МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

РАБОТА ПРОВЕРЕНА

Рецензент
Начальник отдела
суперкомпьютерного моделирования
НИУ ВШЭ, к.ф.-м.н., доцент

_____ П.С. Костенецкий

«___»_____ 2024 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка приложения для генерации валидированных изображений на основе стандарта C2PA с использованием генеративных нейронных сетей

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.04.04.2024.308-1484.ВКР

Научный руководитель,
доцент кафедры СП, к.ф.-м.н.
_____ Г.И. Радченко

Автор работы,
студент группы КЭ-228
_____ Д.В. Стародубцев

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы магистранта

студенту группы КЭ-228
Стародубцеву Дмитрию Владимировичу,
обучающемуся по направлению
09.04.04 «Программная инженерия»
(магистерская программа «Искусственный интеллект и инженерия данных»)

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка приложения для генерации валидированных изображений на основе стандарта C2PA с использованием генеративных нейронных сетей.
2. **Срок сдачи студентом законченной работы:** 20.05.2024 г.
3. **Исходные данные к работе**

3.1. Content Credentials: C2PA Technical Specification. [Электронный ресурс] URL: https://c2pa.org/specifications/specifications/2.0/specs/C2PA_Specification (дата обращения: 01.02.2024 г.).
3.2. Vayadande K., Bhemde S. AI-Based Image Generator Web Application using OpenAI’s DALL-E System. // 2023 International Conference on Recent Advances in Science and Engineering Technology (ICRASET), 2023. – 20 p.
3.3. Сюй А. System Design. // СПб.: Питер, 2023. – 304 с.

4. **Перечень подлежащих разработке вопросов**

4.1. Провести анализ предметной области.
4.2. Провести обзор научной литературы.
4.3. Проектирование архитектуры библиотеки для стандарта C2PA.
4.4. Реализовать библиотеку для стандарта C2PA на языке программирования Python.

4.5. Выбрать модель генеративной нейронной сети.
4.6. Проектирование архитектуры приложения.
4.7. Реализация приложения.
4.8. Тестирование приложения.
5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.ф.-м.н. Г.И. Радченко

Задание принял к исполнению Д.В. Стародубцев

ГЛОССАРИЙ

1. *C2PA (Coalition for Content Provenance and Authenticity)* – это коалиция, созданная для борьбы с дезинформацией в онлайн-среде. Она занимается разработкой и внедрением стандартов и технологий, позволяющих определять происхождение контента в цифровом пространстве и подтверждать его аутентичность [1].

2. *Метаданные* – это данные, которые предоставляют информацию о других данных. Они описывают различные аспекты данных, такие как их структура, формат, контекст или характеристики [2].

3. *Контент* – разнообразные виды информации, которые могут быть созданы и распространены в цифровой или физической форме [3].

4. *Хэширование* – это процесс преобразования входных данных произвольной длины в фиксированную строку фиксированной длины. Эта строка, называемая хэшем или хэш-значением, является результатом применения хэш-функции к исходным данным [4].

5. *Сертификат* – это электронный документ, который подтверждает определенную информацию о субъекте, таком как организация, веб-сайт, пользователь или устройство. Обычно сертификат содержит информацию о ключе и его владельце, а также цифровую подпись, которая гарантирует подлинность этой информации [5].

6. *Приватный ключ* – это конфиденциальная строка символов, используемая в криптографии для создания и проверки цифровых подписей, а также для расшифровки зашифрованных данных [6].

7. *Цифровая подпись* – это электронная метка, создаваемая с использованием криптографической ключевой пары (приватного и открытого ключей), которая позволяет проверить подлинность и целостность электронного документа, сообщения или данных [7].

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В наше время цифровой контент играет важнейшую роль в обществе. Он пронизывает практически все сферы жизни, от образования и науки до развлечений и социальных коммуникаций. Однако с расширением использования цифровых технологий возрастает и важность вопросов, связанных с защитой такого контента.

На протяжении последних лет мы стали свидетелями значительного прогресса в области нейронных сетей. В частности, развитие генеративных нейронных сетей (GANs) [8] позволило нам создавать все более реалистичные изображения, что открывает новые горизонты для творческих и инновационных проектов. Однако эта технология создает и новые угрозы. В частности, возникают риски, связанные с возможностью недобросовестной модификации или подделки изображений. Например, в 2023-м году имел место быть случай на конкурсе фотографии Sony World Photography Awards, где победителем стал фотограф, подавший на конкурс фотографию (рисунок 1), сгенерированную нейронной сетью, в чем после сам и признался [9].

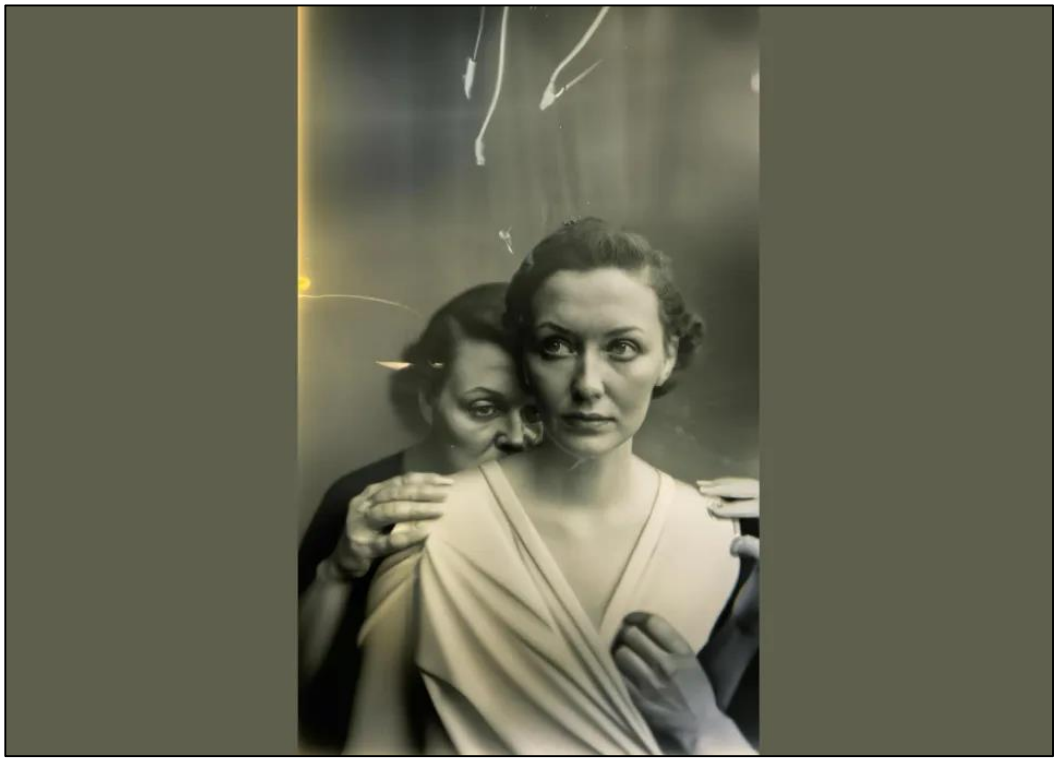

Рисунок 1 – Фотография, сгенерированная нейронными сетями

Представленную фотографию действительно легко перепутать с реальной фотографией и даже сложно представить, что нейронная сеть могла создать настолько подлинный результат. Тем самым фотограф поднял вопрос о значимости влияния нейронных сетей в текущем мире фотографии и возбудил общественность.

В этом контексте становится критически важным обеспечение подлинности и защиты цифрового контента, генерируемого с помощью генеративных нейронных сетей. Важность этой задачи не может быть преувеличена. Ведь подлинность цифрового контента напрямую влияет на точность и надежность информации, которой мы пользуемся каждый день. Одним из ярких примеров являются социальные сети и медиа ресурсы, которые каждый день распространяют информацию о новостях и сложившихся ситуациях в мире. Именно поэтому важно, чтобы контент, которым делятся подобные ресурсы, можно было проверить на подлинность и убедиться в благонадежности рассматриваемого ресурса.

Для решения этой проблемы существуют различные подходы и технологии, включая стандарты верификации контента, такие как C2PA [1]. Они позволяют подтвердить происхождение цифрового контента и установить его подлинность, что обеспечивает надежность и прозрачность в отношении цифрового контента.

Защита и верификация цифрового контента, сгенерированного с помощью генеративных нейронных сетей, являются сложной и актуальной задачей. Решение этой задачи требует глубокого понимания технологий и стандартов, а также способности применять их на практике при разработке приложений и инструментов.

Постановка задачи

Целью выпускной квалификационной работы является разработка приложения для генерации валидированных изображений на основе стандарта C2PA с использованием генеративных нейронных сетей. Разработан-

ное приложение должно быть в открытом доступе, чтобы показать на примере простоту использования средств защиты контента. Для достижения поставленной цели необходимо решить следующие задачи:

1) провести анализ предметной области;
2) провести обзор научной литературы;
3) проектирование архитектуры библиотеки для стандарта C2PA;
4) реализовать библиотеку для стандарта C2PA на языке программирования Python;
5) выбрать модель генеративной нейронной сети;
6) проектирование архитектуры приложения;
7) реализация приложения;
8) тестирование приложения.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 55 страниц, объем списка литературы – 30 источников.

В первой главе описывается обзор предметной области.

Вторая глава посвящена процессу разработки библиотеки для реализации стандарта C2PA на языке программирования Python.

В третьей главе описывается процесс проектирования приложения.

В четвертой главе описывается реализация приложения.

Пятая глава посвящена тестированию приложения.

В приложении А приведены листинги, содержащие исходный код разрабатываемой библиотеки «C2Py».

В приложении Б приведены листинги, содержащие исходный код разрабатываемой системы.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Обзор аналогов

На данный момент существует несколько сервисов, которые позволяют генерировать подлинные изображения с помощью генеративных нейронных сетей.

Во-первых, Adobe Firefly [10], разработанной компанией Adobe (рисунок 2). Данный сервис позволяет генерировать изображения по описанию, но это не главная особенность данного сервиса. Сгенерированное изображение подписывается в соответствии с C2PA стандартом. Это позволяет удостовериться, что полученное изображение действительно является сгенерированным нейронной сетью.

