату, дополнительную заработную плату, отчисления на социальные нужды и прочие расходы.

Формула для вычисления общей суммы затрат выглядит следующим образом:

$$З_р = З_о + З_д + Р_{соц} + Р_{пр} + Р_р. \quad (7.6)$$

Итоговые затраты на разработку программного средства рассчитываются с использованием вышеуказанной формулы, и результаты представлены в таблице 7.2.

Таблица 7.2 – Расчет инвестиций на разработку программного средства

| Наименование статьи затрат | Формула/таблица для расчета | Значение, р. |
|---|---|---|
| 1 Основная заработная плата разработчиков | Таблица 7.1 | 10989,60 |
| 2 Дополнительная заработная плата разработчиков | $З_д = \frac{10989,60 \cdot 15}{100}$ | 1648,44 |
| 3 Отчисления на социальные нужды | $Р_{соц} = \frac{(10989,60 + 1638,44) \cdot 35}{100}$ | 4423,31 |
| 4 Прочие расходы | $Р_{пр} = \frac{10989,60 \cdot 30}{100}$ | 3296,88 |
| 5 Расходы на реализацию | $Р_р = \frac{10989,60 \cdot 3}{100}$ | 329,69 |
| 6 Общая сумма затрат на разработку и реализацию | $З_р = 10989,60 + 1648,44 + 4423,31 + + 3296,88 + 329,69$ | 20687,92 |

## 7.3 Расчет экономического эффекта от реализации программного средства на рынке

Оценка экономического эффекта от реализации программного средства позволит определить его коммерческую привлекательность и потенциальную прибыльность. Эта величина зависит от объема продаж, стоимости продукта и затрат на его разработку.

Для определения цены программного средства следует провести анализ аналогичных продуктов на рынке. Аналогичные решения, подобные данному проекту, часто работают по модели подписки. В таблице 7.3 приведены примеры аналогичных продуктов [33 – 44] и их стоимость за использование.

Таблица 7.3 – Цена использования продуктов–аналогов

| Название продукта | План подписки | Цена, $/месяц |
|---|---|---|
| Invideo | Бизнес | 15 |
| Renderforest | Профессиональный | 21,10 |
| Wave.video | Создатель | 24 |
| Pictory.ai | Профессиональный | 39 |
| Synthesys | Создатель | 41 |
| AI Studios | Персональный | 24 |
| Vidnoz | Бизнес | 56,99 |
| Pipio.ai | Профессиональный | 16 |
| Colossyan | Бизнес | 70 |
| VidAU | Бизнес | 45 |
| Fliki | Стандартный | 21 |
| Elai | Базовый | 23 |

Из представленной таблицы можно вычислить, что средняя стоимость подписки составляет 33 доллара США, что в пересчете на белорусские рубли составляет 108,05 рублей.

Для расчета прироста чистой прибыли необходимо учесть налог на добавленную стоимость (НДС), который рассчитывается по следующей формуле:

$$НДС = \frac{Ц_{отп} \cdot N \cdot Н_{д.с}}{100 + Н_{д.с}}, \quad (7.7)$$

где $N$ – количество оплат за подписку на программный продукт за год, шт.;

$Ц_{отп}$ – отпускная цена подписки программного средства, р.;

$Н_{д.с}$ – ставка налога на добавленную стоимость, %.

Отпускная цена подписки на программное средство установлена на уровне 80 рублей. Ожидается, что за год количество покупок подписки составит примерно пять тысяч. Ставка налога на добавленную стоимость, действующая на март 2025 года согласно законодательству Республики

Беларусь, составляет 20%. Используя эту информацию, можно рассчитать НДС:

$$НДС = \frac{80 \cdot 5000 \cdot 20}{100 + 20} = 66666{,}67 \text{ р.}$$

После вычисления налога на добавленную стоимость, можно рассчитать прирост чистой прибыли, который разработчик получит от реализации программного продукта. Для этого применяется следующая формула:

$$\Delta П_{ч}^{р} = (Ц_{отп} \cdot N - НДС) \cdot Р_{пр} \cdot (1 - \frac{Н_{п}}{100}), \quad (7.8)$$

где $Ц_{отп}$ – отпускная цена подписки программного средства, р.;

$N$ – количество оплат за подписку на программный продукта за год, шт.;

НДС – сумма налога на добавленную стоимость, р.;

$Р_{пр}$ – рентабильность продаж;

$Н_{п}$ – ставка налога на прибыль.

Ставка налога на прибыль, в соответствии с действующим законодательством на март 2025 года, составляет 20%. Рентабельность продаж программного продукта установлена на уровне 30%.

Прирост чистой прибыли вычисляется по следующей формуле:

$$\Delta П_{ч}^{р} = (80 \cdot 5000 - 180083{,}33) \cdot 30\% \cdot (1 - \frac{20}{100}) = 79999{,}99 \text{ р.}$$

## 7.4 Расчет показателей экономической эффективности разработки и реализации программного средства на рынке

Для оценки экономической эффективности разработки и внедрения программного продукта на рынок, необходимо провести сравнительный анализ затрат на его разработку и ожидаемого прироста чистой прибыли за год. Если сумма затрат на разработку меньше годового экономического эффекта, то можно заключить, что инвестиции окупятся менее чем за один год.

Оценка эффективности инвестиций проводится с использованием расчета рентабельности инвестиций (англ. Return on Investment, ROI). Формула для вычисления ROI следующая:

$$ROI = \frac{\Delta П_{ч}^{р} - З_{р}}{З_{р}}, \quad (7.9)$$

где $\Delta П_{ч}^{р}$ – прирост чистой прибыли, полученной от реализации программного средства на рынке, р;

$З_{р}$ – затраты на разработку и реализацию программного средства, р.

$$ROI = \frac{216100 - 20687,92}{20687,92} \cdot 100\% = 286,69\%$$

Данный уровень рентабельности подтверждает экономическую обоснованность реализации проекта и свидетельствует о его высоком потенциале для будущего роста и развития.

## 7.5 Вывод об экономической целесообразности реализации проектного решения

Проект по разработке программного средства для анимации изображений на основе нейронных сетей продемонстрировал положительные результаты с точки зрения экономической целесообразности. Ожидаемый ROI составляет 286,69%, что указывает на высокую рентабельность и привлекательность инвестиций. Чистая прибыль, которую проект может генерировать ежегодно, составляет около 80 тыс. белорусских рублей. Эти показатели подтверждают, что проект обладает значительным потенциалом для получения прибыли в краткосрочной и среднесрочной перспективе.

Для привлечения большего числа пользователей, проект может предложить базовую бесплатную версию программного средства с ограниченными функциями. Это создаст дополнительную привлекательность продукта, позволит пользователям ознакомиться с функционалом и, возможно, приведет к конверсии в платные подписки для получения расширенных возможностей.

Но несмотря на привлекательность продукта, рынок генерации видео уже насыщен конкурентами, предлагающими аналогичные решения. Это создает угрозу для захвата значительной доли рынка, особенно если продукт не будет достаточно уникален или не оправдает ожидания пользователей в плане качества.

Также стоит учесть, что реализация нейронных сетей для генерации видео на основе изображений и текста – это сложная задача. Возможны трудности с вычислительными ресурсами, а также с оптимизацией работы нейросетевых моделей для качественной и быстрой генерации контента.

С учетом правильного подхода к улучшению качества продукта, актуальности для целевых пользователей и эффективному маркетингу, проект имеет все шансы на успех и получение прибыли в короткие сроки.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломного проекта было разработано программное решение, позволяющее создавать анимации из статических изображений с применением современных методов машинного обучения. Основу системы составляет архитектура нейросети First Order Motion Model (FOMM), реализованная с использованием PyTorch. Вычислительное ядро позволяет производить генерацию видеопоследовательностей на основе пары «изображение-видео», что делает систему гибкой и применимой для широкого круга визуальных задач.

Решение обладает модульной архитектурой, включающей в себя:

– backend-сервер на FastAPI, обеспечивающий обработку запросов, взаимодействие с моделью и генерацию выходного видео;

– frontend-интерфейс на Streamlit, предоставляющий пользователю простой и удобный способ загрузки изображений и видео, отслеживания процесса и получения результата;

– интеграцию с платежной системой Stripe, позволяющую реализовать монетизацию доступа к сервису;

– хранение и управление данными о пользователях с использованием PostgreSQL;

– контейнеризацию проекта с помощью Docker, что упрощает развертывание на различных платформах.

Основными достоинствами проекта являются простота использования, простота поддержки, относительно невысокая стоимость, масштабируемость, кроссплатформенность. Проект может функционировать как на системах с GPU, так и без него, что расширяет его применимость, особенно для пользователей без доступа к специализированному оборудованию.

Возможности развития проекта:

– поддержка пакетной обработки;

– интеграция более совершенных моделей генерации с учетом временных связей для более качественной и быстрой работы;

– разработка мобильной версии или облачного API.

В целом реализованная система подтверждает эффективность применения генеративных моделей в задаче автоматической анимации изображений. Предложенный подход позволяет существенно снизить трудозатраты на создание визуального контента, делая процесс доступным для широкой аудитории без навыков программирования или компьютерной графики. Благодаря модульной архитектуре, проект может быть легко адаптирован к другим задачам – генерации лицевой мимики, стилизации видео, виртуальных аватаров и многому другому.

Разработанное решение демонстрирует высокую степень зрелости и готово к практическому применению в образовательных, маркетинговых и креативных сценариях. В дальнейшем проект может быть доработан и выведен на рынок как коммерческий продукт или open-source инструмент.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


[1] RunwayML [Электронный ресурс]. – Режим доступа: https://app.runwayml.com/. – Дата доступа: 03.02.2025.

[2] Deep Nostalgia [Электронный ресурс]. – Режим доступа : https://www.myheritage.com/deep-nostalgia. – Дата доступа: 03.02.2025.

[3] AI Face Animator [Электронный ресурс]. – Режим доступа : https://www.fotor.com/features/face-animator/. – Дата доступа: 03.02.2025.

[4] Cutout.pro [Электронный ресурс]. – Режим доступа : https://www.cutout.pro/. – Дата доступа: 03.02.2025.

[5] X2Face: A network for controlling face generation using images, audio, and pose codes [Электронный ресурс]. – Режим доступа : https://arxiv.org/pdf/1807.10550. – Дата доступа: 10.03.2025.

[6] Animating Arbitrary Objects via Deep Motion Transfer [Электронный ресурс]. – Режим доступа : https://arxiv.org/pdf/1812.08861. – Дата доступа: 10.03.2025.

[7] First Order Motion Model for Image Animation [Электронный ресурс]. – Режим доступа : https://arxiv.org/pdf/2003.00196. – Дата доступа: 10.03.2025.

[8] Python 3.12 documentation [Электронный ресурс]. – Режим доступа : https://docs.python.org/3.12/. – Дата доступа: 01.02.2025.

[9] Самые популярные языки программирования в 2025 году [Электронный ресурс]. – Режим доступа : https://dan-it.com.ua/blog/samye-populjarnye-jazyki-programmirovanija-v-2025-godu/. – Дата доступа: 01.02.2025.

[10] TIOBE Index for March 2025 [Электронный ресурс]. – Режим доступа : https://www.tiobe.com/tiobe-index/. – Дата доступа: 01.02.2025.

[11] PyTorch [Электронный ресурс]. – Режим доступа : https://pytorch.org/. – Дата доступа: 05.02.2025.

[12] TensorFlow [Электронный ресурс]. – Режим доступа : https://www.tensorflow.org/. – Дата доступа: 01.02.2025.

[13] Pytorch vs Tensorflow: A Head-to-Head Comparison [Электронный ресурс]. – Режим доступа : https://viso.ai/deep-learning/pytorch-vs-tensorflow/. – Дата доступа: 01.02.2025.

[14] FastAPI [Электронный ресурс]. – Режим доступа : https://fastapi.tiangolo.com. – Дата доступа: 10.03.2025.

[15] PostgreSQL [Электронный ресурс]. – Режим доступа : https://www.postgresql.org/. – Дата доступа: 10.03.2025.

[16] A Beginners Guide To Streamlit [Электронный ресурс]. – Режим доступа : https://www.geeksforgeeks.org/a-beginners-guide-to-streamlit/. – Дата доступа: 10.03.2025.

[17] Why and how to use Google Colab [Электронный ресурс]. – Режим доступа : https://www.techtarget.com/searchenterpriseai/tutorial/Why-and-how-to-use-Google-Colab. – Дата доступа: 20.02.2025.

[18] PyCharm [Электронный ресурс]. – Режим доступа : https://www.jetbrains.com/pycharm/. – Дата доступа: 20.02.2025.

[19] VoxCeleb [Электронный ресурс]. – Режим доступа : https://www.robots.ox.ac.uk/~vgg/data/voxceleb/. – Дата доступа: 28.02.2025.

[20] Docker [Электронный ресурс]. – Режим доступа : https://docs.docker.com/get-started/docker-overview/. – Дата доступа: 15.03.2025.

[21] Stripe [Электронный ресурс]. – Режим доступа : https://stripe.com/. – Дата доступа: 15.03.2025.

[22] A quick introduction to clean architecture [Электронный ресурс]. – Режим доступа : https://www.freecodecamp.org/news/a-quick-introduction-to-clean-architecture-990c014448d2/. – Дата доступа: 25.02.2025.

[23] PEP 8 – Style Guide for Python Code [Электронный ресурс]. – Режим доступа : https://peps.python.org/pep-0008/. – Дата доступа: 25.02.2025.

[24] GitHub Actions documentation [Электронный ресурс]. – Режим доступа : https://docs.github.com/en/actions. – Дата доступа: 01.04.2025.

[25] Welcome to Alembic’s documentation! [Электронный ресурс]. – Режим доступа : https://alembic.sqlalchemy.org/en/latest/. – Дата доступа: 01.04.2025.

[26] isort [Электронный ресурс]. – Режим доступа : https://pycqa.github.io/isort/. – Дата доступа: 01.04.2025.

[27] Black 25.1.0 documentation [Электронный ресурс]. – Режим доступа : https://black.readthedocs.io/en/stable/. – Дата доступа: 01.04.2025.

[28] pre-commit [Электронный ресурс]. – Режим доступа : https://pre-commit.com/. – Дата доступа: 01.04.2025.

[29] Зарплаты Product-менеджеров [Электронный ресурс]. – Режим доступа : https://geeklink.io/salary-stat/product-menedzher/. – Дата доступа: 01.03.2025.

[30] Зарплаты инженеров машинного обучения [Электронный ресурс]. – Режим доступа : https://geeklink.io/salary-stat/machine-learning-engineer/. – Дата доступа: 01.03.2025.

[31] Зарплаты Python-разработчиков [Электронный ресурс]. – Режим доступа : https://geeklink.io/salary-stat/python-razrabotchik/. – Дата доступа: 01.03.2025.

[32] Зарплаты тестировщиков ПО [Электронный ресурс]. – Режим доступа : https://geeklink.io/salary-stat/testirovshhik-po/. – Дата доступа: 01.03.2025.

[33] Invideo Pricing [Электронный ресурс]. – Режим доступа : https://invideo.io/studio-pricing. – Дата доступа: 05.03.2025.

[34] Renderforest Pricing [Электронный ресурс]. – Режим доступа : https://www.renderforest.com/ru/subscription. – Дата доступа: 05.03.2025.

[35] Wave.video Pricing [Электронный ресурс]. – Режим доступа : https://wave.video/pricing. – Дата доступа: 05.03.2025.

[36] Pictory.ai Pricing [Электронный ресурс]. – Режим доступа : https://pictory.ai/pricing. – Дата доступа: 05.03.2025.

[37] Synthesys Pricing [Электронный ресурс]. – Режим доступа : https://synthesys.io/pricing/. – Дата доступа: 05.03.2025.

[38] AI Studios Pricing [Электронный ресурс]. – Режим доступа : https://www.aistudios.com/pricing. – Дата доступа: 05.03.2025.

[39] Vidnoz Pricing [Электронный ресурс]. – Режим доступа : https://www.vidnoz.com/pricing.html. – Дата доступа: 05.03.2025.

[40] Pipio.ai Pricing [Электронный ресурс]. – Режим доступа : https://www.pipio.ai/pricing. – Дата доступа: 05.03.2025.

[41] Colossyan Pricing [Электронный ресурс]. – Режим доступа : https://www.colossyan.com/pricing. – Дата доступа: 05.03.2025.

[42] Vidau Pricing [Электронный ресурс]. – Режим доступа : https://www.vidau.ai/pricing/. – Дата доступа: 05.03.2025.

[43] Fliki Pricing [Электронный ресурс]. – Режим доступа : https://fliki.ai/pricing. – Дата доступа: 05.03.2025.

[44] Elai Pricing [Электронный ресурс]. – Режим доступа : https://elai.io/pricing/. – Дата доступа: 05.03.2025.

Стандарт предприятия. Дипломные проекты (работы). Общие требования. СТП 01-2024 [Электронный ресурс] : БГУИР. – Режим доступа : https://www.bsuir.by/m/12_100229_1_185678.pdf. – Дата доступа: 20.03.2025.

Экономика проектных решений: методические указания по экономическому обоснованию дипломных проектов : учеб.-метод. пособие / В.Г. Горовой [и др.]. – Минск : БГУИР. 2021. – 107 с.