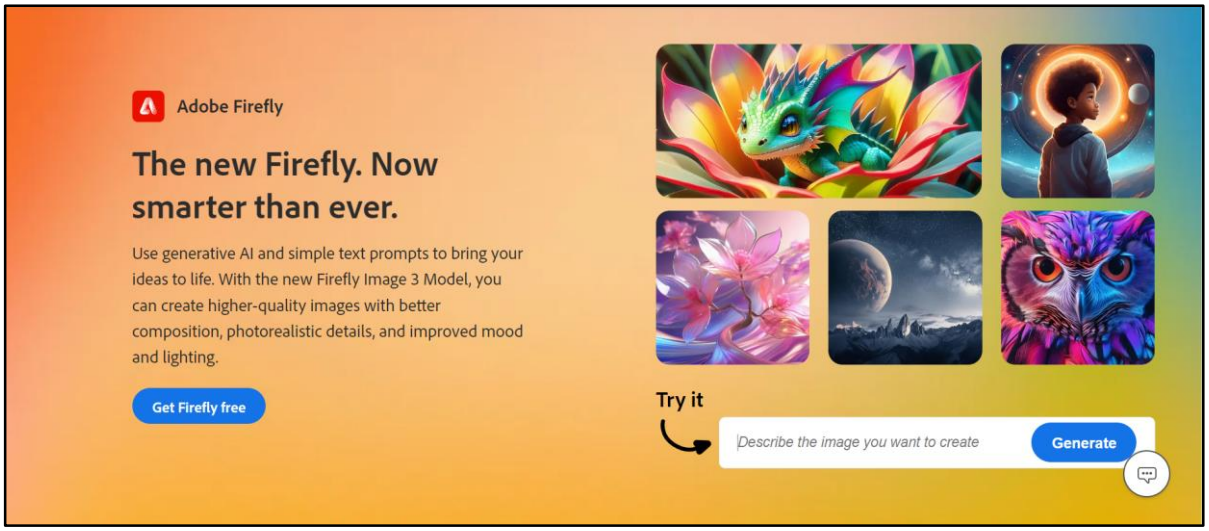

Рисунок 2 – Веб-интерфейс платформы Adobe Firefly

Во-вторых, существует аналогичный сервис под названием Image creator (рисунок 3), созданный компанией Microsoft [11]. Данный сервис позволяет генерировать изображения по описанию с помощью нейронной сети DALL-E. Помимо этого, данный сервис поддерживает стандарт C2PA, что позволяет верифицировать тот факт, что данное изображение было сгенерировано данной нейронной сетью. Это поможет избежать введение в заблуждение пользователей, которые обнаружат сгенерированное изображение.

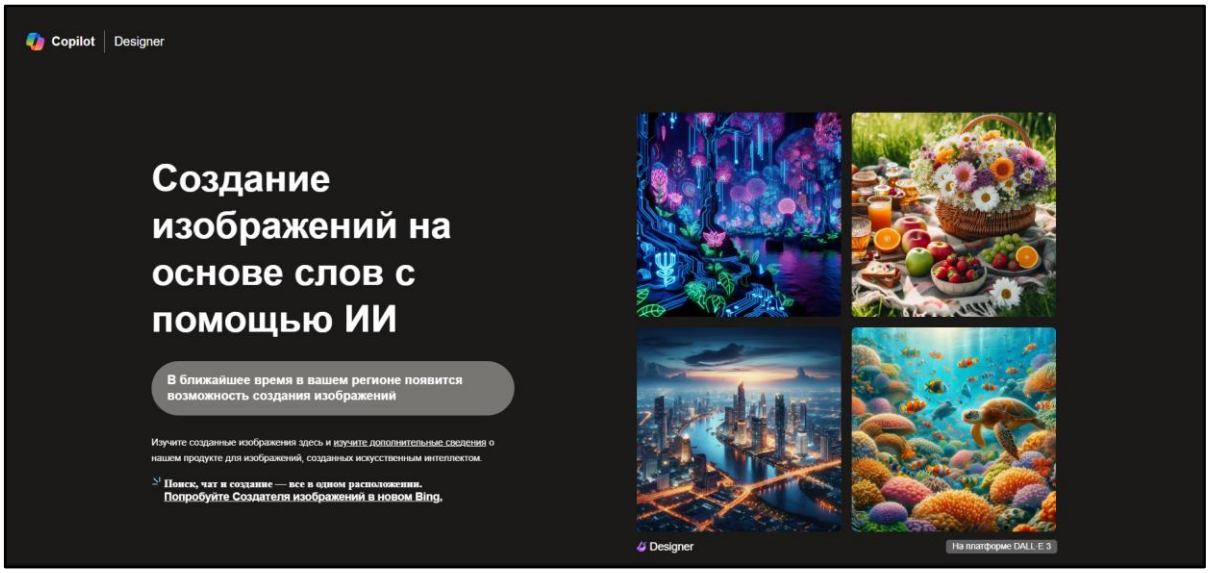

Рисунок 3 – Веб-интерфейс платформы Microsoft Image Creator

Из минусов описанных выше сервисов стоит отметить, что они недоступны для нашего региона.

В-третьих, Kandinsky [12] (рисунок 4), разработанный командой FusionBrain.

Рисунок 4 – Веб-интерфейс платформы Kandinsky

Данный сервис предоставляет широкий функционал, он позволяет работать с изображениями, видео и анимацией. С помощью Kandinsky возможно создавать изображения с нуля, подавая на вход описание желаемого,

помимо этого есть возможность дорабатывать существующие исходные данные. Также стоит отметить, что Kandinsky является разработкой российской компании, поэтому доступ к сервису из нашего региона осуществляется без проблем. Еще одним важным достоинством данного сервиса является открытый бесплатный API.

1.2. Обзор научной литературы

Авторами статьи «DALL-E: Creating images from text» [13] прилагается описание архитектуры модели нейронной сети DALL-E (Differential Adversarial Latent Learned Encoder). Схема сети представлена на рисунке 5.

Архитектура модели представляет собой трансформерную языковую модель на базе GPT-3 (Generative Pre-trained Transformer), получающую на вход текст и изображение в едином потоке до 1280 токенов. Перед подачей на вход изображения предварительно обрабатываются до разрешения 256 на 256 пикселей. Помимо этого, применяется метод сжатия с использованием VQVAE (Vector Quantized Variational Autoencoder) для получения дискретных кодов, позволяющих генерировать изображения с нуля и восстанавливать участки существующих изображений. Обучалась данная модель на двенадцати миллиардах параметрах.

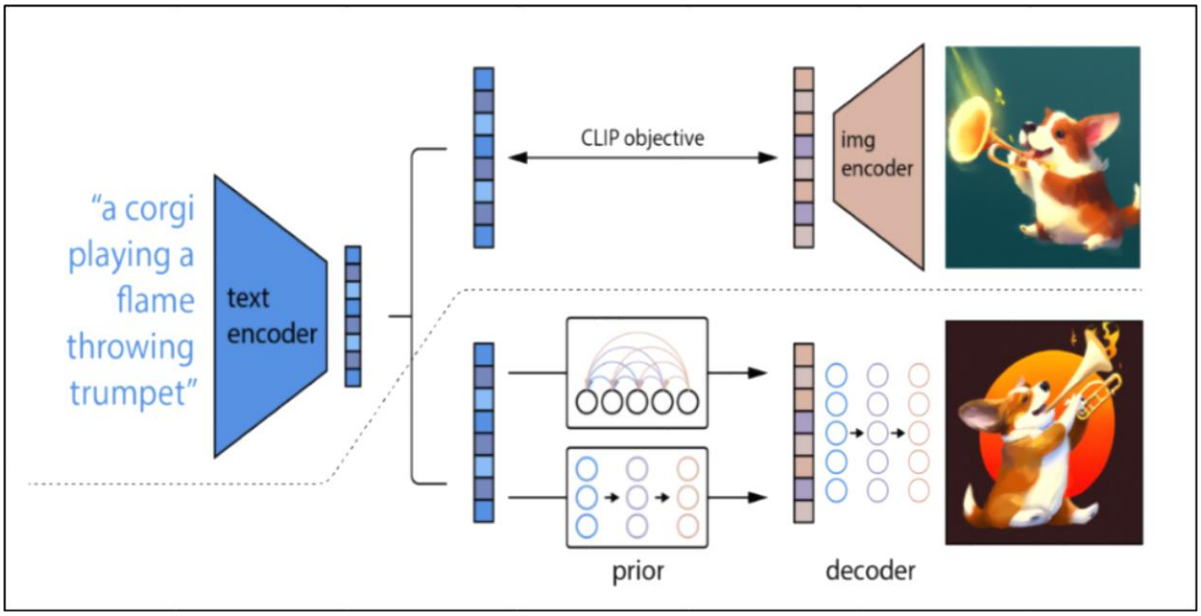

Рисунок 5 – Пример работы DALL-E

Главное предназначение этого решения – создание изображений по текстовым описаниям с использованием набора данных из пар текст-изображение.

Данная модель предоставляет огромное множество возможностей для генерации изображений, анимаций, стилей по текстовому описанию. Также еще одной ключевой особенностью описываемого решения является открытый API для взаимодействия с моделью, что позволяет использовать представленный инструмент без задействования больших личных мощностей.

В статье «The Use of Artificial Intelligence Art Generator Midjourney in Artistic and Advertising Creativity» [14] авторы приводят сравнение моделей Midjourney и DALL-E, в рамках которого говорится, что по своей сути подход к работе у данных моделей схожий, но Midjourney дает более обширные возможности для генерации изображения, в частности, более высокое разрешение исходных изображений (2048 на 2048 против 1024 на1024 у DALL-E).

Midjourney отличается от DALL-E тем, что комбинирует метод CLIP (Contrastive Language-Image Pre-training) с постоянно изменяющимся набором. DALL-E же хорошо работает с крупными сценами, создавая более детализированные, изображения с высоким разрешением. Они также различаются по количеству параметров: Midjourney имеет 3.5 миллиарда параметров, в то время как DALL-E – более двенадцати миллиардов параметров.

Описанная модель является лучшей версией рассмотренной ранее модели DALL-E, но имеет ряд недостатков, главный из них, на мой взгляд, это невозможность использовать бесплатную версию, что создает закрытую инфраструктуру вокруг данного решения. Также у решения нет открытого API, который бы позволил пользоваться возможностями модели, что усложняет процесс знакомства с решением, а также дальнейшее взаимодействие на некоммерческой основе.

Авторы статьи «Kandinsky: An Improved Text-to-Image Synthesis with Image Prior and Latent Diffusion» [15] описывают архитектуру модели и процесс обработки запросов в сервисе Kandinsky. Авторы описывают подход с приоритетом изображения, который комбинирует диффузию и линейные отображения между входным текстом и вложениями изображениями. Модель включает три этапа: кодирование текста, отображение вложений изображения (приоритет изображения) и латентная диффузия. Авторы обучили модель трансформатор-кодировщик на вложениях текста и изображений CLIP-ViT-L14. Для ускорения сходимости процесса диффузии они использовали нормализацию визуальных вложений. Модель латентной диффузии использует комбинацию условных сигналов, включая вложения CLIP-изображений, текстовые вложения CLIP и текстовые вложения XLMR-CLIP. Важно отметить, что авторы не пропускали этап квантизации автоэнкодера во время вывода диффузии для увеличения разнообразия и качества сгенерированных изображений.

1.3. Обзор подходов для защиты контента

В ходе работы были выделены два подхода для защиты контента AMP (Authentication of Media via Provenance) [16] и C2PA (Content Authenticity Protection Alliance) [1].

В первую очередь в качестве подхода для защиты контента рассмотрим стандарт C2PA [1], разработанный коалицией компаний CAI (The Content Authenticity Initiative). Ключевой идеей C2PA является создание надежной системы, отслеживающей каждый этап жизненного цикла цифрового контента – от момента создания до окончательной формы. Эта «цепь хранения» аналогична цифровому отпечатку пальца и обеспечивает прозрачное отслеживание и верификацию любых изменений, внесенных в контент. Важно подчеркнуть, что фокус C2PA не заключается в решении вопросов целостности контента, а в предоставлении пользователям точной и

верифицируемой информации об источнике контента и внесенных в него изменениях.

Техническая часть стандарта основывается на цифровой подписи и хешированию участков контента, предназначенных для защиты. К рассчитанному хэшу может добавляться и другая информация, например, информация об авторе контента и так далее. После чего добавленная информация вместе с хэшем соединяется в единую структуру и подписывается с помощью цифровой подписи. Данный подход позволяет сохранить в метаданных контента информацию об оригинале, включая части самого контента, его автора и т.п. Помимо этого, в метаданные контента возможно сохранить информацию о редактировании контента.

На данный момент разработкой стандарта занимается большое количество компаний по всему миру, например, Adobe, Google, Microsoft, Intel, Nvidia и т.д. Этот факт придает уверенности в развитии стандарта в будущем. В статье «Restoring Trust in Online Content and Demystifying Deepfakes with C2PA Credentials» [17] говорится, что данные стандарт поможет изменить подход обращения пользователей с контентом в сети Интернет. Это позволит пользователям самостоятельно проверять источники информации и определять, что является дезинформацией, а что нет.

Стоит отметить, что данный стандарт имеет открытую спецификацию, которая постоянно обновляется. Помимо этого, CAI (Content Authenticity Initiative) выпустила несколько публично доступных библиотек и инструментов для взаимодействия со стандартом.

Вторым подходом для защиты контента является AMP система, разработанная компанией Microsoft. Данная система реализует подход к обеспечению аутентичности медиаконтента (такого как изображения, видео и звуковые файлы), используя информацию о его происхождении и истории. Основная идея заключается в том, чтобы прикреплять к медиаконтенту цифровые метки или метаданные, которые предоставляют информацию о его источнике, авторе, дате создания, изменениях и путях распространения.

AMP стремится к тому, чтобы медиаконтент был снабжен такой информацией с самого момента его создания и вплоть до момента его использования конечным пользователем. Это позволяет проверить аутентичность контента и определить его происхождение [16].

Основные технические решения AMP базируются на принципах, которые описаны в стандарте C2PA (такие как цифровая подпись, алгоритмы хэширования, составление метаданных контента). Помимо этого, создатели подхода AMP входят в состав основателей коалиции CAI, что свидетельствует о схожести описываемых подходов.

Выводы по первой главе

В ходе анализа предметной области были рассмотрены архитектуры популярных нейронных сетей для генерации изображений. Исходя из рассмотренных аналогов и обзора научной литературы, было решено воспользоваться открытым API для работы с моделью Kandinsky. Это позволит использовать передовую генеративную модель для создания изображений по текстовому описанию в условиях недостаточных вычислительных мощностей.

Помимо этого, был проведен анализ подходов для защиты контента, в рамках которого были рассмотрены два метода, опирающиеся на схожие технические решения, но имеющие разную степень распространенности. Предпочтение было отдано стандарту C2PA, обусловленному более широким объемом доступных исходных данных для исследований, включая открытую техническую документацию и публичные репозитории с исходным кодом инструментов, основанных на данном стандарте.

2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1. Теоретические основы стандарта C2PA

Стандарт C2PA создан для подтверждения достоверности контента и отслеживания действий, произведенным над ним. Данный стандарт описывает возможность добавления метаданных определенного формата в множество различных типов контента, например, изображения, видео, аудио и т.д. Метаданные, описанные стандартом C2PA содержат необходимую информацию для подтверждения подлинности контента, установление авторства, отслеживание истории изменений, произведенных над контентом и т.п.

Процесс создания метаданных контента по стандарту C2PA изображен на рисунке 6.

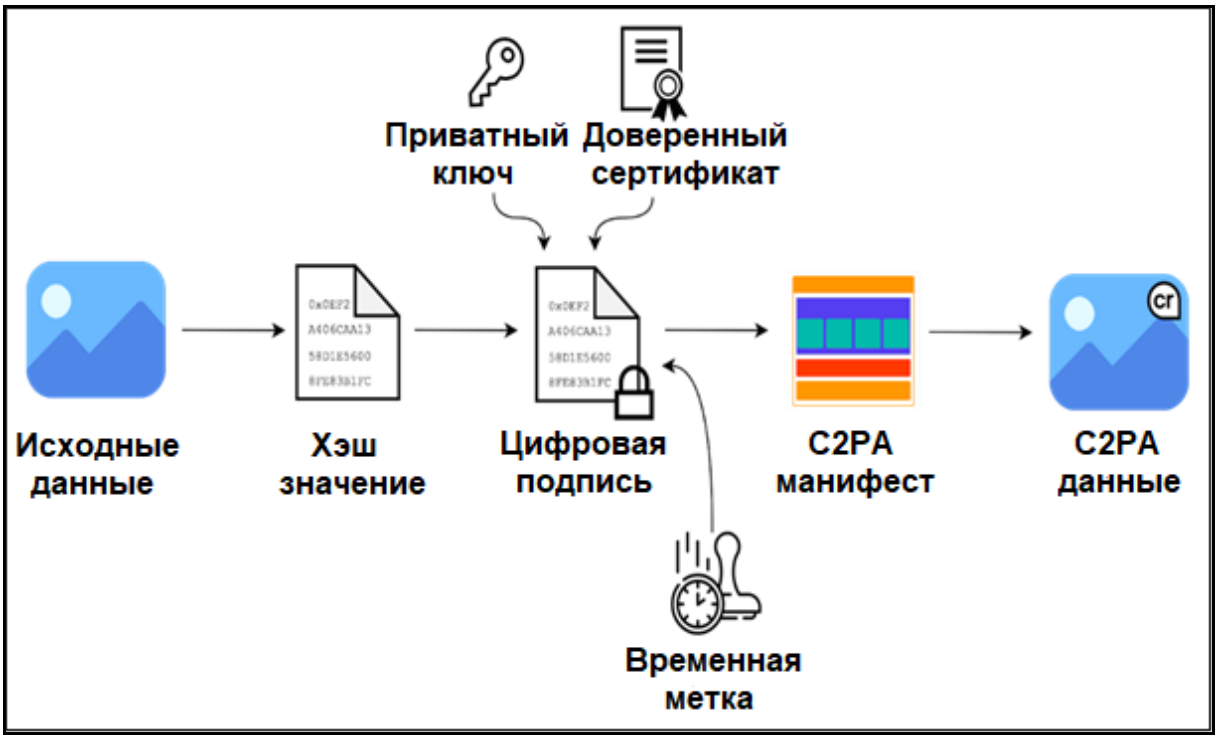

Рисунок 6 – Создание метаданных по стандарту C2PA

Процесс формирования метаданных должен быть инициирован только после получения конечного контента или частей контента, которые в дальнейшем не будут подвергаться изменениям. Это необходимо для обеспечения совместимости информации, содержащейся в метаданных, и контента. Первым шагом для построения структуры C2PA является создание

уникального набора символов по средствам преобразования контента, это возможно сделать с помощью алгоритмов хэширования. Стандарт C2PA [1] поддерживает несколько алгоритмов хэширования, представленных ниже.

1. SHA2-256.
2. SHA2-384.
3. SHA2-512.

Результат работы подобных алгоритмов является строка байт, размер которой зависит от выбранной реализации.

Следующий этап – создание цифровой подписи контента. Цифровая подпись – это метод аутентификации сообщений, документов или данных в цифровой форме. Она использует асимметричное шифрование для создания и проверки электронной подписи. Процесс создания цифровой подписи включает шифрование хэшированных данных закрытым ключом отправителя. Получатель может затем проверить цифровую подпись с использованием открытого ключа отправителя, расшифровав хэш-код и сравнив его с хэш-кодом полученного сообщения. Если хэш-коды совпадают, это подтверждает подлинность сообщения и его непосредственную отправку от владельца закрытого ключа. Цифровая подпись обеспечивает аутентификацию, целостность и невозможность отказа отправителя [18]. Поскольку стандарт C2PA предполагает верификацию подписи метаданных не только во временных рамках действия сертификата, но и спустя продолжительное время, целесообразно использовать долговременные цифровые подписи, которые предполагают использование доверенной временной метки. Доверенная временная метка (trusted timestamp) – это электронная метка, которая подтверждает временной момент, когда определенная информация была создана, отправлена или получена. Это обеспечивает доказательство того, что подпись была произведена на определенном этапе времени. Тем самым, временная метка позволяет определить был ли сертификат валиден во время создания электронной подписи. Стоит отметить, что в данной работе рассматривается исключительно базовое представление о цифровой подписи,

направленное на ознакомление с возможностями использования данного подхода для обеспечения защиты данных в рамках стандарта C2PA. Для получения юридически верной цифровой подписи следует обратиться к соответствующим государственным стандартам.

Важным этапом в создании метаданных является формирование данных в определенную структуру, описанную в стандарте (рисунок 7).