# ПРИЛОЖЕНИЕ А
(обязательное)

## Листинг кода

```
# Файл "app\src\main.py":

import logging
from contextlib import asynccontextmanager

import src
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from src.domain.entities.enums import LogLevelEnum
from src.infrastructure.api import router as api_router
from src.infrastructure.configs import APIConfig
from src.infrastructure.exceptions.handlers import (
    app_exception_handler,
    exception_handlers,
)
from starlette.middleware.sessions import SessionMiddleware

logging.basicConfig(format="[PID:%(process)d] %(pathname)s:%(lineno)d %(message)s",
level=logging.INFO)
logging.getLogger("sqlalchemy.engine").setLevel(logging.INFO)


app = FastAPI(
    title=f"{APIConfig().MODE.capitalize()} Animatica Backend API",
    description="This API is designed for the Animatica application.",
    swagger_ui_parameters={"displayRequestDuration": True},
    version=src.__version__,
    debug=(APIConfig().LOG_LEVEL == LogLevelEnum.DEBUG),
)

app.add_middleware(SessionMiddleware, secret_key=APIConfig().SESSION_SECRET_KEY)
app.add_middleware(
    CORSMiddleware,
    allow_credentials=True,
    allow_origins=APIConfig().allowed_origins,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(api_router, prefix="/api")

for exc_type in exception_handlers:
    app.add_exception_handler(exc_type, app_exception_handler)

# Файл "app\src\domain\entities\auth.py":

from pydantic import BaseModel


class UserAuthInfo(BaseModel):
    sub_id: str

# Файл "app\src\domain\entities\response.py":

from pydantic import BaseModel
from src.infrastructure.exceptions.handlers import handle_exception
from starlette.responses import JSONResponse
from starlette.status import HTTP_200_OK


class ResponseModel(BaseModel):
    pass


class ResponseFailure:
    def __init__(self, err_msg: str, status_code: int) -> None:
        self.err_msg = err_msg
        self.status = status_code

    @classmethod
    def build(cls, exc: Exception) -> JSONResponse:
```

```
        return JSONResponse(*handle_exception(exc))


class ResponseSuccess:
    def __init__(self, payload: ResponseModel, status: int) -> None:
        self.payload = payload
        self.status = status

    @classmethod
    def build(cls, payload: ResponseModel | list[ResponseModel], status: int = HTTP_200_OK,
**kwargs) -> JSONResponse:
        if isinstance(payload, list):
            content = [item.model_dump(**kwargs) for item in payload]
        else:
            content = payload.model_dump(**kwargs)

        return JSONResponse(content=content, status_code=status)

# Файл "app\src\domain\entities\subscription.py":

from pydantic import BaseModel, Field


class Subscription(BaseModel):
    subscription_id: int = Field(gt=0)
    user_id: int = Field(gt=0)
    remaining_queries: int = Field(ge=0)

# Файл "app\src\domain\entities\user.py":

from pydantic import BaseModel, ConfigDict


class UserRead(BaseModel):
    username: str
    model_config = ConfigDict(from_attributes=True)

# Файл "app\src\domain\interfaces\db_repository.py":

from abc import ABC

from sqlalchemy.exc import OperationalError
from sqlalchemy.ext.asyncio import AsyncSession, create_async_engine
from sqlalchemy.orm.session import sessionmaker
from src.infrastructure.configs import DBConfig

engine = create_async_engine(
    DBConfig().db_conn_url,
    pool_size=10,
    max_overflow=20,
)
SessionFactory = sessionmaker(bind=engine, class_=AsyncSession)


class IDBRepository(ABC):
    async def __aenter__(self) -> "IDBRepository":
        self.async_session: AsyncSession = SessionFactory()
        return self

    async def __aexit__(self, exc_type, exc_val, exc_tb) -> None:
        if exc_val:
            await self.async_session.rollback()
            await self.async_session.close()
            if exc_type == OperationalError:
                raise ConnectionError("Could not connect to DataBase instance")
            return
        await self.async_session.commit()
        await self.async_session.close()

# Файл "app\src\domain\interfaces\use_case.py":

from abc import ABC, abstractmethod
from src.domain.entities.response import ResponseFailure, ResponseSuccess


class IUseCase(ABC):
    @abstractmethod
    async def execute(self, *args, **kwargs) -> ResponseSuccess | ResponseFailure:
        pass
```

```
# Файл "app\src\domain\usecases\animation\animate_image_by_video.py":

import httpx
from fastapi import UploadFile
from src.domain.entities.auth import UserAuthInfo
from src.domain.entities.request import RequestModel
from src.domain.entities.response import ResponseFailure
from src.domain.interfaces import IUseCase
from src.infrastructure.configs import MLEngineConfig
from src.infrastructure.exceptions.exceptions import MLEngineException
from src.infrastructure.repositories import (
    SubscriptionRepository,
    UserRepository,
)
from starlette.responses import JSONResponse, StreamingResponse


class AnimateImageByVideoUseCase(IUseCase):
    class Request(RequestModel):
        user: UserAuthInfo
        source_image: UploadFile
        driving_video: UploadFile

    def __init__(self):
        self.ml_engine_config = MLEngineConfig()
        self.user_repository = UserRepository()
        self.subscription_repository = SubscriptionRepository()

    async def execute(self, request: Request) -> StreamingResponse | JSONResponse:
        try:
            source_image = request.source_image
            driving_video = request.driving_video
            files = {
                "source_image": (source_image.filename, await source_image.read(),
source_image.content_type),
                "driving_video": (driving_video.filename, await driving_video.read(),
driving_video.content_type),
            }
            async with self.user_repository as repository:
                user = await repository.get_user_by_username(request.user.sub_id)
                user_id = user.id

            async with httpx.AsyncClient(timeout=None) as client:
                response = await client.post(
                    f"{self.ml_engine_config.BASE_URL}/api/fomm/video",
                    files=files,
                    headers={"X-ML-Engine-Key": self.ml_engine_config.ML_ENGINE_KEY},
                )
                if response.status_code != 200:
                    detail = await response.aread()
                    raise MLEngineException(detail.decode())

                async with self.subscription_repository as repository:
                    subscription = await repository.get_one_by_user_id(user_id=user_id)
                    await repository.update_one(
                        existing_subscription=subscription,
                        queries=-1,
                    )

                return StreamingResponse(response.aiter_bytes(), media_type="video/mp4")

        except Exception as exc:
            return ResponseFailure.build(exc)

# Файл "app\src\domain\usecases\auth\login.py":

from src.domain.entities.request import RequestModel
from src.domain.entities.response import (
    ResponseFailure,
    ResponseModel,
    ResponseSuccess,
)
from src.domain.interfaces import IUseCase
from src.infrastructure.configs.auth_config import AuthConfig
from src.infrastructure.exceptions.exceptions import (
    InvalidCredentialsException,
)
from src.infrastructure.services.auth import AuthService
from src.infrastructure.services.auth.token import TokenService
from starlette.responses import JSONResponse
```

```
class UserLoginUseCase(IUseCase):
    class Request(RequestModel):
        username: str
        password: str

    class Response(ResponseModel):
        token_name: str
        refresh_token: str
        access_token: str
        access_expires_in: int
        refresh_expires_in: int

    def __init__(self) -> None:
        self.auth_service = AuthService()
        self.token_service = TokenService()
        self.auth_config = AuthConfig()

    async def execute(self, request: Request) -> JSONResponse:
        try:
            user = await self.auth_service.authenticate_user(request.username, request.password)
            if not user:
                raise InvalidCredentialsException("Invalid username or password")

            token_data = await self.token_service.generate_tokens(user.username)

            return ResponseSuccess.build(
                self.Response(
                    token_name="Bearer",
                    access_token=token_data["access_token"],
                    refresh_token=token_data["refresh_token"],
                    access_expires_in=token_data["access_expires_in"],
                    refresh_expires_in=token_data["refresh_expires_in"],
                )
            )
        except Exception as exc:
            return ResponseFailure.build(exc)

# Файл "app\src\domain\usecases\auth\signup.py":

from src.domain.entities.request import RequestModel
from src.domain.entities.response import (
    ResponseFailure,
    ResponseModel,
    ResponseSuccess,
)
from src.domain.interfaces import IUseCase
from src.infrastructure.exceptions.exceptions import UserAlreadyExistsException
from src.infrastructure.repositories.user_repository import UserRepository
from starlette import status
from starlette.responses import JSONResponse


class UserSignUpUseCase(IUseCase):
    class Request(RequestModel):
        password: str
        username: str

    class Response(ResponseModel):
        pass

    def __init__(self) -> None:
        self.user_repository = UserRepository()

    async def execute(self, request: Request) -> JSONResponse:
        try:
            async with self.user_repository as repository:
                if await repository.get_user_by_username(request.username):
                    raise UserAlreadyExistsException(f"Username '{request.username}' already
exists")

                await repository.create_user(username=request.username,
password=request.password)

            return ResponseSuccess.build(self.Response(), status=status.HTTP_201_CREATED)

        except Exception as exc:
            return ResponseFailure.build(exc)

# Файл "app\src\domain\usecases\payments\create_checkout_session.py":
```

```
import logging

from src.domain.entities.auth import UserAuthInfo
from src.domain.entities.request import RequestModel
from src.domain.entities.response import (
    ResponseFailure,
    ResponseModel,
    ResponseSuccess,
)
from src.domain.interfaces import IUseCase
from src.infrastructure.configs import APIConfig
from src.infrastructure.repositories.stripe_repository import StripeRepository
from starlette.responses import JSONResponse

log = logging.getLogger(__name__)
class CreateCheckoutSessionUseCase(IUseCase):
    class Request(RequestModel):
        user: UserAuthInfo
        price_id: str

    class Response(ResponseModel):
        checkout_session: dict

    def __init__(self):
        self.redirect_url = f"{APIConfig().FRONTEND_BASE_URL}/pricing"
        self.stripe_repository = StripeRepository()

    async def execute(self, request: Request) -> JSONResponse:
        try:
            log.info(f"Creating checkout session with price_id: {request.price_id}")

            params = {
                "payment_method_types": ["card"],
                "line_items": [{"price": request.price_id, "quantity": 1}],
                "mode": "payment",
                "success_url": f"{self.redirect_url}?success=true",
                "cancel_url": f"{self.redirect_url}?canceled=true",
            }
            checkout_session = await self.stripe_repository.create_checkout_session(params)

            return ResponseSuccess.build(self.Response(checkout_session=checkout_session))
        except Exception as exc:
            return ResponseFailure.build(exc)

# Файл "app\src\domain\usecases\payments\create_payment.py":

from src.domain.entities.auth import UserAuthInfo
from src.domain.entities.request import RequestModel
from src.domain.entities.response import (
    ResponseFailure,
    ResponseModel,
    ResponseSuccess,
)
from src.domain.interfaces import IUseCase
from src.infrastructure.configs import APIConfig
from src.infrastructure.exceptions.exceptions import (
    StripeException,
    StripePaymentException,
)
from src.infrastructure.repositories import (
    SubscriptionRepository,
    UserRepository,
)
from src.infrastructure.repositories.stripe_repository import StripeRepository
from starlette.responses import JSONResponse


class CreatePaymentUseCase(IUseCase):
    class Request(RequestModel):
        user: UserAuthInfo
        session_id: str
        product_id: str

    class Response(ResponseModel):
        remaining_queries: int
        session_id: str

    def __init__(self):
        self.redirect_url = f"{APIConfig().FRONTEND_BASE_URL}/pricing"
        self.stripe_repository = StripeRepository()
```

```
        self.subscription_repository = SubscriptionRepository()
        self.user_repository = UserRepository()

    async def execute(self, request: Request) -> JSONResponse:
        try:
            product = await self.stripe_repository.get_product(request.product_id)
            checkout_session = await
self.stripe_repository.get_checkout_session(request.session_id)
            if checkout_session.payment_status != "paid":
                raise StripePaymentException()

            queries_amount = int(product.metadata.get("queries", 0))
            if queries_amount <= 0:
                raise StripeException("Invalid queries amount in product metadata", 400)

            async with self.user_repository as repository:
                user = await repository.get_user_by_username(request.user.sub_id)
                user_id = user.id

            async with self.subscription_repository as repository:
                subscription = await repository.get_one_by_user_id(user_id=user_id)
                if subscription:
                    subscription = await repository.update_one(
                        existing_subscription=subscription,
                        queries=queries_amount,
                    )
                else:
                    subscription = await repository.add_one(
                        user_id=user_id,
                        queries=queries_amount,
                    )
                remaining_queries = subscription.remaining_queries

            return ResponseSuccess.build(
                self.Response(remaining_queries=remaining_queries,
session_id=checkout_session.id)
            )
        except Exception as exc:
            return ResponseFailure.build(exc)

# Файл "app\src\domain\usecases\payments\get_checkout_session.py":

from src.domain.entities.auth import UserAuthInfo
from src.domain.entities.request import RequestModel
from src.domain.entities.response import (
    ResponseFailure,
    ResponseModel,
    ResponseSuccess,
)
from src.domain.interfaces import IUseCase
from src.infrastructure.repositories.stripe_repository import StripeRepository
from starlette.responses import JSONResponse


class GetCheckoutSessionUseCase(IUseCase):
    class Request(RequestModel):
        user: UserAuthInfo
        session_id: str

    class Response(ResponseModel):
        checkout_session: dict

    def __init__(self):
        self.stripe_repository = StripeRepository()

    async def execute(self, request: Request) -> JSONResponse:
        try:
            checkout_session = await
self.stripe_repository.get_checkout_session(request.session_id)
            return ResponseSuccess.build(self.Response(checkout_session=checkout_session))
        except Exception as exc:
            return ResponseFailure.build(exc)

# Файл "app\src\domain\usecases\subscriptions\get_current_subscription.py":

from src.domain.entities.auth import UserAuthInfo
from src.domain.entities.request import RequestModel
from src.domain.entities.response import (
    ResponseFailure,
    ResponseModel,
```

```
    ResponseSuccess,
)
from src.domain.entities.subscription import Subscription
from src.domain.interfaces import IUseCase
from src.infrastructure.repositories import (
    SubscriptionRepository,
    UserRepository,
)
from starlette.responses import JSONResponse
class GetCurrentSubscriptionUseCase(IUseCase):
    class Request(RequestModel):
        user: UserAuthInfo

    class Response(ResponseModel):
        subscription: Subscription | None

    def __init__(self):
        self.user_repository = UserRepository()
        self.subscription_repository = SubscriptionRepository()

    async def execute(self, request: Request) -> JSONResponse:
        try:
            async with self.user_repository as repository:
                user = await repository.get_user_by_username(username=request.user.sub_id)
                user_id = user.id

            subscription = None
            async with self.subscription_repository as repository:
                current_subscription = await repository.get_one_by_user_id(user_id=user_id)
                if current_subscription:
                    subscription = Subscription(
                        subscription_id=current_subscription.id,
                        user_id=current_subscription.id,
                        remaining_queries=current_subscription.remaining_queries,
                    )

            return ResponseSuccess.build(self.Response(subscription=subscription))
        except Exception as exc:
            return ResponseFailure.build(exc)

# Файл "app\src\domain\usecases\subscriptions\get_limited_subscriptions.py":

from src.domain.entities.auth import UserAuthInfo
from src.domain.entities.request import RequestModel
from src.domain.entities.response import (
    ResponseFailure,
    ResponseModel,
    ResponseSuccess,
)
from src.domain.interfaces import IUseCase
from src.infrastructure.configs import StripeConfig
from src.infrastructure.repositories.stripe_repository import StripeRepository
from starlette.responses import JSONResponse


class GetLimitedSubscriptionsUseCase(IUseCase):
    class Request(RequestModel):
        user: UserAuthInfo

    class Response(ResponseModel):
        object: str
        data: list[dict]
        has_more: bool
        public_key: str

    def __init__(self):
        self.stripe_repository = StripeRepository()
        self.stripe_config = StripeConfig()

    async def execute(self, request: Request) -> JSONResponse:
        try:
            products = await self.stripe_repository.get_products()
            return ResponseSuccess.build(self.Response(**products,
public_key=self.stripe_config.PUBLIC_KEY))
        except Exception as exc:
            return ResponseFailure.build(exc)

# Файл "app\src\infrastructure\api\v0\animation.py":

from typing import Annotated
```

```
from fastapi import Depends, File, UploadFile
from src.domain.entities.auth import UserAuthInfo
from src.domain.usecases.animation import AnimateImageByVideoUseCase
from src.infrastructure.api import APIRouter
from src.infrastructure.services.security.jwt_bearer import JWTBearer

router = APIRouter()


@router.post("/video")
async def animate_image_by_video(
    use_case: Annotated[AnimateImageByVideoUseCase, Depends(AnimateImageByVideoUseCase)],
    user: Annotated[UserAuthInfo, Depends(JWTBearer(auto_error=True))],
    source_image: UploadFile = File(...),
    driving_video: UploadFile = File(...),
):
    return await use_case.execute(use_case.Request(user=user, source_image=source_image,
driving_video=driving_video))