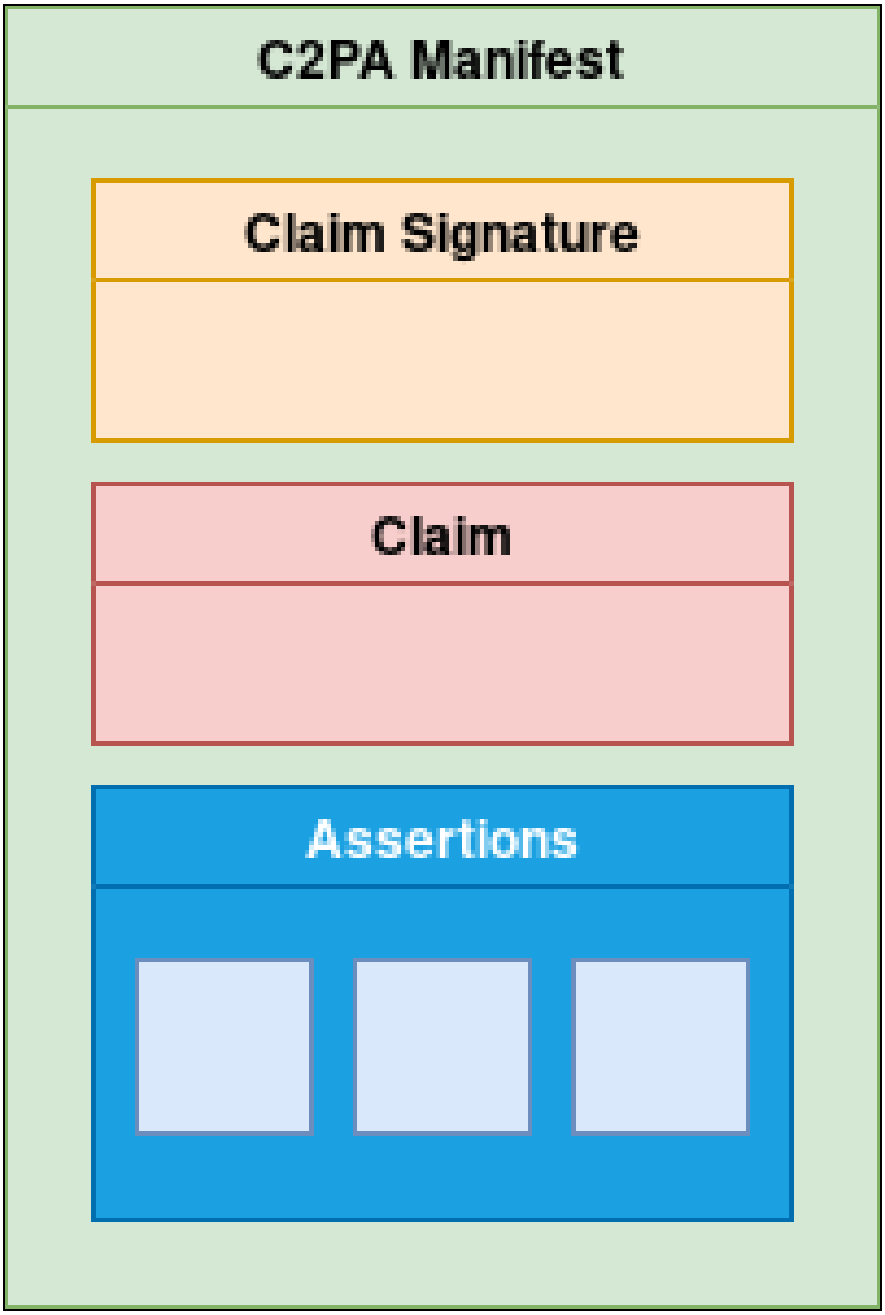

Рисунок 7 – Структура метаданных по стандарту C2PA

Метаданные C2PA содержат ряд инструкций, которые охватывают такие области, как создание ресурсов, авторство, действия редактирования, сбор сведений об устройстве, привязки к контенту и многие другие темы. Эти утверждения (assertions) определяют происхождение данного актива и представляют собой серию сигналов доверия, которые могут быть использованы человеком для улучшения своего представления о надежности актива. Утверждения объединяются с дополнительной информацией в объект, содержащий краткую информацию обо всех утверждениях (assertions) в виде хэша, которая в дальнейшем подтверждается с помощью цифровой подписи и записывается в объект, называемый утверждением (claim signature).

Все эти утверждения, формулы, учетные данные и подписи объединяются в поддающийся проверке блок, называемый манифестом C2PA, с помощью аппаратного или программного компонента, называемого генератором утверждений. Набор манифестов C2PA, хранящихся в хранилище манифестов C2PA актива, представляет данные о его происхождении [1].

Заключительным этапом является преобразование построенной структуру данных C2PA в бинарное представление, для этого используется формат JUMBF [19]. JUMBF определяет структурированный формат данных, который состоит из различных полей и значений, предназначенных для описания метаданных контента. Эти поля включают информацию о контенте, такую как название, автор, дата создания, тип контента, разрешение, длительность (для аудио или видео), использованные кодеки и многое другое. JUMBF предоставляет стандартизированные наборы полей метаданных для различных типов контента. Это способствует согласованности и обеспечивает легкость в интерпретации данных, которые передаются или хранятся в формате JUMBF. Данный формат описывает использование концепции контейнеров для организации данных, ниже представлены некоторые из них.

1. Header Box – контейнер, который содержит метаданные заголовка, такие как версия формата, дата создания и другие основные сведения о данных JUMBF.

2. Metadata Box – контейнер, предназначенный для хранения основных метаданных о контенте, такие как название, автор, дата создания и другие связанные с контентом атрибуты.

3. Format Metadata Box – контейнер, который содержит информацию о формате данных JUMBF, включая определения полей, типы данных, размеры и другие технические аспекты.

4. Additional Metadata Box – здесь могут храниться дополнительные метаданные, не относящиеся к основному описанию контента или формату. Например, ключевые слова, теги, категории и другие дополнительные атрибуты.

5. Authorship and Rights Metadata Box – этот контейнер содержит информацию об авторских правах, лицензировании и других правовых аспектах, связанных с контентом.

6. Location Metadata Box – в этом контейнере могут храниться метаданные о местоположении, такие как географические координаты или другая информация о местоположении, связанная с контентом.

7. Description Metadata Box – здесь может быть описание или комментарии к контенту.

Полученный манифест переводится в байтовое представление, после чего вставляется в необходимый формат данных по определенным правилам. Например, для изображений формата JPG C2PA манифест оформляется в качестве APP11 контейнера и помещается в любое место файла между существующими сегментами.

2.2. Обзор принципов работы трансформенных нейронных сетей

Трансформеры – это тип нейронных сетей, который стал популярным благодаря своей способности эффективно обрабатывать последовательности данных, такие как тексты, аудио или видео. Они обладают рядом особенностей, включая механизм внимания, который позволяет модели фокусироваться на различных аспектах входных данных в зависимости от их важности.

Принцип работы трансформеров заключается в нескольких аспектах: механизм внимания, многослойная архитектура, позиционное кодирование, функция потерь и оптимизация.

Механизм внимания

В трансформерах используется механизм внимания для того, чтобы модель могла фокусироваться на различных частях входных данных в зависимости от их важности.

Во-первых, для каждого элемента в последовательности модель вычисляет вес, который отражает его важность относительно других элементов. Это происходит путем скалярного произведения векторов представления элементов и весовых матриц. Затем веса нормализуются с использованием функции softmax, чтобы получить вероятностное распределение.

Во-вторых, после вычисления весов модель умножает веса на соответствующие векторы представления элементов и складывает полученные произведения, чтобы получить взвешенное представление входных данных.

Многослойная архитектура

Трансформеры обычно состоят из нескольких одинаковых блоков, каждый из которых обрабатывает данные последовательности. Каждый блок включает в себя несколько слоев. Первый слой – это слой внимания. Данный слой вычисляет веса внимания внутри одной последовательности, что позволяет модели учитывать зависимости между различными элемен-

тами в последовательности. После слоя внимания следует несколько полносвязанных слоев, которые применяются к каждому элементу последовательности независимо.

Позиционное кодирование

Поскольку трансформеры не учитывают порядок элементов в последовательности, необходимо добавить информацию о позициях элементов. Это обычно делается путем добавления позиционных кодировок к векторам представления элементов. Позиционные кодировки представляют собой специальные векторы, которые зависят от позиции элемента в последовательности и добавляются к векторам представления элементов.

Функция потерь и оптимизация

После обучения модели необходимо определить функцию потерь, которая измеряет расхождение между предсказанными и истинными значениями. После этого используется алгоритм оптимизации, такой как Adam, который обновляет параметры модели, уменьшая функции потерь.

Таким образом, трансформеры обеспечивают эффективную обработку последовательностей данных, позволяя модели фокусироваться на важных аспектах данных и учитывать их порядок [20].

Выводы по второй главе

В данной главе были рассмотрены теоретические основы стандарта C2PA, а именно процесс создания C2PA манифеста с описанием каждого этапа. Также было предоставлено описание структуры C2PA манифеста и всех составных частей. Было рассказано про преобразование метаданных в формат JUMBF, и описана его структура. Помимо этого, были рассмотрены особенности трансформенных нейронных сетей.

3. ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

В рамках данной работы проводится разработка микросервисного приложения «Kandinsky Captions», которое позволяет пользователю генерировать валидированные изображения по введенному описанию, используя генеративные нейронные сети и способ валидации данных на основе стандарта C2PA. Название для системы было выбрано, основываясь на выбранных подходах для генерации изображений и подтверждения авторства.

3.1. Функциональные и нефункциональные требования

Функциональные требования необходимы для описания поведения системы «Kandinsky Captions». Данные требования важны для представления основных частей системы и их взаимодействия [21]. Для разрабатываемой системы можно определить следующие функциональные требования.

1. Система должна предоставлять пользователю возможность вводить текст, описывающий желаемое изображение.
2. Система должна генерировать изображение по описанию пользователя.
3. Система должна создавать метаданные на основе стандарта C2PA.
4. Система должна встраивать созданные метаданные в сгенерированное изображение.
5. Система должна отображать созданное изображение в графическом интерфейсе.
6. Система должна предоставить пользователю возможность скачать созданное изображение, содержащее встроенные метаданные на основе C2PA.

Помимо функциональных требований, немаловажно определить нефункциональные требование, предназначенные для описания свойств системы, которыми она должна обладать при реализации своего поведения [22]. Для рассматриваемой системы возможно выделить следующие нефункциональные требования.

1. Система должна быть написана на языках программирования Python и JavaScript.

2. Система должна быть реализована в виде микросервисного приложения.

3. Система должна возвращать изображения в формате JPG.

На основе выделенных функциональных требований, была составлена диаграмма вариантов использования системы «Kandinsky Captions» (рисунок 8).

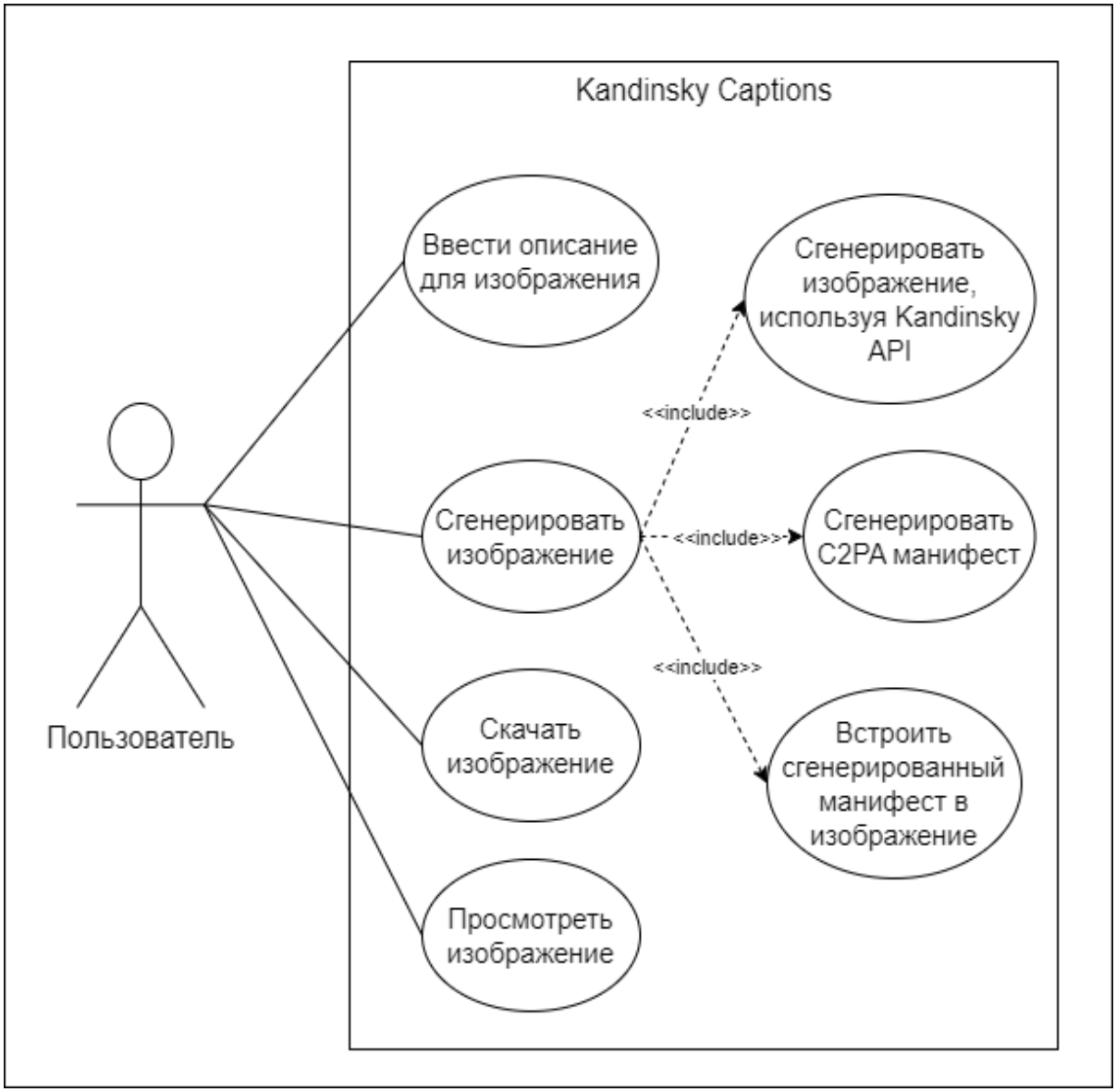

Рисунок 8 – Диаграмма вариантов использования системы «Kandinsky Captions»

В данной системе всего один актор «Пользователь», у него есть доступ ко всей функциональности системы.

Пользователь может ввести описание изображения, которое нужно сгенерировать.

Также пользователь имеет возможность сгенерировать изображение, после чего оно должно отобразиться в поле для просмотра изображения. В рамках данного варианта использования происходит отправка запроса на генерацию изображения, создание C2PA манифеста и его встраивание в изображение.

Помимо этого, пользователь может скачать сгенерированное изображение, которое содержит C2PA метаданные.

В рамках данной работы, помимо основной системы, разрабатывается независимый модуль – библиотека для использования стандарта C2PA, для которого также необходимо определить функциональные и нефункциональные требования.

В качестве функциональных требований для библиотеки C2PA можно выделить следующие пункты.

1. Библиотека должна предоставлять пользователю интерфейс для инициализации и завершения работы библиотеки.
2. Библиотека должна предоставлять пользователю интерфейс для создания assertion определенного типа, выбранного пользователем.
3. Библиотека должна предоставлять пользователю интерфейс для установки списка assertion.
4. Библиотека должна предоставлять пользователю интерфейс для создания манифеста на основе заданного списка assertion.
5. Библиотека должна предоставлять пользователю интерфейс для преобразования манифеста в формат, предназначенный для встраивания в данные.

Также можно определить следующие нефункциональные требования для библиотеки.

1. Библиотека должна быть написана на языке программирования Python.

2. Библиотека должна содержать документацию, описывающую функции интерфейса.

3. Библиотека должна быть доступна в качестве пакета Python.

На основе сформированных функциональных требований была разработана диаграмма вариантов использования для библиотеки «C2Py».

Диаграмма вариантов использования позволяет описать внешние и внутренние процессы системы. Данный вид диаграмм является хорошим способом описать варианты взаимодействия пользователя с системой [22]. Диаграмма вариантов использования для разрабатываемой в рамках данной работы библиотеки «C2Py» (рисунок 9) является в достаточной мере исчерпывающим примером использования.

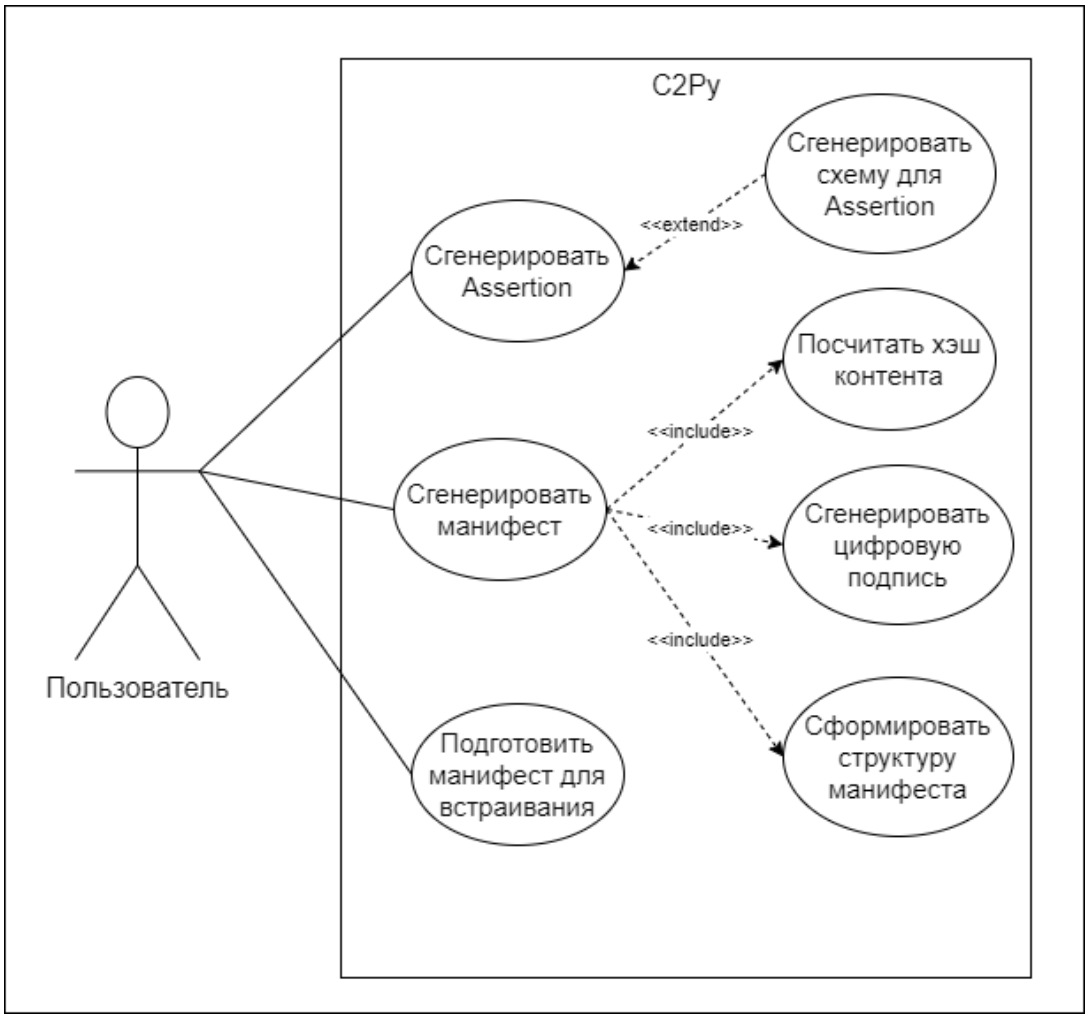

Рисунок 9 – Диаграмма вариантов использования библиотеки

Основным актором, который взаимодействует с системой, является пользователь библиотеки.

Пользователю доступна возможность сгенерировать необходимые Assertions, они описывают данные, которые пользователь желает отразить в манифесте. Для удобства установки Assertions имеется возможность сгенерировать схему Assertion, это позволит быстро и доступно заполнить схему Assertion необходимыми данными.

Также система предоставляет функциональность для генерации манифеста, которая включает в себя расчет хэша исходных данных, генерацию цифровой подписи на основе хэша, а также формирование структуры манифеста.

Помимо этого, пользователю предоставляется возможность подготовить манифест для встраивания в указанный формат данных.

3.2. Архитектура компонентов системы «Kandinsky Captions»

Диаграмма компонентов – это структурный вид диаграммы в языке моделирования UML (Unified Modeling Language), который используется для визуализации архитектуры системы или приложения. Она позволяет показать различные компоненты системы и связи между ними [22]. На рисунке 10 изображена диаграмма компонентов системы «Kandinsky Captions».