# Файл "app\src\infrastructure\api\v0\auth.py":

from typing import Annotated

from fastapi import Body, Depends
from pydantic import SecretStr
from src.domain.usecases.auth import UserLoginUseCase, UserSignUpUseCase
from src.infrastructure.api import APIRouter

router = APIRouter()


@router.post("/login")
async def login(
    use_case: Annotated[UserLoginUseCase, Depends(UserLoginUseCase)],
    username: str = Body(..., min_length=3),
    password: SecretStr = Body(..., min_length=8),
):
    return await use_case.execute(use_case.Request(username=username,
password=password.get_secret_value()))


@router.post("/signup")
async def signup(
    use_case: Annotated[UserSignUpUseCase, Depends(UserSignUpUseCase)],
    username: str = Body(..., min_length=3),
    password: SecretStr = Body(..., min_length=8),
):
    return await use_case.execute(use_case.Request(username=username,
password=password.get_secret_value()))

# Файл "app\src\infrastructure\api\v0\payments.py":

from typing import Annotated

from fastapi import Body, Depends, Query
from src.domain.entities.auth import UserAuthInfo
from src.domain.usecases.payments import (
    CreateCheckoutSessionUseCase,
    CreatePaymentUseCase,
    GetCheckoutSessionUseCase,
)
from src.infrastructure.api import APIRouter
from src.infrastructure.services.security.jwt_bearer import JWTBearer

router = APIRouter()


@router.get("/checkout-session")
async def get_checkout_session(
    use_case: Annotated[GetCheckoutSessionUseCase, Depends(GetCheckoutSessionUseCase)],
    user: Annotated[UserAuthInfo, Depends(JWTBearer(auto_error=True))],
    session_id: str = Query(..., description="Session ID of Stripe checkout session"),
):
    return await use_case.execute(use_case.Request(user=user, session_id=session_id))
@router.post("/checkout-session")
async def create_checkout_session(
    use_case: Annotated[CreateCheckoutSessionUseCase, Depends(CreateCheckoutSessionUseCase)],
    user: Annotated[UserAuthInfo, Depends(JWTBearer(auto_error=True))],
    price_id: str = Query(..., description="Price ID of Stripe checkout session"),
):
```

```
    return await use_case.execute(use_case.Request(user=user, price_id=price_id))


@router.post("")
async def create_payment(
    use_case: Annotated[CreatePaymentUseCase, Depends(CreatePaymentUseCase)],
    user: Annotated[UserAuthInfo, Depends(JWTBearer(auto_error=True))],
    session_id: str = Body(..., description="Session ID of Stripe checkout session"),
    product_id: str = Body(..., description="Payment Stripe Product ID"),
):
    return await use_case.execute(use_case.Request(user=user, session_id=session_id,
product_id=product_id))

# Файл "app\src\infrastructure\api\v0\subscriptions.py":

from typing import Annotated

from fastapi import Depends
from src.domain.entities.auth import UserAuthInfo
from src.domain.usecases.subscriptions import (
    GetCurrentSubscriptionUseCase,
    GetLimitedSubscriptionsUseCase,
)
from src.infrastructure.api import APIRouter
from src.infrastructure.services.security.jwt_bearer import JWTBearer

router = APIRouter()


@router.get("/current")
async def get_current_subscription(
    use_case: Annotated[GetCurrentSubscriptionUseCase, Depends(GetCurrentSubscriptionUseCase)],
    user: Annotated[UserAuthInfo, Depends(JWTBearer(auto_error=True))],
):
    return await use_case.execute(use_case.Request(user=user))


@router.get("/limited")
async def get_limited_subscriptions(
    use_case: Annotated[GetLimitedSubscriptionsUseCase, Depends(GetLimitedSubscriptionsUseCase)],
    user: Annotated[UserAuthInfo, Depends(JWTBearer(auto_error=True))],
):
    return await use_case.execute(use_case.Request(user=user))

# Файл "app\src\infrastructure\models\models.py":

from sqlalchemy import ForeignKey, Integer, String
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column


class Base(DeclarativeBase):
    pass


class User(Base):
    __tablename__ = "user"

    id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
    username: Mapped[str] = mapped_column(String(255), unique=True, nullable=False, index=True)
    hashed_password: Mapped[str] = mapped_column(String(255), nullable=False)


class Subscription(Base):
    __tablename__ = "subscription"

    id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
    user_id: Mapped[int] = mapped_column(ForeignKey("user.id"), nullable=False)
    remaining_queries: Mapped[int] = mapped_column(Integer, default=1, nullable=False)

# Файл "app\src\infrastructure\repositories\stripe_repository.py":

import asyncio

import stripe
from stripe import ListObject, Product
from stripe.checkout import Session


class StripeRepository:
    @classmethod
```

```
    async def get_products(cls) -> ListObject[Product]:
        params = {"active": True, "expand": ["data.default_price"]}
        products = await asyncio.to_thread(stripe.Product.list, **params)
        return products

    @classmethod
    async def get_product(cls, product_id: str) -> Product:
        product = await asyncio.to_thread(stripe.Product.retrieve, product_id)
        return product

    @classmethod
    async def get_checkout_session(cls, session_id: str) -> Session:
        checkout_session = await asyncio.to_thread(stripe.checkout.Session.retrieve, session_id)
        return checkout_session

    @classmethod
    async def create_checkout_session(cls, params: dict) -> Session:
        checkout_session = await asyncio.to_thread(stripe.checkout.Session.create, **params)
        return checkout_session

# Файл "app\src\infrastructure\repositories\subscription_repository.py":

from sqlalchemy import select
from src.domain.interfaces import IDBRepository
from src.infrastructure.models.models import Subscription


class SubscriptionRepository(IDBRepository):
    async def get_one_by_user_id(self, user_id: int) -> Subscription | None:
        stmt = select(Subscription).where(Subscription.user_id == user_id)
        result = await self.async_session.execute(stmt)
        return result.scalar_one_or_none()

    async def add_one(self, user_id: int, queries: int) -> Subscription:
        subscription = Subscription(user_id=user_id, remaining_queries=queries)
        self.async_session.add(subscription)
        await self.async_session.flush()
        return subscription

    async def update_one(self, existing_subscription: Subscription, queries: int) ->
Subscription:
        existing_subscription.remaining_queries += queries
        await self.async_session.flush()
        return existing_subscription

# Файл "app\src\infrastructure\repositories\user_repository.py":

from sqlalchemy import select
from src.domain.interfaces import IDBRepository
from src.infrastructure.models.models import User
from src.infrastructure.services.security.password import PasswordService


class UserRepository(IDBRepository):
    async def create_user(self, username: str, password: str) -> None:
        user = User(
            username=username,
            hashed_password=PasswordService.hash_password(password),
        )
        self.async_session.add(user)

    async def get_user_by_username(self, username: str) -> User | None:
        query = select(User).where(User.username == username)
        result = await self.async_session.execute(query)
        return result.scalars().first()

# Файл "app\src\infrastructure\services\auth\auth.py":

from src.domain.entities.user import UserRead
from src.infrastructure.repositories import UserRepository
from src.infrastructure.services.security.password import PasswordService


class AuthService:
    def __init__(self) -> None:
        self.user_repository = UserRepository()
        self.password_service = PasswordService()

    async def authenticate_user(self, username: str, password: str) -> UserRead | None:
        async with self.user_repository as repository:
```

```
            user = await repository.get_user_by_username(username)

            if not user or not self.password_service.verify_password(password,
user.hashed_password):
                return None

            return UserRead.model_validate(user)

# Файл "app\src\infrastructure\services\auth\token.py":

from datetime import datetime, timedelta, timezone

import jwt
from src.domain.entities.enums import TokenType
from src.infrastructure.configs import AuthConfig


class TokenService:
    def __init__(self):
        self.auth_config = AuthConfig()

    async def generate_tokens(self, username: str) -> dict:
        """Generate access and refresh tokens"""
        access_token_expires = timedelta(minutes=self.auth_config.ACCESS_TOKEN_EXPIRE_MINUTES)
        refresh_token_expires = timedelta(days=self.auth_config.REFRESH_TOKEN_EXPIRE_DAYS)

        access_token = self._create_token(
            data={"sub": username},  # username as unique value
            token_type=TokenType.ACCESS,
            expires_delta=access_token_expires,
        )

        refresh_token = self._create_token(
            data={"sub": username}, token_type=TokenType.REFRESH,
expires_delta=refresh_token_expires
        )

        return {
            "access_token": access_token,
            "refresh_token": refresh_token,
            "access_expires_in": int(access_token_expires.total_seconds()),
            "refresh_expires_in": int(refresh_token_expires.total_seconds()),
        }

    def verify_token(self, token: str) -> dict:
        """Verify JWT token and return payload"""
        return jwt.decode(
            token, self.auth_config.SECRET_KEY, algorithms=[self.auth_config.ALGORITHM],
options={"verify_exp": True}
        )

    def _create_token(self, data: dict, token_type: TokenType, expires_delta: timedelta | None =
None) -> str:
        """Base token creation method"""
        to_encode = data.copy()

        expire = datetime.now(timezone.utc) + (
            expires_delta if expires_delta else
timedelta(minutes=self.auth_config.ACCESS_TOKEN_EXPIRE_MINUTES)
        )

        to_encode.update({"exp": expire, "type": token_type, "iss":
self.auth_config.TOKEN_ISSUER})

        return jwt.encode(payload=to_encode, key=self.auth_config.SECRET_KEY,
algorithm=self.auth_config.ALGORITHM)

# Файл "app\src\infrastructure\services\security\jwt_bearer.py":

import jwt
from fastapi import Request, status
from fastapi.security import HTTPBearer
from src.domain.entities.auth import UserAuthInfo
from src.infrastructure.configs import AuthConfig
from src.infrastructure.exceptions.exceptions import ExpiredTokenException
from src.infrastructure.services.auth.token import TokenService


class JWTBearer(HTTPBearer):
    def __init__(self, auto_error: bool = True):
```

```
        super().__init__(auto_error=auto_error)
        self.token_service = TokenService()
        self.config = AuthConfig()

    async def __call__(self, request: Request = None) -> UserAuthInfo | None:
        credentials = await super().__call__(request)
        if not credentials:
            return None

        try:
            payload = self.token_service.verify_token(credentials.credentials)
            return UserAuthInfo(sub_id=payload["sub"])
        except jwt.PyJWTError:
            if self.auto_error:
                raise ExpiredTokenException("Invalid or expired token",
status.HTTP_401_UNAUTHORIZED)
            return None

# Файл "app\src\infrastructure\services\security\password.py":

from passlib.context import CryptContext

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")


class PasswordService:
    @staticmethod
    def hash_password(password: str) -> str:
        return pwd_context.hash(password)

    @staticmethod
    def verify_password(plain_password: str, hashed_password: str) -> bool:
        return pwd_context.verify(plain_password, hashed_password)

# Файл "app\src\main.py":

import streamlit as st
from services.auth import AuthService
from services.payment import PaymentService
from services.resource import ResourceService
from views import about_page, auth_page, home_page, pricing_page

ResourceService.load_styles("sidebar.css")
credit_card_template = ResourceService.load_template("sidebar/credit_card.html")


def render_sidebar():
    with st.sidebar:
        st.logo("src/assets/images/animatica_logo.png", size="large")

        if st.session_state.logged_in:
            remaining_queries = PaymentService.get_query_balance()
            st.markdown(credit_card_template.format(credits=remaining_queries),
unsafe_allow_html=True)

            if st.button(
                "Logout",
                help="Logout from your account",
                type="primary",
                use_container_width=True,
                key="sidebar_logout",
                icon=":material/logout:",
            ):
                AuthService.logout()

        st.markdown("---")
        st.caption("Made with ❤️ by [kefirchk](https://github.com/kefirchk)")


def main():
    AuthService.check_auth()

    if st.session_state.logged_in:
        pg = st.navigation(pages=[home_page, pricing_page, about_page])
        pg.run()
    else:
        st.navigation(pages=[auth_page])
        auth_page.run()

    render_sidebar()
```

```
if __name__ == "__main__":
    main()

# Файл "app\src\renders\auth.py":

import streamlit as st
from configs import AuthConfig
from services.auth import AuthService


class AuthRender:
    def __init__(self, templates: dict):
        self.templates = templates
        self.auth_service = AuthService()
        self.auth_config = AuthConfig()
        self._init_session_state()

    @staticmethod
    def _init_session_state():
        st.session_state.setdefault("login_error", "")
        st.session_state.setdefault("register_error", "")
        st.session_state.setdefault("register_success", "")

    def render_login_form(self):
        with st.form("login_form"):
            st.markdown(self.templates["login_column"], unsafe_allow_html=True)

            username = st.text_input("Username", key="login_username")
            password = st.text_input("Password", type="password", key="login_password")

            if st.form_submit_button("Login", type="primary", use_container_width=True):
                if self.auth_service.login(username, password):
                    st.rerun()

            if st.session_state.login_error:
                st.error(st.session_state.login_error)

            st.markdown("</div>", unsafe_allow_html=True)

    def render_register_form(self):
        with st.form("register_form"):
            st.markdown(self.templates["register_column"], unsafe_allow_html=True)

            new_username = st.text_input("Username", key="reg_username")
            new_password = st.text_input("Password", type="password", key="reg_password")
            confirm_password = st.text_input("Confirm Password", type="password",
key="reg_confirm_password")

            tos_accepted = st.checkbox(
                "I agree to the Terms of Service",
                key="tos_checkbox",
                help="You must accept the Terms of Service to register",
            )

            with st.expander("View Terms of Service", icon=":material/description:"):
                st.markdown(self.auth_config.TERMS_OF_SERVICE, unsafe_allow_html=True)

            if st.form_submit_button("Register", type="primary", use_container_width=True):
                if not tos_accepted:
                    st.session_state.register_error = "You must accept the Terms of Service"
                elif self.auth_service.register(new_username, new_password, confirm_password):
                    st.rerun()

            if st.session_state.register_error:
                st.error(st.session_state.register_error)
            elif st.session_state.register_success:
                st.success(st.session_state.register_success)

            st.markdown("</div>", unsafe_allow_html=True)

# Файл "app\src\renders\home.py":

import base64
import time

import numpy as np
import streamlit as st
from configs import HomeConfig
from PIL import Image
from renders.utils import centered_button, resize_and_center
```

```
from services.animation import AnimationService
from services.cookie import CookieService
from services.payment import PaymentService


class HomeRender:
    def __init__(self, templates: dict):
        self.templates = templates
        self.animation_service = AnimationService()
        self.home_config = HomeConfig()
        query_balance = PaymentService.get_query_balance()
        CookieService.controller.set("query_balance", query_balance, max_age=31556925)
        self._init_session_state()

    @staticmethod
    def _init_session_state():
        st.session_state.setdefault("animation_result", None)
        st.session_state.setdefault("source_image", None)
        st.session_state.setdefault("driving_video", None)

    @centered_button()
    def render_subscription_required(self):
        st.markdown(self.templates["subscription"], unsafe_allow_html=True)
        if st.button("View Plans", type="primary", help="See subscription options",
use_container_width=True):
            st.switch_page("views/pricing.py")

    def render_image_upload(self, target_width: int = 320):
        uploaded_image = st.file_uploader(
            "Upload source image",
            type=self.home_config.ALLOWED_IMAGE_TYPES,
            key="source_uploader",
            label_visibility="collapsed",
        )
        if uploaded_image:
            st.session_state.source_image = uploaded_image.read()
            img_np = np.array(Image.open(uploaded_image).convert("RGB"))
            resized = resize_and_center(img_np)
            st.image(resized, width=target_width, caption="Source Image")

    def render_video_upload(self):
        uploaded_video = st.file_uploader(
            "Upload driving video",
            type=self.home_config.ALLOWED_VIDEO_TYPES,
            key="video_uploader",
            label_visibility="collapsed",
        )
        if uploaded_video:
            st.session_state.driving_video = uploaded_video.read()
            st.markdown(
                self.templates["driving_video"].format(
                    video_b64=base64.b64encode(st.session_state.driving_video).decode(),
                    target_width=320,
                    target_height=240,
                ),
                unsafe_allow_html=True,
            )

    @centered_button(3)
    def render_generate_button(self):
        if st.button(
            "Generate Animation",
            type="primary",
            use_container_width=True,
            disabled=not st.session_state.source_image or not st.session_state.driving_video,
            icon=":material/animated_images:",
        ):
            self._process_generation_request()

    def _process_generation_request(self):
        if not st.session_state.source_image or not st.session_state.driving_video:
            st.error("Please provide both source image and driving video")
            return

        with st.spinner("⏳ Generating animation... Please wait."):
            response = self.animation_service.generate_animation(
                {
                    "source_image": ("source.jpg", st.session_state.source_image, "image/jpeg"),
                    "driving_video": ("driving.mp4", st.session_state.driving_video,
"video/mp4"),
```

```
                }
            )

        st.session_state.source_image = None
        st.session_state.driving_video = None

        if response["success"]:
            st.session_state.animation_result = response["animation_data"]