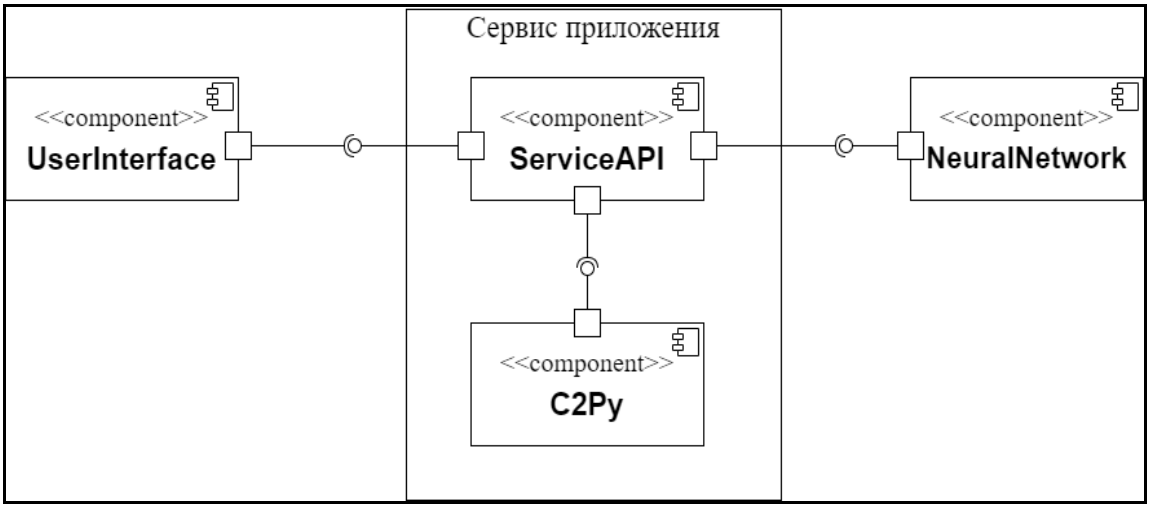

Рисунок 10 – Диаграмма компонентов приложения

Представленная диаграмма состоит из следующих элементов.

1. UserInterface – графический веб-интерфейс приложения. Данный компонент обеспечивает удобное и функциональное взаимодействия пользователя с системой.

2. NeuralNetwork – открытый API для генерации и получения изображения с помощью нейронной сети. Внешний компонент системы, обеспечивающий генерацию изображений с использованием генеративных нейронных сетей. Разработанная система обращается к данному сервису по средствам открытого интерфейса, который обеспечивает функционал для установки необходимых параметров для генерации изображений.

3. ServiceAPI – модуль, реализующий внутреннюю логику приложения. Является связующим звеном системы, обеспечивающий обработку запросов на генерацию изображения от пользователя. Данный компонент формирует запросы на генерацию изображения для компонента NeuralNetwork, и обеспечивает создание метаданных C2PA и встраивание их в сгенерированное изображение.

4. C2Py – разработанная в рамках данной работы библиотека, реализующая стандарт C2PA. Данный компонент предоставляет интерфейс для создания и встраивания метаданных C2PA.

3.3. Проектирование графического интерфейса

Проектирование графического интерфейса является важной частью создания пользовательских приложений с графическим интерфейсом. Оно включает в себя создание удобного и интуитивно понятного пользовательского опыта. Графический интерфейс должен соответствовать обозначенным функциональным требованиям системы, которые выявляются на фазе анализа системы [23]. Данные требования служат основной информацией для создания концепции интерфейса и направлены на разработку простого и удобного в использовании графического интерфейса. На рисунке 11 изображен концепт графического интерфейса разрабатываемой системы.

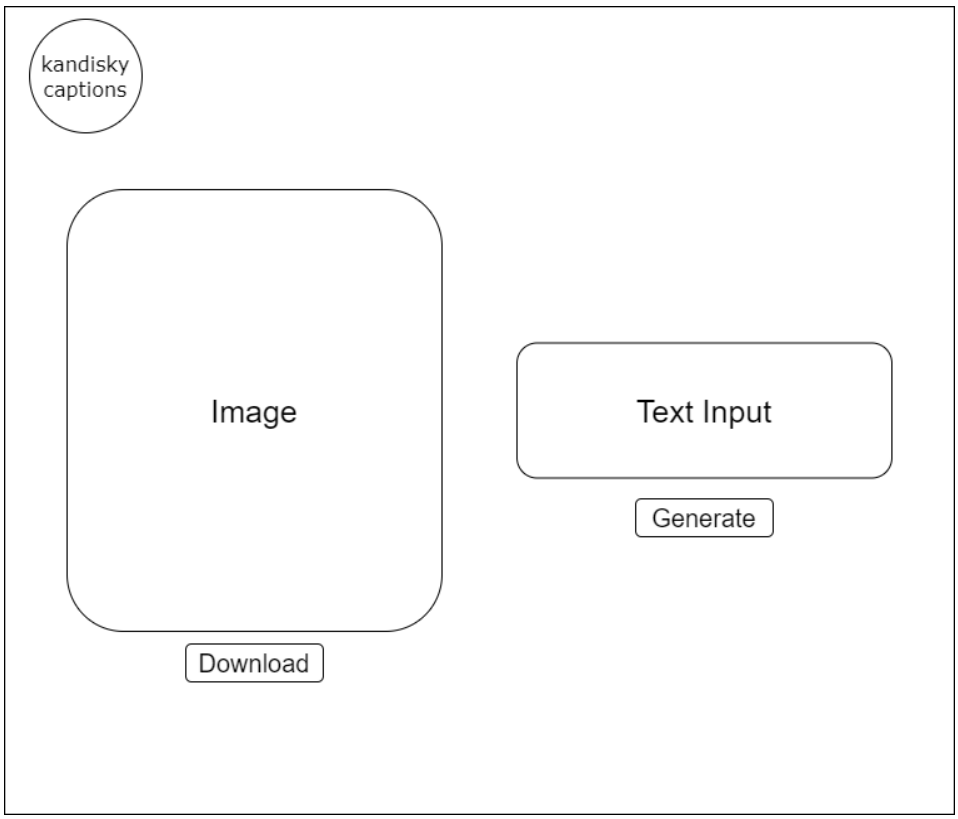

Рисунок 11 – Концепт графического интерфейса разрабатываемой системы

Представленный концепт графического интерфейса разделен на две смысловые части. В левой части содержится окно для просмотра изображения, в котором предполагается вывод сгенерированного изображения. Под окном для просмотра изображения расположена кнопка для скачивания изображения. В правой части находится текстовое поле для ввода текста, на основе которого будет сгенерировано изображение, под текстовым полем изображена кнопка, предназначенная для отправки запроса на генерацию изображения.

Выводы по третьей главе

В данной главе был проведен анализ функциональных и нефункциональных требований разрабатываемой библиотеки «C2Py» и системы «Kandinsky Captions». На основе выделенных требований были построены диаграммы вариантов использования. Помимо этого, была разработана архитектура компонентов системы и создан концепт графического интерфейса системы.

4. РЕАЛИЗАЦИЯ

Обзор используемых средств реализации

Работа с кодовой базой системы производилась в среде разработки Visual Studio Code с использованием Docker контейнеров для разработки. Docker контейнеры для разработки – это удобный инструмент виртуализации окружения для разработки, позволяющий устанавливать необходимые зависимости в Docker контейнер. Это позволят эффективно использовать пространство локальной машины, не устанавливая на нее большое количество пакетов и средств разработки. Помимо этого, Docker контейнеры для разработки дают возможность быстро устанавливать среду разработки на другой машине без необходимости вручную устанавливать все необходимы средства для разработки [24].

Разработка библиотеки, реализующая стандарт C2PA проводилась на языке программирования Python 3.8. Исходный код серверной части системы был также написан с использованием языка программирования Python 3.8 и фреймворка Flask 2.0.3. Клиентская часть приложения реализована на языке JavaScript и программной платформы Node.js 20.12.0. Контейнеризация системы проводилась с помощью Docker.

4.1. Разработка библиотеки на основе стандарта C2PA

Стандарт C2PA имеет сложную вложенную структуру, реализация которой хорошо ложится на принципы объектно-ориентрованного программирования. Все базовые типы стандарта были реализованы с помощью классов, используя принципы ООП.

JUMBF

Базовым классом в реализации JUMBF структур является класс Box. Данный класс содержит в себе базовую логику всех JUMBF объектов, например, подсчет длины объекта, определение типа объекта и сериализация.

Следующими по значимости являются классы Description Box и Content Box. Данные объекты нужны для описания содержимого контента и непосредственно хранения контента. Description Box наследует функциональность базового класса Box и дополнительно имеет поля для более подробного описания содержимого, такие как тип контента и метка. Content Box также наследуется от класса Box и служит в качестве контейнера для содержимого, которое записывается в поле payload.

Super Box является главным объектом JUMBF структуры, который объединяет в себе все описанные выше классы. Данный класс содержит в себе Description Box, необходимый для описания содержимого, и список объектов содержимого, который может включать в себя объекты типа Content Box и Super Box.

На рисунке 12 показана схема, описывающая взаимодействия между структурами JUMBF.