            st.success("✅ Animation generated successfully!")
            time.sleep(3)
            st.rerun()
        else:
            st.error(f"❌ Animation generation failed: {response['message']}")

    @centered_button(3)
    def render_download_button(self):
        st.download_button(
            label="Download Video",
            type="primary",
            data=st.session_state.animation_result,
            file_name="generated_animation.mp4",
            mime="video/mp4",
            use_container_width=True,
            icon=":material/download:",
        )

# Файл "app\src\renders\pricing.py":

import time

import streamlit as st
from configs import APIConfig
from configs.pricing import PricingConfig
from services.auth import AuthService
from services.cookie import CookieService
from services.payment import PaymentService
from streamlit.delta_generator import DeltaGenerator


class PricingRender:
    def __init__(self, templates: dict) -> None:
        self.templates = templates
        self.auth_service = AuthService()
        self.payment_service = PaymentService()
        self.cookie = CookieService.controller
        self.api_config = APIConfig()
        self.pricing_config = PricingConfig()
        self._init_session_state()
        self._handle_payment_callback()
        self.stripe_public_key, self.products = self._load_products()

    @staticmethod
    def _init_session_state() -> None:
        st.session_state.setdefault("selected_product_id", None)
        st.session_state.setdefault("selected_price_id", None)

    def _handle_payment_callback(self) -> None:
        def reload():
            st.query_params.clear()
            time.sleep(self.pricing_config.MESSAGE_DELAY)
            st.rerun()

        if st.query_params.get("success") == "true":
            st.success("✅ Payment was successful! Thank you for your purchase!")
            if self.cookie.get("session_id") and self.cookie.get("product_id"):
                self.payment_service.make_payment()
                reload()

        elif st.query_params.get("canceled") == "true":
            st.warning("❌ Payment canceled")
            reload()

    def _load_products(self) -> tuple[str, list]:
        response = self.auth_service.make_authenticated_request(
            "GET", f"{self.api_config.BASE_URL}/api/v0/subscriptions/limited"
        )
        data = response.json()
        return data.get("public_key"), data.get("data", [])
    def _select_product(self, product_id: str, price_id: str) -> None:
```

```
        st.session_state.selected_product_id = product_id
        st.session_state.selected_price_id = price_id
        self.cookie.set("price_id", price_id, max_age=31556925)

    def render_plan_card(self, product: dict, col: DeltaGenerator) -> None:
        with col:
            product_id = product["id"]
            price_id = product["default_price"]["id"]
            is_selected = st.session_state.selected_product_id == product_id

            st.button(
                product["name"].removeprefix("Animatica "),
                key=f"plan_btn_{product_id}",
                type="primary" if is_selected else "secondary",
                use_container_width=True,
                on_click=self._select_product,
                args=(product_id, price_id),
            )

            features_html = "".join(
                f'<li class="plan-feature">{feature["name"]}</li>' for feature in
product["marketing_features"]
            )

            st.markdown(
                self.templates["plan_card"].format(
                    card_class="plan-card selected" if is_selected else "plan-card",
                    plan_name=product["name"].removeprefix("Animatica "),
                    price=f"${product['default_price']['unit_amount'] / 100}",
                    discount=(
                        f"Discount {product['default_price']['metadata'].get('discount', '')}%"
                        if product["default_price"]["metadata"].get("discount")
                        else ""
                    ),
                    features=features_html,
                ),
                unsafe_allow_html=True,
            )

    def render_subscribe_button(self, selected_product: dict) -> None:
        st.markdown(
            self.templates["selection"].format(selected_plan=selected_product["name"]),
            unsafe_allow_html=True,
        )

        response = self.auth_service.make_authenticated_request(
            "POST",
            f"{self.api_config.BASE_URL}/api/v0/payments/checkout-session",
            params={"price_id": st.session_state.selected_price_id},
        )
        self.cookie.set("product_id", st.session_state.selected_product_id, max_age=31556925)

        if response.status_code == 200:
            checkout_session = response.json()["checkout_session"]
            template = self.templates["checkout_button"].format(url=checkout_session["url"])
            st.markdown(template, unsafe_allow_html=True)
            self.cookie.set("session_id", checkout_session["id"], max_age=60)

        elif response.status_code == 401:
            self.auth_service.logout()
            st.switch_page("views/auth.py")

        else:
            error_msg = response.json().get("detail", "Unknown payment error")
            st.error(f"Payment error: {error_msg}")

# Файл "app\src\services\animation.py":

from configs import APIConfig
from services.auth import AuthService


class AnimationService:
    def __init__(self) -> None:
        self.api_url = APIConfig().BASE_URL
        self.auth_service = AuthService()

    def generate_animation(self, files: dict) -> dict:
        try:
            response = self.auth_service.make_authenticated_request(
```

```
                "POST",
                f"{self.api_url}/api/v0/animation/video",
                files=files,
                timeout=1200,
            )
            response.raise_for_status()
            return {"success": True, "animation_data": response.content, "message": "OK"}
        except Exception as e:
            return {"success": False, "animation_data": None, "message": str(e)}

# Файл "app\src\services\auth.py":

import requests
import streamlit as st
from configs.api import APIConfig
from services.cookie import CookieService


class AuthService:
    api_config = APIConfig()

    @staticmethod
    def check_auth():
        if "logged_in" not in st.session_state:
            auth_data = CookieService.get_auth_cookies()
            if auth_data:
                try:
                    st.session_state.update(
                        {
                            "logged_in": True,
                            "username": auth_data.get("username", ""),
                            "access_token": auth_data["access_token"],
                            "refresh_token": auth_data["refresh_token"],
                        }
                    )
                except:
                    CookieService.clear_auth_cookies()
                    st.session_state.logged_in = False
            else:
                st.session_state.logged_in = False

    @classmethod
    def login(cls, username: str, password: str) -> bool:
        try:
            response = requests.post(
                f"{cls.api_config.BASE_URL}/api/v0/auth/login", json={"username": username,
"password": password}
            )

            if response.status_code == 200:
                token_data = response.json()
                token_data["username"] = username
                st.session_state.logged_in = True
                st.session_state.username = username
                CookieService.set_auth_cookies(token_data)
                st.session_state.login_error = ""
                return True
            else:
                error_msg = (
                    response.json()["detail"][0]["msg"] if isinstance(response.json(), dict) else
response.json()
                )
                st.session_state.login_error = error_msg
                return False
        except requests.exceptions.RequestException as e:
            st.session_state.login_error = f"Connection error: {str(e)}"
            return False

    @classmethod
    def register(cls, username: str, password: str, confirm_password: str) -> bool:
        if password != confirm_password:
            st.session_state.register_error = "Passwords don't match"
            return False

        try:
            response = requests.post(
                f"{cls.api_config.BASE_URL}/api/v0/auth/signup", json={"username": username,
"password": password}
            )
            if response.status_code == 201:
```

```
                st.session_state.register_error = ""
                st.session_state.register_success = "Registration successful! Please login."
                return True
            else:
                error_msg = (
                    response.json()["detail"][0]["msg"] if isinstance(response.json(), dict) else
response.json()
                )
                st.session_state.register_error = error_msg
                return False
        except requests.exceptions.RequestException as e:
            st.session_state.register_error = f"Connection error: {str(e)}"
            return False

    @staticmethod
    def logout():
        st.session_state.logged_in = False
        CookieService.clear_auth_cookies()
        st.rerun()

    @staticmethod
    def make_authenticated_request(method: str, url: str, **kwargs):
        headers = CookieService.get_auth_headers()
        if "headers" in kwargs:
            headers.update(kwargs["headers"])
        kwargs["headers"] = headers

        try:
            response = requests.request(method, url, **kwargs)
            if response.status_code == 401:
                CookieService.clear_auth_cookies()
                st.session_state.logged_in = False
            return response
        except requests.exceptions.RequestException as e:
            st.error(f"Connection error: {str(e)}")
            return None

# Файл "app\src\services\cookie.py":

from streamlit_cookies_controller import CookieController


class CookieService:
    controller = CookieController()

    @classmethod
    def set_auth_cookies(cls, token_data: dict):
        cls.controller.set("auth_data", token_data)

    @classmethod
    def get_auth_cookies(cls) -> dict | None:
        auth_data = cls.controller.get("auth_data")
        return auth_data

    @classmethod
    def clear_auth_cookies(cls):
        if cls.controller.get("auth_data"):
            cls.controller.remove("auth_data")
        if cls.controller.get("query_balance"):
            cls.controller.remove("query_balance")

    @classmethod
    def get_auth_headers(cls) -> dict:
        auth_data = cls.controller.get("auth_data")
        if auth_data and "access_token" in auth_data:
            return {"Authorization": f'Bearer {auth_data["access_token"]}'}
        return {}

# Файл "app\src\services\payment.py":

import time

import streamlit as st
from configs.api import APIConfig
from configs.pricing import PricingConfig
from services.auth import AuthService
from services.cookie import CookieService


class PaymentService:
```

```
    api_config = APIConfig()
    pricing_config = PricingConfig()

    @classmethod
    def get_query_balance(cls):
        current_subscription_response = AuthService.make_authenticated_request(
            "GET",
            f"{cls.api_config.BASE_URL}/api/v0/subscriptions/current",
        )
        if current_subscription_response.status_code == 200:
            subscription = current_subscription_response.json()["subscription"]
            remaining_queries = subscription.get("remaining_queries", 0) if subscription else 0
            return remaining_queries
        else:
            st.error("Can't get remaining queries!")
            AuthService.logout()

    @classmethod
    def make_payment(cls):
        response = AuthService.make_authenticated_request(
            "POST",
            f"{cls.api_config.BASE_URL}/api/v0/payments",
            json={
                "session_id": CookieService.controller.get("session_id"),
                "product_id": CookieService.controller.get("product_id"),
            },
        )
        if response.status_code == 200:
            data = response.json()
            CookieService.controller.set("query_balance", data["remaining_queries"],
max_age=31556925)
        else:
            st.error(f"Payment failed: {response.json()}")
            time.sleep(cls.pricing_config.MESSAGE_DELAY)

# Файл "app\src\services\resource.py":

import base64
from pathlib import Path

import streamlit as st


class ResourceService:
    """Loader of application resources (styles, templates, images)"""

    @staticmethod
    def load_styles(style_name: str):
        css_path = Path(__file__).parent.parent / "assets" / "styles" / style_name
        with open(css_path, "r", encoding="utf-8") as f:
            st.markdown(f"<style>{f.read()}</style>", unsafe_allow_html=True)

    @staticmethod
    def load_template(template_name: str):
        template_path = Path(__file__).parent.parent / "templates" / template_name
        with open(template_path, "r", encoding="utf-8") as f:
            return f.read()

    @staticmethod
    def load_image(image_name: str):
        image_path = Path(__file__).parent.parent / "assets" / "images" / image_name
        with open(image_path, "rb") as f:
            return base64.b64encode(f.read()).decode()

# Файл "app\src\run_model.py":

import sys
from argparse import ArgumentParser, Namespace

import torch
from src.ml.runners import (
    AnimationRunner,
    ReconstructionRunner,
    TrainingRunner,
)
from src.ml.services.logging import LoggingService


def parse_args() -> Namespace:
    parser = ArgumentParser()
```

```
    parser.add_argument("--configs", required=True)
    parser.add_argument("--mode", choices=["train", "reconstruction", "animate"], required=True)
    parser.add_argument("--log_dir", default="log")
    parser.add_argument("--checkpoint", default=None)
    parser.add_argument("--device_ids", default="0", type=lambda x: list(map(int, x.split(","))))
    parser.add_argument("--verbose", action="store_true")
    return parser.parse_args()


if __name__ == "__main__":
    if sys.version_info[0] < 3:
        raise Exception("Use Python 3 or higher.")

    torch.multiprocessing.set_start_method("spawn", force=True)  # Safety for CUDA

    args = parse_args()
    log = LoggingService.setup_logger(__name__)

    mode_to_runner = {"train": TrainingRunner, "reconstruction": ReconstructionRunner, "animate":
AnimationRunner}

    runner_class = mode_to_runner[args.mode]
    runner = runner_class(args, log)
    runner.run()

# Файл "app\src\run_server.py":

import logging

import src.server as server
from fastapi import Depends, FastAPI
from src.server.config import APIConfig, LogLevelEnum
from src.server.exceptions import app_exception_handler, exception_handlers
from src.server.routers import router as fomm_router

logging.basicConfig(format="[PID:%(process)d] %(pathname)s:%(lineno)d %(message)s",
level=logging.INFO)
config = APIConfig()

app = FastAPI(
    title=f"{config.MODE.capitalize()} Animatica ML API",
    description="This API is designed for the Animatica application.",
    swagger_ui_parameters={"displayRequestDuration": True},
    version=server.__version__,
    debug=(config.LOG_LEVEL == LogLevelEnum.DEBUG),
)

app.include_router(fomm_router, prefix="/api", dependencies=[Depends(server.validate_api_key)])

for exc_type in exception_handlers:
    app.add_exception_handler(exc_type, app_exception_handler)

# Файл "app\src\ml\datasets\dataset_repeater.py":

from torch.utils.data import Dataset


class DatasetRepeater(Dataset):
    def __init__(self, dataset, num_repeats: int = 100) -> None:
        self.dataset = dataset
        self.num_repeats = num_repeats

    def __len__(self) -> int:
        return self.num_repeats * self.dataset.__len__()

    def __getitem__(self, idx: int):
        return self.dataset[idx % self.dataset.__len__()]

# Файл "app\src\ml\datasets\frames_dataset.py":

import glob
import os

import numpy as np
from skimage import io
from skimage.util import img_as_float32
from sklearn.model_selection import train_test_split
from src.ml.datasets.augmentation import AllAugmentationTransform
from src.ml.datasets.utils import read_video
from torch.utils.data import Dataset
```

```
class FramesDataset(Dataset):
    def __init__(
        self,
        root_dir: str,
        frame_shape=(256, 256, 3),
        id_sampling: bool = False,
        is_train: bool = True,
        random_seed=0,
        pairs_list=None,
        augmentation_params=None,
    ) -> None:
        self.root_dir = root_dir
        self.videos = os.listdir(root_dir)
        self.frame_shape = tuple(frame_shape)
        self.pairs_list = pairs_list
        self.id_sampling = id_sampling
        if os.path.exists(os.path.join(root_dir, "train")):
            assert os.path.exists(os.path.join(root_dir, "test"))
            print("Use predefined train-test split.")
            if id_sampling:
                train_videos = {
                    os.path.basename(video).split("#")[0] for video in
os.listdir(os.path.join(root_dir, "train"))
                }
                train_videos = list(train_videos)
            else:
                train_videos = os.listdir(os.path.join(root_dir, "train"))
            test_videos = os.listdir(os.path.join(root_dir, "test"))
            self.root_dir = os.path.join(self.root_dir, "train" if is_train else "test")
        else:
            print("Use random train-test split.")
            train_videos, test_videos = train_test_split(self.videos, random_state=random_seed,
test_size=0.2)

        self.videos = train_videos if is_train else test_videos
        self.transform = AllAugmentationTransform(**augmentation_params) if is_train else None
        self.is_train = is_train

    def __len__(self) -> int:
        return len(self.videos)

    def __getitem__(self, idx: int):
        if self.is_train and self.id_sampling:
            name = self.videos[idx]
            path = np.random.choice(glob.glob(os.path.join(self.root_dir, name + "*.mp4")))
        else:
            name = self.videos[idx]
            path = os.path.join(self.root_dir, name)

        video_name = os.path.basename(path)

        if self.is_train and os.path.isdir(path):
            frames = os.listdir(path)
            num_frames = len(frames)
            frame_idx = np.sort(np.random.choice(num_frames, replace=True, size=2))
            video_array = [img_as_float32(io.imread(os.path.join(path, frames[idx]))) for idx in
frame_idx]
        else:
            video_array = read_video(path, frame_shape=self.frame_shape)
            num_frames = len(video_array)
            frame_idx = (
                np.sort(np.random.choice(num_frames, replace=True, size=2)) if self.is_train else
range(num_frames)
            )
            video_array = video_array[frame_idx]

        if self.transform is not None:
            video_array = self.transform(video_array)

        out = {}
        if self.is_train:
            source = np.array(video_array[0], dtype="float32")
            driving = np.array(video_array[1], dtype="float32")

            out["driving"] = driving.transpose((2, 0, 1))
            out["source"] = source.transpose((2, 0, 1))
        else:
            video = np.array(video_array, dtype="float32")
            out["video"] = video.transpose((3, 0, 1, 2))
```

```
        out["name"] = video_name
        return out

# Файл "app\src\ml\datasets\paired_dataset.py":

import numpy as np
import pandas as pd
from torch.utils.data import Dataset


class PairedDataset(Dataset):
    def __init__(self, initial_dataset, number_of_pairs: int, seed=0) -> None:
        self.initial_dataset = initial_dataset
        pairs_list = self.initial_dataset.pairs_list

        np.random.seed(seed)

        if not pairs_list:
            max_idx = min(number_of_pairs, len(initial_dataset))
            nx, ny = max_idx, max_idx
            xy = np.mgrid[:nx, :ny].reshape(2, -1).T
            number_of_pairs = min(xy.shape[0], number_of_pairs)
            self.pairs = xy.take(np.random.choice(xy.shape[0], number_of_pairs, replace=False),
axis=0)
        else:
            videos = self.initial_dataset.videos
            name_to_index = {name: index for index, name in enumerate(videos)}
            pairs = pd.read_csv(pairs_list)
            pairs = pairs[np.logical_and(pairs["source"].isin(videos),
pairs["driving"].isin(videos))]

            number_of_pairs = min(pairs.shape[0], number_of_pairs)
            self.pairs = []
            self.start_frames = []
            for ind in range(number_of_pairs):
                self.pairs.append((name_to_index[pairs["driving"].iloc[ind]],
name_to_index[pairs["source"].iloc[ind]]))

    def __len__(self) -> int:
        return len(self.pairs)

    def __getitem__(self, idx: int):
        pair = self.pairs[idx]
        first = self.initial_dataset[pair[0]]
        second = self.initial_dataset[pair[1]]
        first = {"driving_" + key: value for key, value in first.items()}
        second = {"source_" + key: value for key, value in second.items()}

        return {**first, **second}

# Файл "app\src\ml\datasets\augmentation\all_transforms.py":

from src.ml.datasets.augmentation.transforms import (
    ColorJitter,
    RandomCrop,
    RandomFlip,
    RandomResize,
    RandomRotation,
)


class AllAugmentationTransform:
    def __init__(
        self, resize_param=None, rotation_param=None, flip_param=None, crop_param=None,
jitter_param=None
    ) -> None:
        self.transforms = []

        params = {
            RandomFlip: flip_param,
            RandomRotation: rotation_param,
            RandomResize: resize_param,
            RandomCrop: crop_param,
            ColorJitter: jitter_param,
        }
        for Factory, param in params.items():
            if param:
                self.transforms.append(Factory(**param))

    def __call__(self, clip):
```

```
        for t in self.transforms:
            clip = t(clip)
        return clip

# Файл "app\src\ml\datasets\augmentation\transforms\color_jitter.py":

import random

import numpy as np
import torchvision
from skimage.util import img_as_float, img_as_ubyte
from src.ml.datasets.augmentation.utils import is_numpy_clip, is_pil_clip


class ColorJitter:
    def __init__(self, brightness: float = 0, contrast: float = 0, saturation: float = 0, hue:
float = 0) -> None:
        self.brightness = brightness
        self.contrast = contrast
        self.saturation = saturation
        self.hue = hue

    @staticmethod
    def get_params(brightness: float, contrast: float, saturation: float, hue: float):
        return (
            random.uniform(max(0.0, 1 - brightness), 1 + brightness) if brightness > 0 else None,
            random.uniform(max(0.0, 1 - contrast), 1 + contrast) if contrast > 0 else None,
            random.uniform(max(0.0, 1 - saturation), 1 + saturation) if saturation > 0 else None,
            random.uniform(-hue, hue) if hue > 0 else None,
        )

    def __call__(self, clip):
        return self.apply_color_jitter(clip, self.brightness, self.contrast, self.saturation,
self.hue)

    @staticmethod
    def apply_pipeline(img, transforms):
        for t in transforms:
            img = t(img)
        return img

    @classmethod
    def apply_color_jitter(cls, clip, brightness: float, contrast: float, saturation: float, hue:
float) -> list:
        brightness, contrast, saturation, hue = ColorJitter.get_params(brightness, contrast,
saturation, hue)
        transforms = []
        if brightness:
            transforms.append(lambda img:
torchvision.transforms.functional.adjust_brightness(img, brightness))
        if saturation:
            transforms.append(lambda img:
torchvision.transforms.functional.adjust_saturation(img, saturation))
        if hue:
            transforms.append(lambda img: torchvision.transforms.functional.adjust_hue(img, hue))
        if contrast:
            transforms.append(lambda img: torchvision.transforms.functional.adjust_contrast(img,
contrast))
        random.shuffle(transforms)

        if is_numpy_clip(clip):
            img_transforms = [img_as_ubyte, torchvision.transforms.ToPILImage()] + transforms +
[np.array, img_as_float]
            return [cls.apply_pipeline(img, img_transforms) for img in clip]
        elif is_pil_clip(clip):
            return [cls.apply_pipeline(img, transforms) for img in clip]
        else:
            raise TypeError(f"Unsupported clip type: {type(clip[0])}")

# Файл "app\src\ml\datasets\augmentation\transforms\random_crop.py":

import random

from src.ml.datasets.augmentation.utils import (
    crop_clip,
    is_numpy_clip,
    is_pil_clip,
    pad_clip,
)
```

```
class RandomCrop:
    def __init__(self, size: int | tuple[int, int]) -> None:
        self.size = (size, size) if isinstance(size, int) else size

    def __call__(self, clip):
        h, w = self.size
        if is_numpy_clip(clip):
            im_h, im_w, im_c = clip[0].shape
        elif is_pil_clip(clip):
            im_w, im_h = clip[0].size
        else:
            raise TypeError("Expected numpy.ndarray or PIL.Image" + "but got list of
{0}".format(type(clip[0])))

        clip = pad_clip(clip, h, w)
        im_h, im_w = clip.shape[1:3]
        x1 = 0 if h == im_h else random.randint(0, im_w - w)
        y1 = 0 if w == im_w else random.randint(0, im_h - h)
        cropped = crop_clip(clip, y1, x1, h, w)

        return cropped

# Файл "app\src\ml\datasets\augmentation\transforms\random_flip.py":

import random
import numpy as np


class RandomFlip:
    def __init__(self, time_flip: bool = False, horizontal_flip: bool = False) -> None:
        self.time_flip = time_flip
        self.horizontal_flip = horizontal_flip

    def __call__(self, clip):
        if random.random() < 0.5 and self.time_flip:
            clip = clip[::-1]
        if random.random() < 0.5 and self.horizontal_flip:
            clip = [np.fliplr(img) for img in clip]
        return clip

# Файл "app\src\ml\datasets\augmentation\transforms\random_resize.py":

import random

from src.ml.datasets.augmentation.utils import (
    is_numpy_clip,
    is_pil_clip,
    resize_clip,
)


class RandomResize:
    def __init__(self, ratio: tuple[float] = (3.0 / 4.0, 4.0 / 3.0), interpolation: str =
"nearest") -> None:
        self.ratio = ratio
        self.interpolation = interpolation

    def __call__(self, clip):
        scaling_factor = random.uniform(self.ratio[0], self.ratio[1])

        if is_numpy_clip(clip):
            im_h, im_w, im_c = clip[0].shape
        elif is_pil_clip(clip):
            im_w, im_h = clip[0].size

        new_w = int(im_w * scaling_factor)
        new_h = int(im_h * scaling_factor)
        new_size = (new_w, new_h)
        resized = resize_clip(clip, new_size, interpolation=self.interpolation)

        return resized

# Файл "app\src\ml\datasets\augmentation\transforms\random_rotation.py":

import random
from skimage.transform import rotate
from src.ml.datasets.augmentation.utils import is_numpy_clip, is_pil_clip


class RandomRotation:
```

```
    def __init__(self, degrees: int | tuple) -> None:
        if isinstance(degrees, int):
            if degrees < 0:
                raise ValueError("If degrees is a single number," "must be positive")
            degrees = (-degrees, degrees)
        else:
            if len(degrees) != 2:
                raise ValueError("If degrees is a sequence," "it must be of len 2.")

        self.degrees = degrees

    def __call__(self, clip):
        angle = random.uniform(self.degrees[0], self.degrees[1])
        if is_numpy_clip(clip):
            rotated = [rotate(image=img, angle=angle, preserve_range=True) for img in clip]
        elif is_pil_clip(clip):
            rotated = [img.rotate(angle) for img in clip]
        else:
            raise TypeError("Expected numpy.ndarray or PIL.Image" + "but got list of
{0}".format(type(clip[0])))

        return rotated

# Файл "app\src\ml\modules\blocks\anti_alias.py":

import torch
import torch.nn.functional as F


class AntiAliasInterpolation2d(torch.nn.Module):
    def __init__(self, channels: int, scale: float) -> None:
        super().__init__()
        if scale > 1.0:
            raise ValueError("Scale factor must be <= 1.0 for anti-aliased downsampling")

        self.scale = scale
        self.groups = channels

        if scale == 1.0:
            return  # Identity operation

        # Calculate optimal Gaussian kernel
        sigma = (1 / scale - 1) / 2
        kernel_size = 2 * round(sigma * 4) + 1
        self.ka = kernel_size // 2
        self.kb = kernel_size - self.ka - 1  # More efficient padding calculation

        # Vectorized kernel creation
        grid = torch.arange(kernel_size, dtype=torch.float32) - (kernel_size - 1) / 2
        x, y = torch.meshgrid(grid, grid, indexing="ij")
        kernel = torch.exp(-(x**2 + y**2) / (2 * sigma**2))
        kernel = kernel / kernel.sum()  # Normalize

        # Register as buffer for proper device handling
        self.register_buffer(
            "weight", kernel.view(1, 1, kernel_size, kernel_size).expand(channels, -1, -1, -
1).contiguous()
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        if self.scale == 1.0:
            return x

        x = F.pad(x, (self.ka, self.kb, self.ka, self.kb))
        x = F.conv2d(x, weight=self.weight, groups=self.groups)
        x = F.interpolate(x, scale_factor=(self.scale, self.scale))

        return x

# Файл "app\src\ml\modules\blocks\conv_blocks.py":

import torch.nn.functional as F
from torch import Tensor, nn
from torch.nn import BatchNorm2d


class ResBlock2d(nn.Module):
    def __init__(self, in_features: int, kernel_size: int | tuple[int, int], padding: int |
tuple[int, int]) -> None:
        super().__init__()
```

```
        self.conv1 = nn.Conv2d(in_features, in_features, kernel_size, padding=padding)
        self.conv2 = nn.Conv2d(in_features, in_features, kernel_size, padding=padding)
        self.norm1 = BatchNorm2d(in_features, affine=True)
        self.norm2 = BatchNorm2d(in_features, affine=True)

    def forward(self, x: Tensor) -> Tensor:
        out = F.relu(self.norm1(x))
        out = F.relu(self.norm2(self.conv1(out)))
        return self.conv2(out) + x


class UpBlock2d(nn.Module):
    def __init__(
        self, in_features: int, out_features: int, kernel_size: int = 3, padding: int = 1,
groups: int = 1
    ) -> None:
        super().__init__()
        self.conv = nn.Conv2d(in_features, out_features, kernel_size, padding=padding,
groups=groups)
        self.norm = BatchNorm2d(out_features, affine=True)

    def forward(self, x: Tensor) -> Tensor:
        out = F.interpolate(x, scale_factor=2)
        return F.relu(self.norm(self.conv(out)))


class DownBlock2d(nn.Module):
    def __init__(
        self, in_features: int, out_features: int, kernel_size: int = 3, padding: int = 1,
groups: int = 1
    ) -> None:
        super().__init__()
        self.conv = nn.Conv2d(in_features, out_features, kernel_size, padding=padding,
groups=groups)
        self.norm = BatchNorm2d(out_features, affine=True)
        self.pool = nn.AvgPool2d(kernel_size=(2, 2))

    def forward(self, x: Tensor) -> Tensor:
        return self.pool(F.relu(self.norm(self.conv(x))))


class SameBlock2d(nn.Module):
    def __init__(
        self,
        in_features: int,
        out_features: int,
        groups: int = 1,
        kernel_size: int | tuple[int, int] = 3,
        padding: int | tuple[int, int] = 1,
    ) -> None:
        super().__init__()
        self.conv = nn.Conv2d(in_features, out_features, kernel_size, padding=padding,
groups=groups)
        self.norm = BatchNorm2d(out_features, affine=True)

    def forward(self, x: Tensor) -> Tensor:
        return F.relu(self.norm(self.conv(x)))

# Файл "app\src\ml\modules\blocks\decoder.py":

import torch
from src.ml.modules.blocks.conv_blocks import UpBlock2d
from torch import nn


class Decoder(nn.Module):
    def __init__(self, block_expansion: int, in_features: int, num_blocks: int = 3, max_features:
int = 256) -> None:
        super().__init__()

        def num_channels(scale):
            return min(max_features, block_expansion * (2**scale))

        self.up_blocks = nn.ModuleList(
            [
                UpBlock2d(
                    in_features=(1 if i == num_blocks - 1 else 2) * num_channels(i + 1),
out_features=num_channels(i)
                )
                for i in reversed(range(num_blocks))
```

```
            ]
        )

        self.out_filters = block_expansion + in_features

    def forward(self, x):
        out = x.pop()
        for block in self.up_blocks:
            out = block(out)
            skip = x.pop()
            out = torch.cat([out, skip], dim=1)
        return out

# Файл "app\src\ml\modules\blocks\encoder.py":

from src.ml.modules.blocks.conv_blocks import DownBlock2d
from torch import nn


class Encoder(nn.Module):
    def __init__(self, block_expansion: int, in_features: int, num_blocks: int = 3, max_features:
int = 256) -> None:
        super().__init__()

        def num_channels(scale) -> int:
            return min(max_features, block_expansion * (2**scale))

        self.down_blocks = nn.ModuleList(
            [
                DownBlock2d(in_features=in_features if i == 0 else num_channels(i),
out_features=num_channels(i + 1))
                for i in range(num_blocks)
            ]
        )

    def forward(self, x):
        outs = [x]
        for block in self.down_blocks:
            outs.append(block(outs[-1]))
        return outs

# Файл "app\src\ml\modules\blocks\hourglass.py":

from src.ml.modules.blocks.decoder import Decoder
from src.ml.modules.blocks.encoder import Encoder
from torch import Tensor, nn


class Hourglass(nn.Module):
    def __init__(self, block_expansion: int, in_features: int, num_blocks: int = 3, max_features:
int = 256) -> None:
        super().__init__()
        self.encoder = Encoder(block_expansion, in_features, num_blocks, max_features)
        self.decoder = Decoder(block_expansion, in_features, num_blocks, max_features)
        self.out_filters = self.decoder.out_filters

    def forward(self, x: Tensor) -> Tensor:
        return self.decoder(self.encoder(x))

# Файл "app\src\ml\modules\dense_motion\dense_motion.py":

import torch
import torch.nn.functional as F
from src.ml.modules.blocks import AntiAliasInterpolation2d, Hourglass
from src.ml.modules.utils.utils import kp2gaussian, make_coordinate_grid
from torch import nn


class DenseMotionNetwork(nn.Module):
    def __init__(
        self,
        block_expansion: int,
        num_blocks: int,
        max_features: int,
        num_kp: int,
        num_channels: int,
        estimate_occlusion_map: bool = False,
        scale_factor: int = 1,
        kp_variance: float = 0.01,
    ) -> None:
```

```
        super().__init__()
        self.num_kp = num_kp
        self.scale_factor = scale_factor
        self.kp_variance = kp_variance

        # Input feature calculation (K+1)*(C+1) for heatmaps and deformed features
        in_features = (num_kp + 1) * (num_channels + 1)

        # Core components
        self.hourglass = Hourglass(block_expansion, in_features, num_blocks, max_features)

        # Output layers
        self.mask = nn.Conv2d(self.hourglass.out_filters, num_kp + 1, kernel_size=7, padding=3)
        self.occlusion = (
            nn.Conv2d(self.hourglass.out_filters, 1, kernel_size=7, padding=3) if
estimate_occlusion_map else None
        )

        # Preprocessing
        self.down = AntiAliasInterpolation2d(num_channels, scale_factor) if scale_factor != 1
else None

    def create_heatmap_representations(
        self, source_image: torch.Tensor, kp_driving: dict, kp_source: dict
    ) -> torch.Tensor:
        """Generates heatmap representations for keypoint differences."""
        spatial_size = source_image.shape[2:]
        gaussian_driving = kp2gaussian(kp_driving, spatial_size=spatial_size,
kp_variance=self.kp_variance)
        gaussian_source = kp2gaussian(kp_source, spatial_size=spatial_size,
kp_variance=self.kp_variance)

        # Add background channel
        heatmaps = torch.cat(
            [torch.zeros_like(gaussian_driving[:, :1]), gaussian_driving - gaussian_source],
dim=1  # Background
        )

        return heatmaps.unsqueeze(2)  # Add dimension for concatenation

    def create_sparse_motions(self, source_image: torch.Tensor, kp_driving: dict, kp_source:
dict) -> torch.Tensor:
        bs, _, h, w = source_image.shape
        device = source_image.device

        # Create base coordinate grid
        identity_grid = make_coordinate_grid((h, w), kp_source["value"].type()).to(device)
        identity_grid = identity_grid.view(1, 1, h, w, 2)

        # Calculate driving to source transformation
        coordinate_grid = identity_grid - kp_driving["value"].view(bs, self.num_kp, 1, 1, 2)

        # Apply Jacobian transformation if available
        if "jacobian" in kp_driving:
            jacobian = torch.matmul(kp_source["jacobian"], torch.inverse(kp_driving["jacobian"]))
            jacobian = jacobian.view(bs, self.num_kp, 1, 1, 2, 2)
            coordinate_grid = torch.einsum("bkhwij,bkhwj->bkhwi", jacobian, coordinate_grid)

        driving_to_source = coordinate_grid + kp_source["value"].view(bs, self.num_kp, 1, 1, 2)

        # Combine with identity grid for background
        return torch.cat([identity_grid.expand(bs, -1, h, w, 2), driving_to_source], dim=1)

    def _compute_deformation(self, sparse_motions: torch.Tensor, mask: torch.Tensor) ->
torch.Tensor:
        return torch.einsum("bkhw,bkhwc->bhwc", mask, sparse_motions.permute(0, 1, 4, 2, 3))

    def create_deformed_source_image(self, source_image: torch.Tensor, sparse_motions:
torch.Tensor) -> torch.Tensor:
        bs, _, h, w = source_image.shape

        # Prepare source image for warping
        source_expanded = source_image.unsqueeze(1).expand(-1, self.num_kp + 1, -1, -1, -1)
        source_expanded = source_expanded.reshape(bs * (self.num_kp + 1), -1, h, w)

        # Perform grid sampling
        motions_flat = sparse_motions.view(bs * (self.num_kp + 1), h, w, 2)
        deformed = F.grid_sample(source_expanded, motions_flat, align_corners=True)

        return deformed.view(bs, self.num_kp + 1, -1, h, w)
```

```
    def forward(self, source_image: torch.Tensor, kp_driving: dict, kp_source: dict) -> dict[str,
torch.Tensor]:
        # Preprocess input
        if self.down:
            source_image = self.down(source_image)

        bs, _, h, w = source_image.shape
        out_dict = dict()

        # Compute motion components