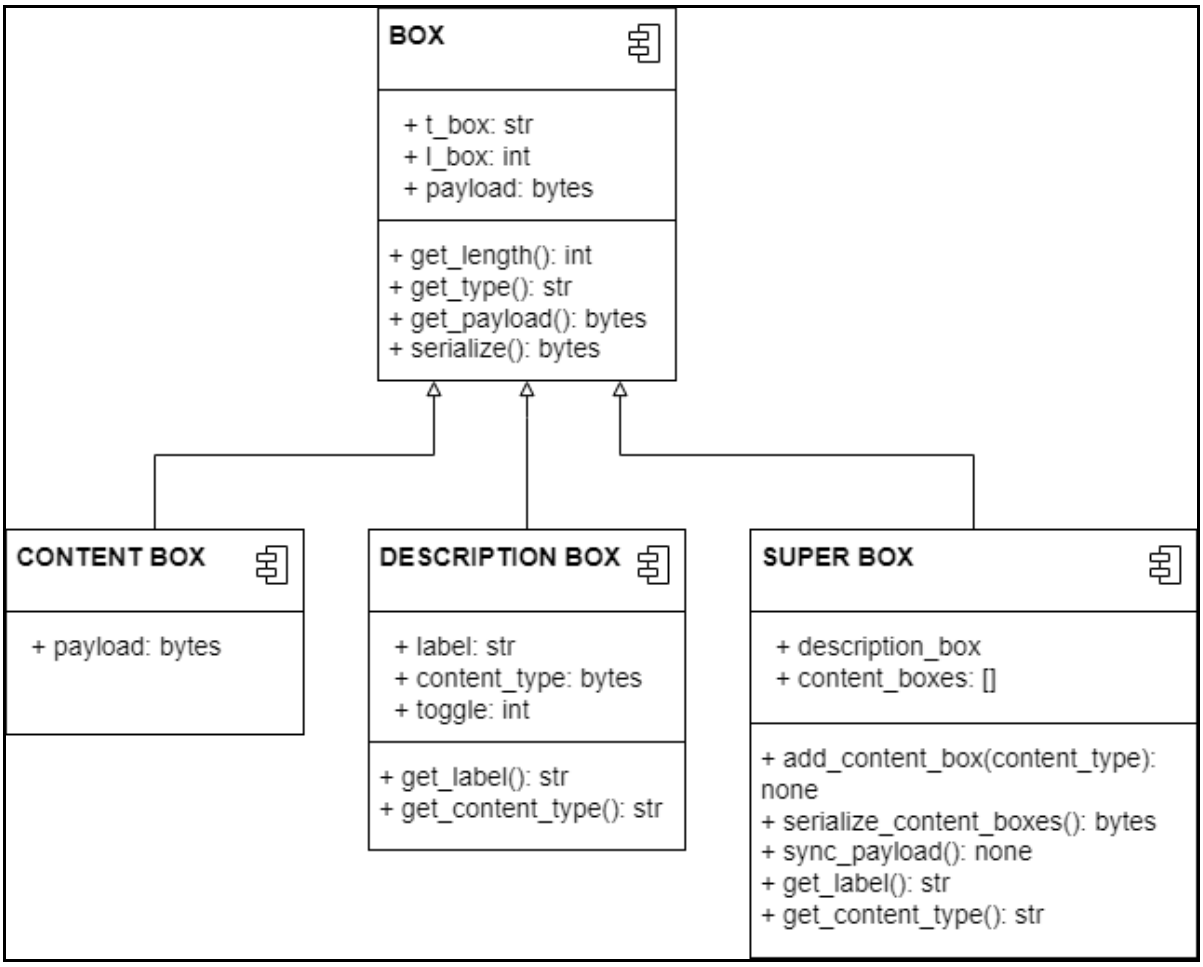

Рисунок 12 – Взаимодействия JUMBF структур

C2PA объекты

Стандарт C2PA описывает объекты внутренней структуры, которые по свойствам схожи с объектами JUMBF. Все базовые сущности, описываемые в стандарте, наследуются от Super Box класса, описанного выше.

Весь контент, который может быть записан в C2PA метаданные, описывается с помощью обозначенных в стандарте схемах или сырых данных, например, assertion для отображения миниатюры изображения. Данные схемы описываются с помощью JSON и CBOR форматов. Конструктор класса Assertion принимает на вход тип Assertion и схему, описывающую входные данные. Исходный код реализации класса Assertion представлен в листинге 1 приложения А.

JSON (JavaScript Object Notation) – это формат обмена данными, который используется для передачи структурированной информации. Он представляет собой текстовый формат, состоящий из пар ключ – значение [25].

CBOR (Concise Binary Object Representation) – формат компактного двоичного представления данных, который похож на JSON. Данный формат использует двоичное представление, что делает его более компактным и эффективным для передачи, чем текстовый формат JSON [26].

Существующие assertions сохраняются в качестве списка в объект под названием Assertion Store, который так же наследуется от Super Box. Для инициализации данного объекта необходимо передать на вход конструктору список assertions.

Объявленные assertions объекты несут важную информацию о метаданных контента, поэтому их важно безопасно хранить в метаданных, для этого используется объект Claim. При инициализации данный объект принимает на вход список assertions (assertion store) и формирует схему формата CBOR, которая служит полезной нагрузкой данного объекта. Схема формата CBOR содержит следующие поля, отображающие основную информацию, содержащуюся в объекте.

1. Alg – строка, содержащая название алгоритма хэширования, который будет использоваться для хэширования assertions.

2. Claim generator – строка, содержащая информацию о лице, который формирует Claim.

3. Signature – строка, содержащая путь до Claim Signature объекта в рамках JUMBF системы.

4. Assertions – список словарей, содержащих информацию о каждом assertion, а именно ссылку на assertion в рамках JUMBF системы и хэш значения, посчитанное на основе полезной нагрузки объекта assertion. Исходный код реализации класса Claim представлен в листинге 2 приложения А.

Самый важный объект всей структуры C2PA – Claim Signature. Он необходим для хранения информации о цифровой подписи и саму цифровую подпись. Данный объект так же, как и Claim генерируют схему, содержащую приватный ключ, цепочку сертификатов и цифровую подпись. Для генерации цифровой подписи необходимы приватный ключ и исходные данные, поэтому при инициализации объект Claim принимает на вход приватный ключ, цепочку сертификатов (нужна для валидации цифровой подписи) и Assertion Store. Исходный код реализации класса ClaimSignature представлен в листинге 3 приложения А.

Manifest – объект, который объединяет в себя Assertion Store, Claim и Claim Signature. При этом Manifest объект может быть не один, поэтому чтобы сохранять несколько Manifest, нужен объект Manifest Store, который содержит в себе список объектов Manifest.

На рисунке 13 представлена структурная схема объектов стандарта C2PA.

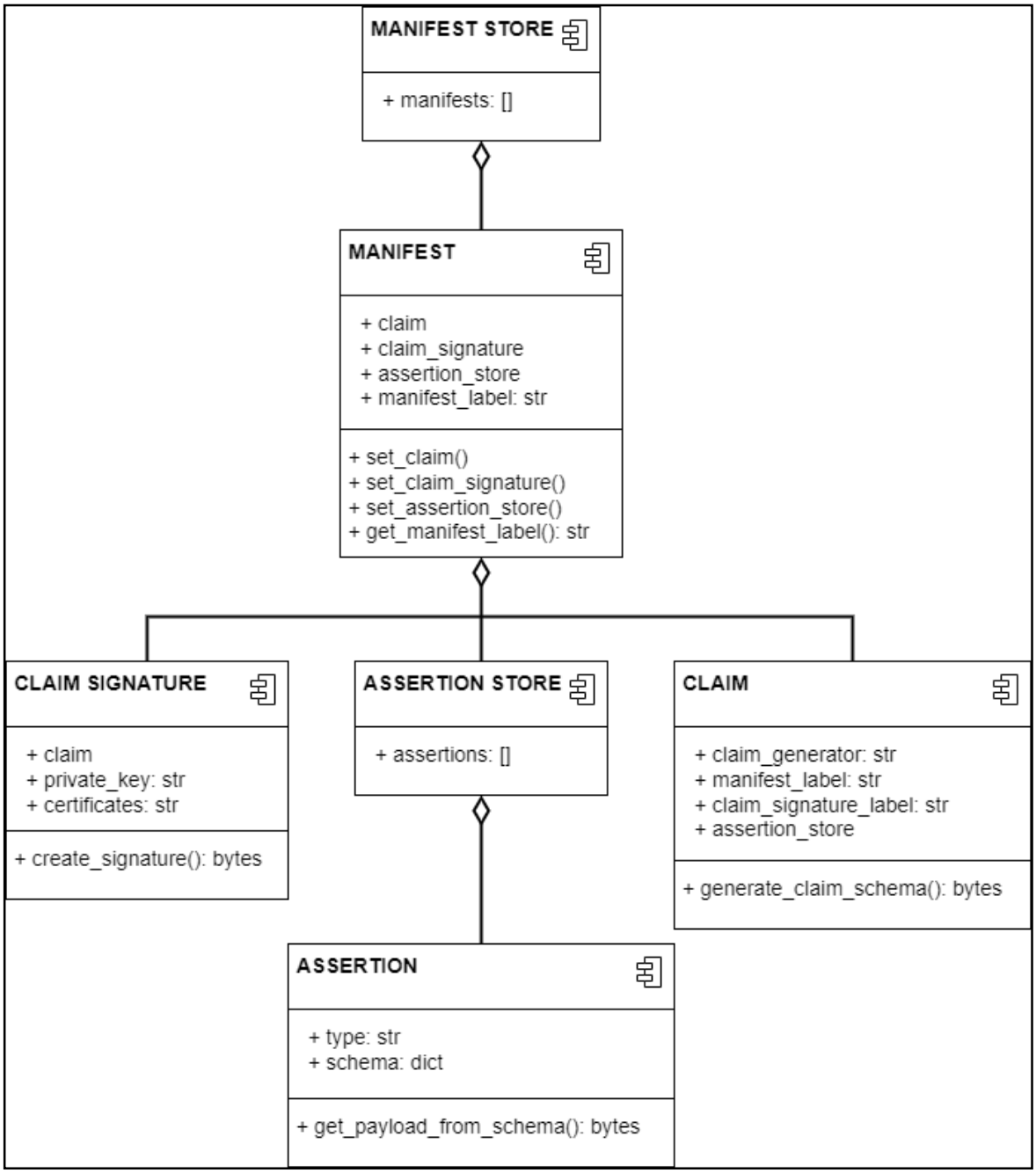

Рисунок 13 – Объекты C2PA

Встраивание C2PA манифеста в исходные данные

В рамках первой версии библиотеки рассматривается возможность встраивание C2PA манифеста только в изображения формата JPG.

JPG изображение состоит из сегментов, максимальная длина которых составляет 65 535 байт. Каждый сегмент имеет заголовок, описывающий

длину и содержимое сегмента. В первую очередь идут маркеры, обозначающие начало сегмента (FF) и его тип. Для хранения C2PA манифеста используется APP11 сегмент, который имеет значение EB.

После маркеров, обозначающих начало сегмента идет поле, содержащее значение его длины, включая начальные маркеры, данное поле обозначается как LE.

После следует тип контента (CI), в случае с JPG форматом это два байта, содержащие значение JP.

Сегментов одного типа может быть несколько, так как длина содержимого может быть больше, чем обозначенная максимальная длина сегмента. В данном случае, необходимо создать новый сегмент, в котором будет содержаться остальная часть содержимого. К тому же изображение может содержать несколько сегментов одного типа, не связанных друг с другом. Для регулирования подобных ситуаций нужны поля заголовка сегмента, называемые EN (индекс сегмента) и Z (номер сегмента в последовательности).