        heatmaps = self.create_heatmap_representations(source_image, kp_driving, kp_source)
        sparse_motions = self.create_sparse_motions(source_image, kp_driving, kp_source)
        deformed_source = self.create_deformed_source_image(source_image, sparse_motions)
        out_dict["sparse_deformed"] = deformed_source

        hourglass_input = torch.cat([heatmaps, deformed_source], dim=2)
        hourglass_input = hourglass_input.view(bs, -1, h, w)
        prediction = self.hourglass(hourglass_input)
        mask = F.softmax(self.mask(prediction), dim=1)
        out_dict["mask"] = mask

        mask = mask.unsqueeze(2)
        sparse_motions = sparse_motions.permute(0, 1, 4, 2, 3)
        deformation = (sparse_motions * mask).sum(dim=1)
        deformation = deformation.permute(0, 2, 3, 1)
        out_dict["deformation"] = deformation

        # Add occlusion map if needed
        if self.occlusion:
            out_dict["occlusion_map"] = torch.sigmoid(self.occlusion(prediction))

        return out_dict

# Файл "app\src\ml\modules\discriminator\blocks.py":

import torch
import torch.nn.functional as F
from torch import nn


class DownBlock2d(nn.Module):
    def __init__(
        self,
        in_features: int,
        out_features: int,
        norm: bool = False,
        kernel_size: int = 4,
        pool: bool = False,
        sn: bool = False,
    ) -> None:
        super().__init__()
        self.conv = nn.Conv2d(in_channels=in_features, out_channels=out_features,
kernel_size=kernel_size)
        self.conv = nn.utils.spectral_norm(self.conv) if sn else self.conv
        self.norm: nn.Module = nn.InstanceNorm2d(out_features, affine=True) if norm else
nn.Identity()
        self.pool: nn.Module = nn.AvgPool2d(kernel_size=2) if pool else nn.Identity()

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.conv(x)
        x = self.norm(x)
        x = F.leaky_relu(x, 0.2)
        x = self.pool(x)
        return x

# Файл "app\src\ml\modules\discriminator\discriminator_full_model.py":

import torch
from src.ml.modules.utils import ImagePyramide
from src.ml.modules.utils.utils import detach_kp
from torch import nn


class DiscriminatorFullModel(torch.nn.Module):
    def __init__(self, kp_extractor, generator: nn.Module, discriminator: nn.Module,
train_params: dict) -> None:
        super().__init__()
        self.kp_extractor = kp_extractor
        self.generator = generator
```

```
        self.discriminator = discriminator
        self.train_params = train_params

        self.scales = self.discriminator.scales
        self.loss_weights = train_params["loss_weights"]

        self.pyramid = ImagePyramide(self.scales, generator.num_channels)
        if torch.cuda.is_available():
            self.pyramid = self.pyramid.cuda()

    def forward(self, x: dict, generated: dict) -> dict[str, float]:
        # Extract pyramids for real and generated frames
        pyramide_real = self.pyramid(x["driving"])
        pyramide_fake = self.pyramid(generated["prediction"].detach())

        # Detach keypoints to stop gradient to generator
        kp_driving_detached = detach_kp(generated["kp_driving"])

        # Run discriminator
        disc_fake = self.discriminator(pyramide_fake, kp=kp_driving_detached)
        disc_real = self.discriminator(pyramide_real, kp=kp_driving_detached)

        # Compute GAN loss (LSGAN formulation)
        disc_loss_total = 0.0
        for scale in self.scales:
            key = f"prediction_map_{scale}"
            real_map = disc_real[key]
            fake_map = disc_fake[key]
            scale_loss = (1 - real_map) ** 2 + fake_map**2
            disc_loss_total += self.loss_weights["discriminator_gan"] * scale_loss.mean()

        return {"disc_gan": disc_loss_total}

# Файл "app\src\ml\modules\discriminator\multi_scale_discriminator.py":

import torch
from src.ml.modules.discriminator.pix2pix_discriminator import Discriminator
from torch import nn


class MultiScaleDiscriminator(nn.Module):
    def __init__(self, scales=(), **kwargs) -> None:
        super().__init__()
        self.scales = scales

        discs = {}
        for scale in scales:
            discs[str(scale).replace(".", "-")] = Discriminator(**kwargs)
        self.discs = nn.ModuleDict(discs)

    def forward(self, x: dict, kp=None) -> dict[str, torch.Tensor] | list[torch.Tensor]:
        out_dict = {}

        for scale, disc in self.discs.items():
            scale = str(scale).replace("-", ".")
            key = f"prediction_{scale}"

            feature_maps, prediction_map = disc(x[key], kp)

            out_dict[f"feature_maps_{scale}"] = feature_maps
            out_dict[f"prediction_map_{scale}"] = prediction_map

        return out_dict

# Файл "app\src\ml\modules\discriminator\pix2pix_discriminator.py":

import torch
from src.ml.modules.discriminator.blocks import DownBlock2d
from src.ml.modules.utils.utils import kp2gaussian
from torch import nn


class Discriminator(nn.Module):
    def __init__(
        self,
        num_channels: int = 3,
        block_expansion: int = 64,
        num_blocks: int = 4,
        max_features: int = 512,
        sn: bool = False,
```

```
        use_kp: bool = False,
        num_kp: int = 10,
        kp_variance: float = 0.01,
        **_
    ) -> None:
        super().__init__()
        self.use_kp = use_kp
        self.kp_variance = kp_variance

        self.down_blocks = nn.ModuleList()
        for i in range(num_blocks):
            self.down_blocks.append(
                DownBlock2d(
                    in_features=(
                        num_channels + num_kp * use_kp if i == 0 else min(max_features,
block_expansion * (2**i))
                    ),
                    out_features=min(max_features, block_expansion * (2 ** (i + 1))),
                    norm=(i != 0),
                    kernel_size=4,
                    pool=(i != num_blocks - 1),
                    sn=sn,
                )
            )

        conv = nn.Conv2d(self.down_blocks[-1].conv.out_channels, out_channels=1, kernel_size=1)
        self.conv = nn.utils.spectral_norm(conv) if sn else conv

    def forward(self, x: torch.Tensor, kp: dict | None = None) -> tuple[list[torch.Tensor],
torch.Tensor]:
        if self.use_kp:
            heatmap = kp2gaussian(kp, x.shape[2:], self.kp_variance)
            x = torch.cat([x, heatmap], dim=1)

        feature_maps = []
        for down_block in self.down_blocks:
            x = down_block(x)
            feature_maps.append(x)

        prediction_map = self.conv(x)

        return feature_maps, prediction_map

# Файл "app\src\ml\modules\generator\generator_full_model.py":

import torch
from src.ml.modules.generator.vgg19 import Vgg19
from src.ml.modules.utils import ImagePyramide, Transform
from src.ml.modules.utils.utils import detach_kp
from torch import nn


class GeneratorFullModel(nn.Module):
    def __init__(self, kp_extractor, generator: nn.Module, discriminator: nn.Module,
train_params: dict) -> None:
        super().__init__()
        self.kp_extractor = kp_extractor
        self.generator = generator
        self.discriminator = discriminator
        self.train_params = train_params

        self.scales = train_params["scales"]
        self.disc_scales = self.discriminator.scales
        self.loss_weights = train_params["loss_weights"]

        self.pyramid = ImagePyramide(self.scales, generator.num_channels)
        if torch.cuda.is_available():
            self.pyramid = self.pyramid.cuda()

        if any(self.loss_weights["perceptual"]):
            self.vgg = Vgg19()
            if torch.cuda.is_available():
                self.vgg = self.vgg.cuda()

    def forward(self, x: dict) -> tuple[dict, dict]:
        # Keypoint extraction
        kp_source = self.kp_extractor(x["source"])
        kp_driving = self.kp_extractor(x["driving"])

        generated = self.generator(x["source"], kp_source=kp_source, kp_driving=kp_driving)
```

```
        generated.update({"kp_source": kp_source, "kp_driving": kp_driving})
        loss_values = {}
        pyramide_real = self.pyramid(x["driving"])
        pyramide_fake = self.pyramid(generated["prediction"])

        if any(self.loss_weights["perceptual"]):
            perceptual_loss = 0
            for scale in self.scales:
                x_vgg = self.vgg(pyramide_fake[f"prediction_{scale}"])
                y_vgg = self.vgg(pyramide_real[f"prediction_{scale}"])
                for i, weight in enumerate(self.loss_weights["perceptual"]):
                    perceptual_loss += weight * torch.abs(x_vgg[i] - y_vgg[i].detach()).mean()
            loss_values["perceptual"] = perceptual_loss

        if self.loss_weights["generator_gan"] != 0:
            detached_kp_driving = detach_kp(kp_driving)
            disc_maps_fake = self.discriminator(pyramide_fake, kp=detached_kp_driving)
            disc_maps_real = self.discriminator(pyramide_real, kp=detached_kp_driving)

            value_total = 0
            for scale in self.disc_scales:
                key = f"prediction_map_{scale}"
                value = ((1 - disc_maps_fake[key]) ** 2).mean()
                value_total += self.loss_weights["generator_gan"] * value
            loss_values["gen_gan"] = value_total

            if any(self.loss_weights["feature_matching"]):
                fm_loss = 0
                for scale in self.disc_scales:
                    key = f"feature_maps_{scale}"
                    for i, (real_f, gen_f) in enumerate(zip(disc_maps_real[key],
disc_maps_fake[key])):
                        weight = self.loss_weights["feature_matching"][i]
                        if weight != 0:
                            fm_loss += weight * torch.abs(real_f - gen_f).mean()
                loss_values["feature_matching"] = value_total

        eq_val_w = self.loss_weights["equivariance_value"]
        eq_jac_w = self.loss_weights["equivariance_jacobian"]
        if eq_val_w + eq_jac_w != 0:
            transform = Transform(x["driving"].shape[0], **self.train_params["transform_params"])
            transformed_frame = transform.transform_frame(x["driving"])
            transformed_kp = self.kp_extractor(transformed_frame)
            generated.update({"transformed_frame": transformed_frame, "transformed_kp":
transformed_kp})

            if eq_val_w != 0:
                warped = transform.warp_coordinates(transformed_kp["value"])
                val_loss = torch.abs(kp_driving["value"] - warped).mean()
                loss_values["equivariance_value"] = eq_val_w * val_loss

            if eq_jac_w != 0:
                jacobian_transformed = torch.matmul(
                    transform.jacobian(transformed_kp["value"]), transformed_kp["jacobian"]
                )
                normed = torch.matmul(torch.inverse(kp_driving["jacobian"]),
jacobian_transformed)
                identity = torch.eye(2).view(1, 1, 2, 2).type(normed.type())
                jac_loss = torch.abs(identity - normed).mean()
                loss_values["equivariance_jacobian"] = eq_jac_w * jac_loss

        return loss_values, generated

# Файл "app\src\ml\modules\generator\occlusion_aware_generator.py":

import torch
import torch.nn.functional as F
from src.ml.modules.blocks import (
    DownBlock2d,
    ResBlock2d,
    SameBlock2d,
    UpBlock2d,
)
from src.ml.modules.dense_motion import DenseMotionNetwork
from torch import nn


class OcclusionAwareGenerator(nn.Module):
    def __init__(
        self,
```

```
        num_channels: int,
        num_kp: int,
        block_expansion: int,
        max_features: int,
        num_down_blocks: int,
        num_bottleneck_blocks: int,
        estimate_occlusion_map: bool = False,
        dense_motion_params: dict = None,
        estimate_jacobian: bool = False,
    ) -> None:
        super().__init__()

        # Dense Motion Network setup
        self.dense_motion_network = (
            DenseMotionNetwork(
                num_kp=num_kp,
                num_channels=num_channels,
                estimate_occlusion_map=estimate_occlusion_map,
                **dense_motion_params
            )
            if dense_motion_params
            else None
        )

        # First block: SameBlock2d
        self.first = SameBlock2d(num_channels, block_expansion, kernel_size=(7, 7), padding=(3,
3))

        # Downsampling blocks
        self.down_blocks = nn.ModuleList(
            [
                DownBlock2d(
                    min(max_features, block_expansion * (2**i)), min(max_features,
block_expansion * (2 ** (i + 1)))
                )
                for i in range(num_down_blocks)
            ]
        )

        self.up_blocks = nn.ModuleList(
            [
                UpBlock2d(
                    min(max_features, block_expansion * (2 ** (num_down_blocks - i))),
                    min(max_features, block_expansion * (2 ** (num_down_blocks - i - 1))),
                )
                for i in range(num_down_blocks)
            ]
        )

        self.bottleneck = torch.nn.Sequential()
        in_features = min(max_features, block_expansion * (2**num_down_blocks))
        for i in range(num_bottleneck_blocks):
            self.bottleneck.add_module("r" + str(i), ResBlock2d(in_features, kernel_size=(3, 3),
padding=(1, 1)))

        # Final output layer
        self.final = nn.Conv2d(block_expansion, num_channels, kernel_size=(7, 7), padding=(3, 3))

        # Store options for occlusion map and channels
        self.estimate_occlusion_map = estimate_occlusion_map
        self.num_channels = num_channels

    @staticmethod
    def deform_input(inp, deformation):
        _, h_old, w_old, _ = deformation.shape
        _, _, h, w = inp.shape
        if h_old != h or w_old != w:
            deformation = deformation.permute(0, 3, 1, 2)
            deformation = F.interpolate(deformation, size=(h, w), mode="bilinear")
            deformation = deformation.permute(0, 2, 3, 1)
        return F.grid_sample(inp, deformation, align_corners=True)

    def forward(self, source_image, kp_driving, kp_source) -> dict[str, torch.Tensor]:
        # Encoding (downsampling) part
        out = self.first(source_image)
        for i in range(len(self.down_blocks)):
            out = self.down_blocks[i](out)

        output_dict = {}
```

```
        # Transforming feature representation according to deformation and occlusion
        if self.dense_motion_network:
            dense_motion = self.dense_motion_network(
                source_image=source_image, kp_driving=kp_driving, kp_source=kp_source
            )
            output_dict.update(
                {
                    "mask": dense_motion["mask"],
                    "sparse_deformed": dense_motion["sparse_deformed"],
                }
            )

            if "occlusion_map" in dense_motion:
                occlusion_map = dense_motion["occlusion_map"]
                output_dict["occlusion_map"] = occlusion_map
            else:
                occlusion_map = None
            deformation = dense_motion["deformation"]
            out = self.deform_input(out, deformation)

            if occlusion_map is not None:
                if out.shape[2:] != occlusion_map.shape[2:]:
                    occlusion_map = F.interpolate(occlusion_map, size=out.shape[2:],
mode="bilinear")
                out = out * occlusion_map

            output_dict["deformed"] = self.deform_input(source_image, deformation)

        # Decoding part
        out = self.bottleneck(out)
        for up_block in self.up_blocks:
            out = up_block(out)

        # Final prediction and sigmoid activation
        out = self.final(out)
        out = F.sigmoid(out)

        output_dict["prediction"] = out

        return output_dict

# Файл "app\src\ml\modules\generator\vgg19.py":

import torch
from torchvision import models


class Vgg19(torch.nn.Module):
    def __init__(self, requires_grad: bool = False) -> None:
        super().__init__()
        vgg_features = models.vgg19(pretrained=True).features

        # Define VGG slices corresponding to relu1_2, relu2_2, relu3_2, relu4_2, relu5_2
        self.slices = torch.nn.ModuleList(
            [
                vgg_features[:2],  # relu1_2
                vgg_features[2:7],  # relu2_2
                vgg_features[7:12],  # relu3_2
                vgg_features[12:21],  # relu4_2
                vgg_features[21:30],  # relu5_2
            ]
        )

        # Normalization buffers (not learnable, no grad)
        self.register_buffer("mean", torch.tensor([0.485, 0.456, 0.406]).view(1, 3, 1, 1))
        self.register_buffer("std", torch.tensor([0.229, 0.224, 0.225]).view(1, 3, 1, 1))

        if not requires_grad:
            for param in self.parameters():
                param.requires_grad = False

    def forward(self, x: torch.Tensor) -> list:
        x = (x - self.mean) / self.std
        features = []
        for slice in self.slices:
            x = slice(x)
            features.append(x)
        return features

# Файл "app\src\ml\modules\keypoint_detector\keypoint_detector.py":
```

```
import torch
import torch.nn.functional as F
from src.ml.modules.blocks import AntiAliasInterpolation2d, Hourglass
from src.ml.modules.utils.utils import gaussian2kp
from torch import Tensor, nn


class KPDetector(nn.Module):
    def __init__(
        self,
        block_expansion: int,
        num_kp: int,
        num_channels: int,
        max_features: int,
        num_blocks: int,
        temperature: float,
        estimate_jacobian: bool = False,
        scale_factor: float = 1.0,
        single_jacobian_map: bool = False,
        pad: str | int | tuple[int, int] = 0,
    ) -> None:
        super().__init__()
        self.temperature = temperature
        self.scale_factor = scale_factor
        self.estimate_jacobian = estimate_jacobian
        self.num_kp = num_kp

        # Backbone feature extractor
        self.predictor = Hourglass(
            block_expansion, in_features=num_channels, max_features=max_features,
num_blocks=num_blocks
        )

        # Keypoint heatmap prediction
        self.kp = nn.Conv2d(in_channels=self.predictor.out_filters, out_channels=num_kp,
kernel_size=7, padding=pad)

        if estimate_jacobian:
            self.num_jacobian_maps = 1 if single_jacobian_map else num_kp
            self.jacobian = nn.Conv2d(
                in_channels=self.predictor.out_filters,
                out_channels=4 * self.num_jacobian_maps,
                kernel_size=7,
                padding=pad,
            )
            self.jacobian.weight.data.zero_()
            self.jacobian.bias.data.copy_(torch.tensor([1, 0, 0, 1] * self.num_jacobian_maps,
dtype=torch.float))
        else:
            self.jacobian = None

        self.down = AntiAliasInterpolation2d(num_channels, scale_factor) if scale_factor != 1
else None

    def forward(self, x) -> dict[str, Tensor]:
        if self.down:
            x = self.down(x)

        feature_map = self.predictor(x)
        heatmap = self.kp(feature_map)
        B, K, H, W = heatmap.shape
        heatmap_flat = heatmap.view(B, K, -1)
        heatmap = F.softmax(heatmap_flat / self.temperature, dim=2).view(B, K, H, W)
        out = gaussian2kp(heatmap)

        if self.jacobian:
            jacobian_map = self.jacobian(feature_map)  # shape: [B, 4*num_maps, H, W]
            jacobian_map = jacobian_map.view(B, self.num_jacobian_maps, 4, H, W)

            weighted = (heatmap.unsqueeze(2) * jacobian_map).view(B, K, 4, -1)
            jacobian = weighted.sum(dim=-1).view(B, K, 2, 2)
            out["jacobian"] = jacobian

        return out

# Файл "app\src\ml\modules\utils\image_pyramide.py":

import torch
from src.ml.modules.blocks import AntiAliasInterpolation2d
from torch import nn
```

```
class ImagePyramide(torch.nn.Module):
    def __init__(self, scales: list[float], num_channels: int) -> None:
        super().__init__()

        # Validate input scales
        if not scales or any(scale <= 0 for scale in scales):
            raise ValueError("Scales must be positive values")

        # Create downsampling modules
        self.downs = nn.ModuleDict(
            {f"scale_{scale}": AntiAliasInterpolation2d(num_channels, scale) for scale in scales}
        )

        # Cache scale names for faster forward pass
        self.scale_names = list(self.downs.keys())
        self.output_keys = [f'prediction_{scale.split("_")[1]}' for scale in self.scale_names]

    def forward(self, x: torch.Tensor) -> dict[str, torch.Tensor]:
        return {key: down_module(x) for key, down_module in zip(self.output_keys,
self.downs.values())}

# Файл "app\src\ml\modules\utils\transform.py":

import torch
import torch.nn.functional as F
from src.ml.modules.utils.utils import make_coordinate_grid
from torch.autograd import grad


class Transform:
    def __init__(self, bs: int, **kwargs) -> None:
        self.bs = bs

        # Initialize random affine transformation parameters
        noise = torch.normal(mean=0, std=kwargs["sigma_affine"] * torch.ones([bs, 2, 3]))
        self.theta = noise + torch.eye(2, 3).view(1, 2, 3)

        # Initialize TPS transformation if parameters provided
        self.tps = ("sigma_tps" in kwargs) and ("points_tps" in kwargs)
        if self.tps:
            self.control_points = make_coordinate_grid((kwargs["points_tps"],
kwargs["points_tps"]), type=noise.type())
            self.control_points = self.control_points.unsqueeze(0)
            self.control_params = torch.normal(
                mean=0, std=kwargs["sigma_tps"] * torch.ones([bs, 1, kwargs["points_tps"] ** 2])
            )

    def transform_frame(self, frame: torch.Tensor) -> torch.Tensor:
        grid = make_coordinate_grid(frame.shape[2:], type=frame.type()).unsqueeze(0)
        grid = grid.view(1, frame.shape[2] * frame.shape[3], 2)
        warped_grid = self.warp_coordinates(grid).view(self.bs, *frame.shape[2:], 2)
        return F.grid_sample(frame, warped_grid, padding_mode="reflection", align_corners=True)

    def warp_coordinates(self, coordinates: torch.Tensor) -> torch.Tensor:
        """Apply both affine and TPS transformations to coordinates.
        Args:
            coordinates: Tensor of shape [bs, N, 2]
        Returns:
            Transformed coordinates of same shape
        """
        # Apply affine transformation
        transformed = self._apply_affine_transform(coordinates)

        # Apply TPS transformation if enabled
        if self.tps:
            transformed = self._apply_tps_transform(transformed, coordinates)

        return transformed

    def _apply_affine_transform(self, coordinates: torch.Tensor) -> torch.Tensor:
        """Apply affine transformation to coordinates."""
        # Efficient affine transform using einsum
        return torch.einsum(
            "bni,bij->bnj",
            torch.cat([coordinates, torch.ones_like(coordinates[..., :1])], dim=-1),
            self.theta.type_as(coordinates),
        )

    def _apply_tps_transform(self, transformed: torch.Tensor, coordinates: torch.Tensor) ->
torch.Tensor:
```

```
        # Calculate distances between coordinates and control points
        diff = coordinates.unsqueeze(2) - self.control_points.type_as(coordinates).view(1, 1, -1,
2)
        distances = torch.norm(diff, p=1, dim=-1)  # L1 distance

        # Compute TPS radial basis function
        result = distances**2 * torch.log(distances + 1e-6)
        result = (result * self.control_params.type_as(coordinates)).sum(dim=2)

        return transformed + result.unsqueeze(-1)

    def jacobian(self, coordinates: torch.Tensor) -> torch.Tensor:
        new_coordinates = self.warp_coordinates(coordinates)
        grad_x = grad(new_coordinates[..., 0].sum(), coordinates, create_graph=True)
        grad_y = grad(new_coordinates[..., 1].sum(), coordinates, create_graph=True)
        jacobian = torch.cat([grad_x[0].unsqueeze(-2), grad_y[0].unsqueeze(-2)], dim=-2)
        return jacobian

# Файл "app\src\ml\runners\animation.py":

from src.ml.datasets import FramesDataset
from src.ml.runners.base import BaseRunner
from src.ml.services.animation import AnimationService


class AnimationRunner(BaseRunner):
    def __init__(self, args, log):
        super().__init__(args, log)
        self.animation_service = AnimationService()

    def run(self):
        dataset = FramesDataset(is_train=False, **self.config["dataset_params"])
        self.log.info("Animation started...")
        self.animation_service.animate(
            self.config, self.generator, self.kp_detector, self.args.checkpoint, self.log_dir,
dataset
        )

# Файл "app\src\ml\runners\base.py":

from abc import ABC, abstractmethod
from argparse import Namespace
from logging import Logger

from src.ml.services.model import ModelService


class BaseRunner(ABC):
    def __init__(self, args: Namespace, log: Logger):
        self.args = args
        self.log = log
        self.device_ids = args.device_ids
        self.verbose = args.verbose
        model_service = ModelService(args.configs, args.checkpoint, args.log_dir, cpu=args.cpu)
        self.config = model_service.config
        self.log_dir = model_service.log_dir
        self.generator, self.discriminator, self.kp_detector =
model_service.init_training_models(self.device_ids)

    @abstractmethod
    def run(self):
        pass

# Файл "app\src\ml\runners\reconstruction.py":

from src.ml.datasets import FramesDataset
from src.ml.runners.base import BaseRunner
from src.ml.services.reconstruction import ReconstructionService


class ReconstructionRunner(BaseRunner):
    def run(self):
        dataset = FramesDataset(is_train=False, **self.config["dataset_params"])
        self.log.info("Reconstruction started...")
        ReconstructionService.reconstruction(
            self.config, self.generator, self.kp_detector, self.args.checkpoint, self.log_dir,
dataset
        )

# Файл "app\src\ml\runners\training.py":
```

```
from src.ml.datasets import FramesDataset
from src.ml.runners.base import BaseRunner
from src.ml.services.training import TrainingService


class TrainingRunner(BaseRunner):
    def run(self):
        dataset = FramesDataset(is_train=True, **self.config["dataset_params"])
        self.log.info("Training started...")
        TrainingService.train(
            self.config,
            self.generator,
            self.discriminator,
            self.kp_detector,
            self.args.checkpoint,
            self.log_dir,
            dataset,
            self.device_ids,
        )

# Файл "app\src\ml\services\animation.py":

import os

import numpy as np
import torch
from scipy.spatial import ConvexHull
from src.ml.datasets import PairedDataset
from src.ml.datasets.replicate import DataParallelWithCallback
from src.ml.services.logging import LoggingService
from src.ml.services.visualization import VisualizationService
from torch.utils.data import DataLoader
from tqdm import tqdm


class AnimationService:
    @classmethod
    def make_animation(
        cls,
        source_image: np.ndarray,
        driving_video: list[np.ndarray],
        generator: torch.nn.Module,
        kp_detector: torch.nn.Module,
        relative: bool = True,
        adapt_movement_scale: bool = True,
        cpu: bool = False,
    ) -> list[np.ndarray]:
        device = torch.device("cpu") if cpu else torch.device("cuda")

        with torch.no_grad():
            # Convert inputs to tensors
            source = cls._image_to_tensor(source_image).to(device)
            driving = cls._video_to_tensor(driving_video).to(device)

            # Get keypoints
            kp_source = kp_detector(source)
            kp_driving_initial = kp_detector(driving[:, :, 0])

            predictions = []
            for frame_idx in tqdm(range(driving.shape[2]), desc="Generating fomm"):
                driving_frame = driving[:, :, frame_idx]
                kp_driving = kp_detector(driving_frame)

                kp_norm = cls.normalize_kp(
                    kp_source=kp_source,
                    kp_driving=kp_driving,
                    kp_driving_initial=kp_driving_initial,
                    use_relative_movement=relative,
                    use_relative_jacobian=relative,
                    adapt_movement_scale=adapt_movement_scale,
                )

                out = generator(source, kp_source=kp_source, kp_driving=kp_norm)
                predictions.append(cls._tensor_to_image(out["prediction"]))

        return predictions

    @staticmethod
    def _image_to_tensor(image: np.ndarray) -> torch.Tensor:
```

```
        return torch.from_numpy(image[np.newaxis].astype(np.float32)).permute(0, 3, 1, 2)

    @staticmethod
    def _tensor_to_image(tensor: torch.Tensor) -> np.ndarray:
        return np.transpose(tensor.data.cpu().numpy(), [0, 2, 3, 1])[0]

    @staticmethod
    def _video_to_tensor(video: list[np.ndarray]) -> torch.Tensor:
        return torch.from_numpy(np.array(video)[np.newaxis].astype(np.float32)).permute(0, 4, 1,
2, 3)

    @staticmethod
    def animate(
        config: dict,
        generator: torch.nn.Module,
        kp_detector: torch.nn.Module,
        checkpoint: str,
        log_dir: str,
        dataset: PairedDataset,
        imageio=None,
    ) -> None:
        # Setup directories
        log_dir = os.path.join(log_dir, "services")
        png_dir = os.path.join(log_dir, "png")
        os.makedirs(png_dir, exist_ok=True)

        # Load checkpoint
        if not checkpoint:
            raise ValueError("Checkpoint must be specified for fomm mode")
        LoggingService.load_cpk(checkpoint, generator=generator, kp_detector=kp_detector)

        # Prepare models
        if torch.cuda.is_available():
            generator = DataParallelWithCallback(generator)
            kp_detector = DataParallelWithCallback(kp_detector)

        generator.eval()
        kp_detector.eval()

        # Create dataset and dataloader
        animate_params = config["animate_params"]
        dataset = PairedDataset(initial_dataset=dataset,
number_of_pairs=animate_params["num_pairs"])
        dataloader = DataLoader(dataset, batch_size=1, shuffle=False, num_workers=2)

        # Initialize visualizer
        visualizer = VisualizationService(**config["visualizer_params"])

        # Process each pair
        for x in tqdm(dataloader, desc="Animating dataset"):
            with torch.no_grad():
                predictions, visualizations = [], []
                driving_video = x["driving_video"]
                source_frame = x["source_video"][:, :, 0, :, :]

                # Get keypoints
                kp_source = kp_detector(source_frame)
                kp_driving_initial = kp_detector(driving_video[:, :, 0])

                # Process each frame
                for frame_idx in range(driving_video.shape[2]):
                    driving_frame = driving_video[:, :, frame_idx]
                    kp_driving = kp_detector(driving_frame)

                    kp_norm = AnimationService.normalize_kp(
                        kp_source=kp_source,
                        kp_driving=kp_driving,
                        kp_driving_initial=kp_driving_initial,
                        **animate_params["normalization_params"],
                    )

                    out = generator(source_frame, kp_source=kp_source, kp_driving=kp_norm)

                    # Prepare outputs
                    out.update({"kp_driving": kp_driving, "kp_source": kp_source, "kp_norm":
kp_norm})
                    del out["sparse_deformed"]

                    # Save results
                    predictions.append(AnimationService._tensor_to_image(out["prediction"]))
```

```
                    visualizations.append(visualizer.visualize(source=source_frame,
driving=driving_frame, out=out))

                # Save outputs
                result_name = f"{x['driving_name'][0]}-{x['source_name'][0]}"
                predictions = np.concatenate(predictions, axis=1)

                imageio.imsave(
                    os.path.join(png_dir, f"{result_name}.png"),
                    (255 * predictions).astype(np.uint8),
                )

                imageio.mimsave(
                    os.path.join(log_dir, f"{result_name}{animate_params['format']}"),
                    visualizations,
                )

    @classmethod
    def normalize_kp(
        cls,
        kp_source: dict[str, torch.Tensor],
        kp_driving: dict[str, torch.Tensor],
        kp_driving_initial: dict[str, torch.Tensor],
        adapt_movement_scale: bool = False,
        use_relative_movement: bool = False,
        use_relative_jacobian: bool = False,
    ) -> dict[str, torch.Tensor]:
        kp_new = kp_driving.copy()

        if adapt_movement_scale:
            scale = cls._calculate_movement_scale(kp_source, kp_driving_initial)
        else:
            scale = 1.0

        if use_relative_movement:
            kp_value_diff = (kp_driving["value"] - kp_driving_initial["value"]) * scale
            kp_new["value"] = kp_value_diff + kp_source["value"]

            if use_relative_jacobian:
                jacobian_diff = torch.matmul(kp_driving["jacobian"],
torch.inverse(kp_driving_initial["jacobian"]))
                kp_new["jacobian"] = torch.matmul(jacobian_diff, kp_source["jacobian"])

        return kp_new

    @staticmethod
    def _calculate_movement_scale(
        kp_source: dict[str, torch.Tensor], kp_driving_initial: dict[str, torch.Tensor]
    ) -> float:
        """Calculate movement scale based on convex hull areas."""
        source_area = ConvexHull(kp_source["value"][0].cpu().numpy()).volume
        driving_area = ConvexHull(kp_driving_initial["value"][0].cpu().numpy()).volume
        return np.sqrt(source_area) / np.sqrt(driving_area)

# Файл "app\src\ml\services\logging.py":

import collections
import logging
import os
import sys

import imageio
import numpy as np
import torch
from src.ml.services.visualization import VisualizationService


class LoggingService:
    def __init__(
        self,
        log_dir: str,
        log_file_name: str = "log.txt",
        checkpoint_freq: int = 100,
        visualizer_params=None,
        zfill_num: int = 8,
    ) -> None:
        self.loss_list = []
        self.cpk_dir = log_dir
        self.visualizations_dir = os.path.join(log_dir, "train-vis")
        os.makedirs(self.visualizations_dir, exist_ok=True)
```

```
        self.log_file = open(os.path.join(log_dir, log_file_name), "a")
        self.zfill_num = zfill_num
        self.visualizer = VisualizationService(**visualizer_params)
        self.checkpoint_freq = checkpoint_freq
        self.epoch = 0
        self.best_loss = float("inf")
        self.names = None
        self.models = None

    def log_scores(self, loss_names) -> None:
        loss_mean = np.array(self.loss_list).mean(axis=0)

        loss_string = "; ".join(["%s - %.5f" % (name, value) for name, value in zip(loss_names,
loss_mean)])
        loss_string = str(self.epoch).zfill(self.zfill_num) + ") " + loss_string

        print(loss_string, file=self.log_file)
        self.loss_list = []
        self.log_file.flush()

    def visualize_rec(self, inp, out) -> None:
        image = self.visualizer.visualize(inp["driving"], inp["source"], out)
        imageio.imsave(
            os.path.join(self.visualizations_dir, "%s-rec.png" %
str(self.epoch).zfill(self.zfill_num)), image
        )

    def save_cpk(self, emergent: bool = False) -> None:
        cpk = {k: v.state_dict() for k, v in self.models.items()}
        cpk["epoch"] = self.epoch
        cpk_path = os.path.join(self.cpk_dir, "%s-checkpoint.pth.tar" %
str(self.epoch).zfill(self.zfill_num))
        if not (os.path.exists(cpk_path) and emergent):
            torch.save(cpk, cpk_path)

    @staticmethod
    def load_cpk(
        checkpoint_path: str,
        generator=None,
        discriminator=None,
        kp_detector=None,
        optimizer_generator=None,
        optimizer_discriminator=None,
        optimizer_kp_detector=None,
    ):
        log = LoggingService.setup_logger(__name__)
        checkpoint = torch.load(checkpoint_path)
        if generator:
            generator.load_state_dict(checkpoint["generator"])
        if kp_detector:
            kp_detector.load_state_dict(checkpoint["kp_detector"])
        if discriminator:
            try:
                discriminator.load_state_dict(checkpoint["discriminator"])
            except:
                log.error("No discriminator in the state-dict. Dicriminator will be randomly
initialized")
        if optimizer_generator:
            optimizer_generator.load_state_dict(checkpoint["optimizer_generator"])
        if optimizer_discriminator:
            try:
                optimizer_discriminator.load_state_dict(checkpoint["optimizer_discriminator"])
            except RuntimeError:
                log.error("No discriminator optimizer in the state-dict. Optimizer will be not
initialized")
        if optimizer_kp_detector:
            optimizer_kp_detector.load_state_dict(checkpoint["optimizer_kp_detector"])

        return checkpoint["epoch"]

    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_val, exc_tb) -> None:
        if "models" in self.__dict__:
            self.save_cpk()
        self.log_file.close()

    def log_iter(self, losses) -> None:
        losses = collections.OrderedDict(losses.items())
```

```
        if self.names is None:
            self.names = list(losses.keys())
        self.loss_list.append(list(losses.values()))

    def log_epoch(self, epoch, models, inp, out) -> None:
        self.epoch = epoch
        self.models = models
        if (self.epoch + 1) % self.checkpoint_freq == 0:
            self.save_cpk()
        self.log_scores(self.names)
        self.visualize_rec(inp, out)

    @staticmethod
    def setup_logger(name: str) -> logging.Logger:
        logger = logging.getLogger(name)
        logger.setLevel(logging.DEBUG)
        formatter = logging.Formatter(
            "%(asctime)s - %(name)s - %(levelname)s - %(message)s", datefmt="%Y-%m-%d %H:%M:%S"
        )
        console_handler = logging.StreamHandler(sys.stdout)
        console_handler.setLevel(logging.INFO)
        console_handler.setFormatter(formatter)
        logger.addHandler(console_handler)
        return logger

# Файл "app\src\ml\services\model.py":

import os
from shutil import copy
from time import gmtime, strftime

import torch
import yaml
from src.ml.datasets.replicate import DataParallelWithCallback
from src.ml.modules.discriminator import MultiScaleDiscriminator
from src.ml.modules.generator import OcclusionAwareGenerator
from src.ml.modules.keypoint_detector import KPDetector


class ModelService:
    def __init__(
        self,
        config_path: str,
        checkpoint_path: str = None,
        log_dir: str = "log",
        cpu: bool = False,
        verbose: bool = False,
    ) -> None:
        self.config_path = config_path
        self.checkpoint_path = checkpoint_path
        self.log_dir = log_dir
        self.cpu = cpu
        self.verbose = verbose
        self.config = self._load_config()
        self.log_dir = self._prepare_log_dir()
        self.device = torch.device("cpu" if cpu else "cuda:0")

    def _load_config(self):
        with open(self.config_path) as f:
            return yaml.safe_load(f)

    def _prepare_log_dir(self):
        if self.checkpoint_path:
            return os.path.join(*os.path.split(self.checkpoint_path)[:-1])
        else:
            log_dir = os.path.join(self.log_dir,
os.path.basename(self.config_path).split(".")[0])
            log_dir += " " + strftime("%d_%m_%y_%H.%M.%S", gmtime())
            os.makedirs(log_dir, exist_ok=True)

            config_copy_path = os.path.join(log_dir, os.path.basename(self.config_path))
            if not os.path.exists(config_copy_path):
                copy(self.config_path, config_copy_path)

            return log_dir

    def init_training_models(self, device_ids: list) -> tuple[torch.nn.Module, torch.nn.Module,
torch.nn.Module]:
        generator = OcclusionAwareGenerator(
            **self.config["model_params"]["generator_params"],
```

```
**self.config["model_params"]["common_params"]
        )
        discriminator = MultiScaleDiscriminator(
            **self.config["model_params"]["discriminator_params"],
**self.config["model_params"]["common_params"]
        )
        kp_detector = KPDetector(
            **self.config["model_params"]["kp_detector_params"],
**self.config["model_params"]["common_params"]
        )

        if torch.cuda.is_available() and not self.cpu:
            generator.to(device_ids[0])
            discriminator.to(device_ids[0])
            kp_detector.to(device_ids[0])

        if self.verbose:
            print(generator)
            print(discriminator)
            print(kp_detector)

        return generator, discriminator, kp_detector

    def load_eval_models(self):
        generator = OcclusionAwareGenerator(
            **self.config["model_params"]["generator_params"],
            **self.config["model_params"]["common_params"],
        )

        kp_detector = KPDetector(
            **self.config["model_params"]["kp_detector_params"],
            **self.config["model_params"]["common_params"],
        )

        if not self.cpu:
            generator.cuda()
            kp_detector.cuda()

        checkpoint = torch.load(self.checkpoint_path, map_location=self.device)
        generator.load_state_dict(checkpoint["generator"])
        kp_detector.load_state_dict(checkpoint["kp_detector"])

        if not self.cpu:
            generator = DataParallelWithCallback(generator)
            kp_detector = DataParallelWithCallback(kp_detector)

        generator.eval()
        kp_detector.eval()

        return generator, kp_detector

# Файл "app\src\ml\services\reconstruction.py":

import os

import cv2
import numpy as np
import torch
from src.ml.datasets.replicate import DataParallelWithCallback
from src.ml.services.logging import LoggingService
from src.ml.services.visualization import VisualizationService
from torch.utils.data import DataLoader
from tqdm import tqdm

log = LoggingService.setup_logger(__name__)


class ReconstructionService:
    @staticmethod
    def reconstruction(config: dict, generator, kp_detector, checkpoint: str, log_dir: str,
dataset) -> None:
        png_dir = os.path.join(log_dir, "reconstruction/png")
        log_dir = os.path.join(log_dir, "reconstruction")

        if not checkpoint:
            raise AttributeError("Checkpoint should be specified for mode='reconstruction'.")
        LoggingService.load_cpk(checkpoint, generator=generator, kp_detector=kp_detector)

        dataloader = DataLoader(dataset, batch_size=1, shuffle=False, num_workers=1)
```

```
        os.makedirs(log_dir, exist_ok=True)
        os.makedirs(png_dir, exist_ok=True)

        loss_list = []
        if torch.cuda.is_available():
            generator = DataParallelWithCallback(generator)
            kp_detector = DataParallelWithCallback(kp_detector)

        generator.eval()
        kp_detector.eval()

        for it, x in tqdm(enumerate(dataloader)):
            if config["reconstruction_params"]["num_videos"] is not None:
                if it > config["reconstruction_params"]["num_videos"]:
                    break
            with torch.no_grad():
                predictions = []
                visualizations = []
                if torch.cuda.is_available():
                    x["video"] = x["video"].cuda()

                kp_source = kp_detector(x["video"][:, :, 0])
                for frame_idx in range(x["video"].shape[2]):
                    source = x["video"][:, :, 0]
                    driving = x["video"][:, :, frame_idx]
                    kp_driving = kp_detector(driving)
                    out = generator(source, kp_source=kp_source, kp_driving=kp_driving)
                    out["kp_source"] = kp_source
                    out["kp_driving"] = kp_driving
                    del out["sparse_deformed"]
                    predictions.append(np.transpose(out["prediction"].data.cpu().numpy(), [0, 2,
3, 1])[0])

                    visualization =
VisualizationService(**config["visualizer_params"]).visualize(
                        source=source, driving=driving, out=out
                    )
                    visualizations.append(visualization)

                    loss_list.append(torch.abs(out["prediction"] - driving).mean().cpu().numpy())

                predictions = np.concatenate(predictions, axis=1)

                image = (255 * predictions).astype(np.uint8)
                image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
                cv2.imwrite(os.path.join(png_dir, x["name"][0] + ".png"), image)

                image_name = x["name"][0] + config["reconstruction_params"]["format"]

                if visualizations:
                    fps = config["reconstruction_params"].get("fps", 30)
                    frame_height, frame_width = visualizations[0].shape[:2]

                    # Determine FourCC based on file extension
                    file_ext = config["reconstruction_params"]["format"].lower()
                    fourcc = {".mp4": "mp4v", ".avi": "XVID", ".mov": "MJPG"}.get(file_ext,
"mp4v")
                    fourcc_code = cv2.VideoWriter_fourcc(*fourcc)
                    video_path = os.path.join(log_dir, image_name)
                    video_writer = cv2.VideoWriter(video_path, fourcc_code, fps, (frame_width,
frame_height))
                    for frame in visualizations:
                        frame_bgr = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
                        video_writer.write(frame_bgr)
                    video_writer.release()

        log.info("Reconstruction loss:", np.mean(loss_list))

# Файл "app\src\ml\services\training.py":

import torch
from src.ml.datasets.dataset_repeater import DatasetRepeater
from src.ml.datasets.replicate import DataParallelWithCallback
from src.ml.modules.discriminator.discriminator_full_model import DiscriminatorFullModel
from src.ml.modules.generator import GeneratorFullModel
from src.ml.services.logging import LoggingService

from torch.optim.lr_scheduler import MultiStepLR
from torch.utils.data import DataLoader
from tqdm import trange
```

```
class TrainingService:
    @staticmethod
    def train(
        config: dict, generator, discriminator, kp_detector, checkpoint: str, log_dir: str,
dataset, device_ids: list
    ) -> None:
        train_params = config["train_params"]

        optimizer_generator = torch.optim.Adam(
            generator.parameters(), lr=train_params["lr_generator"], betas=(0.5, 0.999)
        )
        optimizer_discriminator = torch.optim.Adam(
            discriminator.parameters(), lr=train_params["lr_discriminator"], betas=(0.5, 0.999)
        )
        optimizer_kp_detector = torch.optim.Adam(
            kp_detector.parameters(), lr=train_params["lr_kp_detector"], betas=(0.5, 0.999)
        )

        if checkpoint:
            start_epoch = LoggingService.load_cpk(
                checkpoint,
                generator,
                discriminator,
                kp_detector,
                optimizer_generator,
                optimizer_discriminator,
                None if train_params["lr_kp_detector"] == 0 else optimizer_kp_detector,
            )
        else:
            start_epoch = 0

        scheduler_generator = MultiStepLR(
            optimizer_generator, train_params["epoch_milestones"], gamma=0.1,
last_epoch=start_epoch - 1
        )
        scheduler_discriminator = MultiStepLR(
            optimizer_discriminator, train_params["epoch_milestones"], gamma=0.1,
last_epoch=start_epoch - 1
        )
        scheduler_kp_detector = MultiStepLR(
            optimizer_kp_detector,
            train_params["epoch_milestones"],
            gamma=0.1,
            last_epoch=-1 + start_epoch * (train_params["lr_kp_detector"] != 0),
        )

        if "num_repeats" in train_params or train_params["num_repeats"] != 1:
            dataset = DatasetRepeater(dataset, train_params["num_repeats"])
        dataloader = DataLoader(
            dataset, batch_size=train_params["batch_size"], shuffle=True, num_workers=6,
drop_last=True
        )

        generator_full = GeneratorFullModel(kp_detector, generator, discriminator, train_params)
        discriminator_full = DiscriminatorFullModel(kp_detector, generator, discriminator,
train_params)

        if torch.cuda.is_available():
            generator_full = DataParallelWithCallback(generator_full, device_ids=device_ids)
            discriminator_full = DataParallelWithCallback(discriminator_full,
device_ids=device_ids)

        with LoggingService(
            log_dir=log_dir,
            visualizer_params=config["visualizer_params"],
            checkpoint_freq=train_params["checkpoint_freq"],
        ) as logger:
            for epoch in trange(start_epoch, train_params["num_epochs"]):
                for x in dataloader:
                    losses_generator, generated = generator_full(x)
                    loss_values = [val.mean() for val in losses_generator.values()]
                    loss = sum(loss_values)
                    loss.backward()
                    optimizer_generator.step()
                    optimizer_generator.zero_grad()
                    optimizer_kp_detector.step()
                    optimizer_kp_detector.zero_grad()

                    if train_params["loss_weights"]["generator_gan"] != 0:
                        optimizer_discriminator.zero_grad()
```

```
                        losses_discriminator = discriminator_full(x, generated)
                        loss_values = [val.mean() for val in losses_discriminator.values()]
                        loss = sum(loss_values)

                        loss.backward()
                        optimizer_discriminator.step()
                        optimizer_discriminator.zero_grad()
                    else:
                        losses_discriminator = {}

                    losses_generator.update(losses_discriminator)
                    losses = {key: value.mean().detach().data.cpu().numpy() for key, value in
losses_generator.items()}
                    logger.log_iter(losses=losses)

                scheduler_generator.step()
                scheduler_discriminator.step()
                scheduler_kp_detector.step()

                logger.log_epoch(
                    epoch,
                    {
                        "generator": generator,
                        "discriminator": discriminator,
                        "kp_detector": kp_detector,
                        "optimizer_generator": optimizer_generator,
                        "optimizer_discriminator": optimizer_discriminator,
                        "optimizer_kp_detector": optimizer_kp_detector,
                    },
                    inp=x,
                    out=generated,
                )

# Файл "app\src\ml\services\video_animation.py":

from pathlib import Path

import cv2
import numpy as np
from skimage.util import img_as_ubyte
from src.ml.services.animation import AnimationService
from src.ml.services.logging import LoggingService
from src.ml.services.model import ModelService
from src.ml.services.utils import find_best_frame, load_video, preprocess_image


class VideoAnimationService:
    def __init__(
        self,
        config_path: str,
        checkpoint_path: str,
        source_image_path: str,
        driving_video_path: str,
        result_video_path: str,
        relative: bool = False,
        adapt_scale: bool = False,
        find_best: bool = False,
        best_frame: int | None = None,
        cpu: bool = False,
    ) -> None:
        self.log = LoggingService.setup_logger(__name__)
        self._validate_paths(source_image_path, driving_video_path)
        self.config_path = config_path
        self.checkpoint_path = checkpoint_path
        self.source_image_path = source_image_path
        self.driving_video_path = driving_video_path
        self.result_video_path = result_video_path
        self.relative = relative
        self.adapt_scale = adapt_scale
        self.find_best = find_best
        self.best_frame = best_frame
        self.cpu = cpu

        model_service = ModelService(config_path, checkpoint_path, cpu=cpu)
        self.generator, self.kp_detector = model_service.load_eval_models()
        self._log_init()

    @staticmethod
    def _validate_paths(*paths: str) -> None:
        for path in paths:
```

```
            if not Path(path).exists():
                raise FileNotFoundError(f"Path does not exist: {path}")

    def _log_init(self) -> None:
        self.log.info(f"Initialized VideoAnimationService with:")
        self.log.info(f"Source: {self.source_image_path}")
        self.log.info(f"Driving: {self.driving_video_path}")
        self.log.info(f"Output: {self.result_video_path}")
        self.log.info(f"Settings: relative={self.relative}, adapt_scale={self.adapt_scale}")

    def run(self) -> None:
        try:
            source_image = preprocess_image(self.source_image_path)
            driving_video, fps = load_video(self.driving_video_path)
            predictions = self._generate_animations(source_image, driving_video)
            self._save_video(predictions, fps)
        except Exception as e:
            self.log.error(f"Animation failed: {str(e)}")
            raise

    def _generate_animations(self, source_image: np.ndarray, driving_video: list[np.ndarray]) ->
list[np.ndarray]:
        if not self.find_best and self.best_frame is None:
            return AnimationService.make_animation(
                source_image,
                driving_video,
                self.generator,
                self.kp_detector,
                relative=self.relative,
                adapt_movement_scale=self.adapt_scale,
                cpu=self.cpu,
            )

        i = (
            self.best_frame
            if self.best_frame is not None
            else find_best_frame(source_image, driving_video, cpu=self.cpu)
        )
        self.log.info(f"Using frame {i} as best match")

        # Split video at best frame
        driving_forward = driving_video[i:]
        driving_backward = driving_video[: i + 1][::-1]

        # Process both segments
        predictions_forward = AnimationService.make_animation(
            source_image,
            driving_forward,
            self.generator,
            self.kp_detector,
            relative=self.relative,
            adapt_movement_scale=self.adapt_scale,
            cpu=self.cpu,
        )
        predictions_backward = AnimationService.make_animation(
            source_image,
            driving_backward,
            self.generator,
            self.kp_detector,
            relative=self.relative,
            adapt_movement_scale=self.adapt_scale,
            cpu=self.cpu,
        )

        return predictions_backward[::-1] + predictions_forward[1:]

    def _save_video(self, frames: list[np.ndarray], fps: float) -> None:
        if not frames:
            raise ValueError("No frames to save")

        height, width = frames[0].shape[:2]
        fourcc = cv2.VideoWriter_fourcc(*"mp4v")

        out = None
        try:
            out = cv2.VideoWriter(self.result_video_path, fourcc, fps, (width, height))

            if not out.isOpened():
                raise IOError("Could not open video writer")
```

```
            for frame in frames:
                frame_bgr = cv2.cvtColor(img_as_ubyte(frame), cv2.COLOR_RGB2BGR)
                out.write(frame_bgr)

            self.log.info(f"Successfully saved video to {self.result_video_path}")
        finally:
            out.release() if "out" in locals() else None

# Файл "app\src\server\routers.py":

import logging
import os
import shutil
from tempfile import NamedTemporaryFile
from typing import Iterator
from fastapi import APIRouter, File, HTTPException, UploadFile
from src.ml.services.video_animation import VideoAnimationService
from starlette.responses import StreamingResponse

router = APIRouter(prefix="/fomm", tags=["First Order Motion Model"])
log = logging.getLogger(__name__)

def stream_and_cleanup(filepath: str) -> Iterator[bytes]:
    try:
        with open(filepath, "rb") as f:
            yield from iter(lambda: f.read(8192), b"")
    finally:
        if os.path.exists(filepath):
            os.remove(filepath)

@router.post("/video")
async def animate_image_by_video(
    source_image: UploadFile = File(...),
    driving_video: UploadFile = File(...),
):
    try:
        log.info("Save temporary files")
        with NamedTemporaryFile(suffix=".png", delete=False) as temp_image_file,
NamedTemporaryFile(
            suffix=".mp4", delete=False
        ) as temp_video_file:

            shutil.copyfileobj(source_image.file, temp_image_file)
            shutil.copyfileobj(driving_video.file, temp_video_file)
            temp_image_path = temp_image_file.name
            temp_video_path = temp_video_file.name

        output_video_path = temp_video_path.replace(".mp4", "_result.mp4")
        log.info(f"Path to final output video: {output_video_path}")

        service = VideoAnimationService(
            config_path="./data/configs/vox-256.yaml",
            checkpoint_path="./data/checkpoints/vox-cpk.pth.tar",
            source_image_path=temp_image_path,
            driving_video_path=temp_video_path,
            result_video_path=output_video_path,
            relative=False,
            adapt_scale=False,
            find_best=False,
            best_frame=None,
            cpu=True,
        )
        log.info("Run image animation service")
        service.run()

        return StreamingResponse(stream_and_cleanup(output_video_path), media_type="video/mp4")

    except Exception as e:
        log.error(f"Failed to process video: {e}")
        raise HTTPException(status_code=500, detail=f"Failed to process video: {e}")

    finally:
        log.info("Remove temp files")
        for path in [locals().get("temp_image_path"), locals().get("temp_video_path")]:
            if path and os.path.exists(path):
                os.remove(path)
```

**ПРИЛОЖЕНИЕ Б**
(обязательное)

**Спецификация**

**ПРИЛОЖЕНИЕ В**
(обязательное)

**Ведомость документов**