Далее следуют два поля t_box и l_box, описываемых JUMBF форматом, они содержат значения длины и типа содержимого сегмента.

Заключающей частью сегмента является содержимое сегмента. В случае с C2PA манифестом, содержимым является сериализованный манифест в формате JUMBF.

Исходный код реализации встраивания в JPG формат данных представлен в листинге 4 приложения А.

Интерфейс библиотеки «C2Py»

Разработка интерфейса для библиотеки является одной из важнейших частей разработки библиотеки. Интерфейс библиотеки должен позволять пользователю удобно и эффективно взаимодействовать с программным модулем, используя весь доступный функционал модуля. Помимо этого, интерфейс библиотеки должен быть спроектирован таким образом, чтобы

дальнейшее расширение функциональности модуля можно было включить в интерфейс, не ломая функциональность предыдущих версий модуля.

Следующие функции были выделены в качестве основных функций интерфейса для разрабатываемого модуля.

1. `C2Py_GenerateAssertion` – функция, позволяющая задать пользовательский assertion.

2. `C2Py_GenerateManifest` – функция для генерации C2PA манифеста.

3. `C2Py_EmplaceManifest` – функция для встраивания манифеста в исходный тип данных.

Функция `C2Py_GenerateAssertion` принимает на вход тип assertion и схему, описывающую исходные данные assertion. Данная схема может быть представлена в виде Python словаря или необработанных данных, тип схемы зависит от типа assertion. Результатом работы функции является объект Assertion.

Для корректной работы функции `C2Py_GenerateManifest` требуется передать список объектов типа Assertion, а также приватный ключ и цепочку сертификатов, необходимых для создания цифровой подписи. Функция возвращает сгенерированный объект типа ManifestStore.

Функция `C2Py_EmplaceManifest` ожидает получить на вход тип исходного формата данных и исходные данные в бинарном представлении. Помимо этого, необходимо указать количество байт, которое нужно отступить от начала исходных данных, для вставки C2PA манифеста. Также данная функция ожидает получить объект типа Manifest. Результатом работы функции является массив байт, который содержит исходные данные со встроенным C2PA манифестом.

Порядок вызова описанных функций интерфейса библиотеки обозначен в диаграмме последовательности, представленной на рисунке 14.

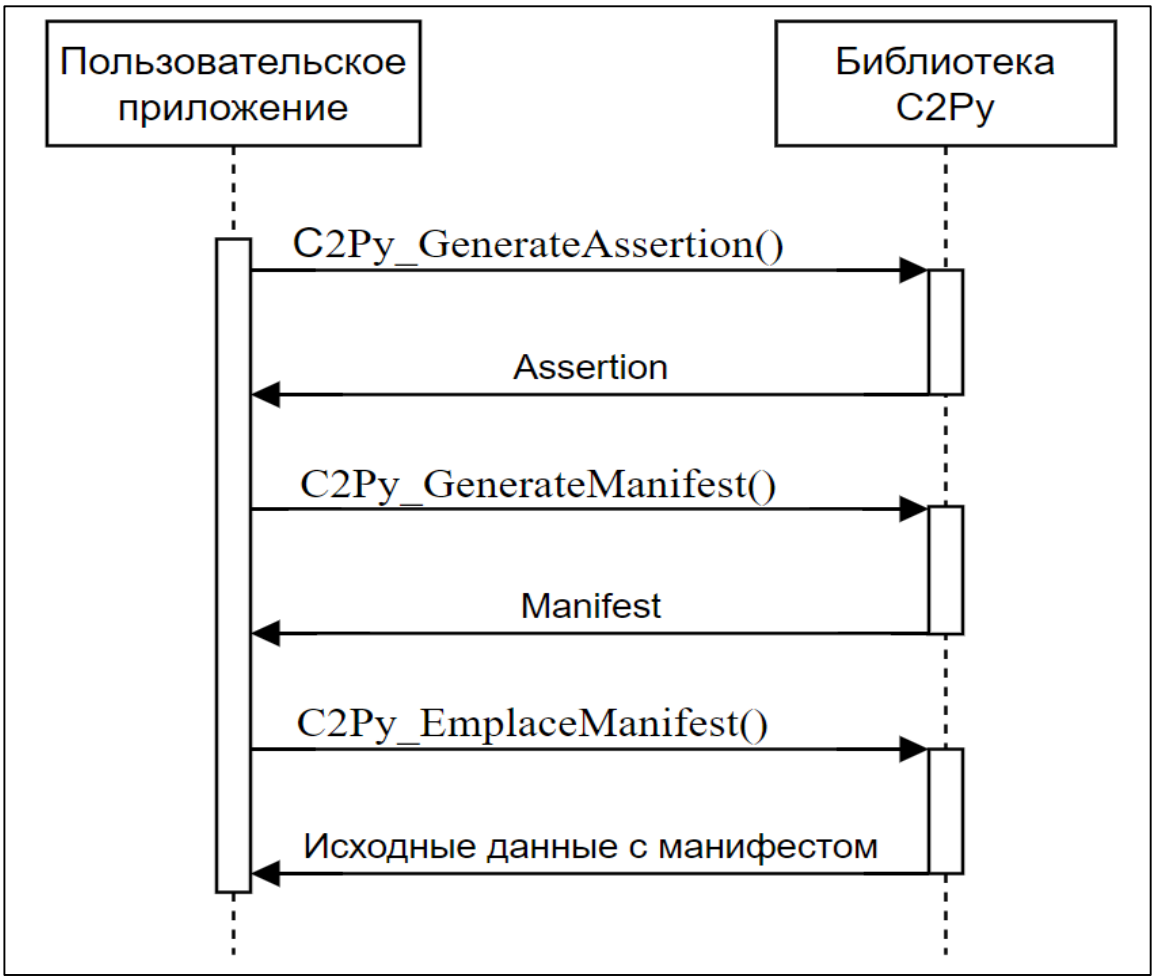

Рисунок 14 – Порядок взаимодействия с интерфейсом библиотеки C2Py

Исходный код реализации интерфейса библиотеки «C2Py» представлен в листинге 5 приложения А.

Тестовое приложение

В рамках разработки библиотеки было реализовано тестовое приложение, которое служит примером использования и выступает в качестве интеграционного теста.

В данном приложении в качестве исходных данных используется изображение в формате JPG. С помощью функций интерфейса разработанной библиотеки составляется список объектов assertion, который включает в себя информацию об авторе, миниатюру изображения, хэш значение исходного изображения. С использованием составленной информации создается C2PA манифест, который встраивается в исходное изображение.

В результате реализации тестового приложения был получен наглядный и функциональный пример использования разработанной библиотеки.

4.2. Разработка серверной части приложения

Серверная часть приложения представляет собой микросервис, который принимает запросы от клиентской части приложения и выполняет основную логику работы приложения, а именно генерацию изображения и встраивание C2PA манифеста в него. Данный микросервис был реализован на языке Python с использованием фреймворка Flask.

Endpoints

Интерфейс для взаимодействия с серверной частью приложения состоит из двух ключевых точек для обработки запросов от клиента, такие точки называются endpoint [27].

В разрабатываемой приложении необходим только один endpoint типа POST, который принимает текст, описывающий изображение, которое пользователь хочет сгенерировать, формирует запрос для Kandinsky API и отправляет сформированный запрос. После получения изображения в бинарном представлении от Kandinsky API, генерируется C2PA манифест с использованием разработанной библиотеки «C2Py». Если манифест сгенерировался без ошибок, он вставляется в исходное изображение. Готовое изображение со встроенным манифестом отправляется на клиентскую часть приложения.

API Kandinsky

Для обращения к Kandinsky API необходимо сформировать запрос, который содержит информацию о желаемом изображении (размер, описание, количество изображений), а также данные авторизации для доступа к API.

Для удобного обращения к Kandinsky API был разработан модуль, позволяющий задать всю необходимую информацию для получения сгенерированного изображения. Данный модуль разработан по принципам ООП

и представляет собой класс, для инициализации которого необходимо передать данные авторизации в API. После чего необходимо сформировать запрос для генерации изображения, это реализована с помощью метода класса `get_model`. После того, как запрос сгенерирован, необходимо указать параметры данного запроса, а именно желаемые размеры изображения и его описание, для этого реализована функция `generate`. Поскольку генерация изображения может занять некоторое количество времени, необходимо реализовать механизм проверки готовности результата, это выполнено используя подход активного ожидания. Активное ожидание представляет собой итерирование по циклу, внутри которого отправляются запросы на API для получения статуса генерации, после положительного ответа, ожидание прекращается. Описанный алгоритм реализован в методе `check_generation`.

Исходный код реализации модуля для обращения к Kandinsky API представлен в листинге 6 приложения Б.

C2Py интеграция

Разработанная библиотека «C2Py» интегрирована в серверную часть приложения в качестве модуля с открытым интерфейсом, функции которого используются для создания C2PA манифеста.

После того, как сгенерированное изображение получено от Kandinsky API, с помощью интерфейсной функции библиотеки составляется список Assertion объектов, а именно производится хэширование исходного изображения и составляется информация об авторе. После этого генерируется манифест и вставляется в исходное изображение.

4.3. Разработка клиентской части приложения

Клиентская часть приложения представляет собой веб-страницу, которая служит для удобного взаимодействия пользователя с системой. На разработанной странице находятся элементы, представленные ниже.

1. Элемент для отображения сгенерированного изображения.
2. Текстовое поле для ввода описания изображения.

3. Кнопка для отправки запроса на генерацию изображения.
4. Кнопка для скачивания сгенерированного изображения.

Значение текстового поля обновляется и записывается в переменную, которая предназначена для формирования запроса в API. Кнопка для отправки запроса на генерацию изображения составляет запрос в JSON формате, в котором содержится значение переменной с текстовым описанием, и отправляет его в необходимый endpoint разработанного API. В качестве ответа приходит изображение в бинарном представление, которое передается в элемент для отображения изображения.

Кнопка для скачивания должна срабатывать только тогда, когда пользователь ввел запрос на генерацию изображения и получил результат. Данная кнопка по своей сути является ссылкой на данные, которые были получены в результате генерации изображения, поэтому на серверной части не нужен отдельный endpoint на скачивание изображения.

Исходный код реализации компонентов клиентской части системы «Kandinsky Captions» представлен в листинге 7 приложении Б.

На рисунках 15-16 показан графический интерфейс приложения «Kandinsky Captions».

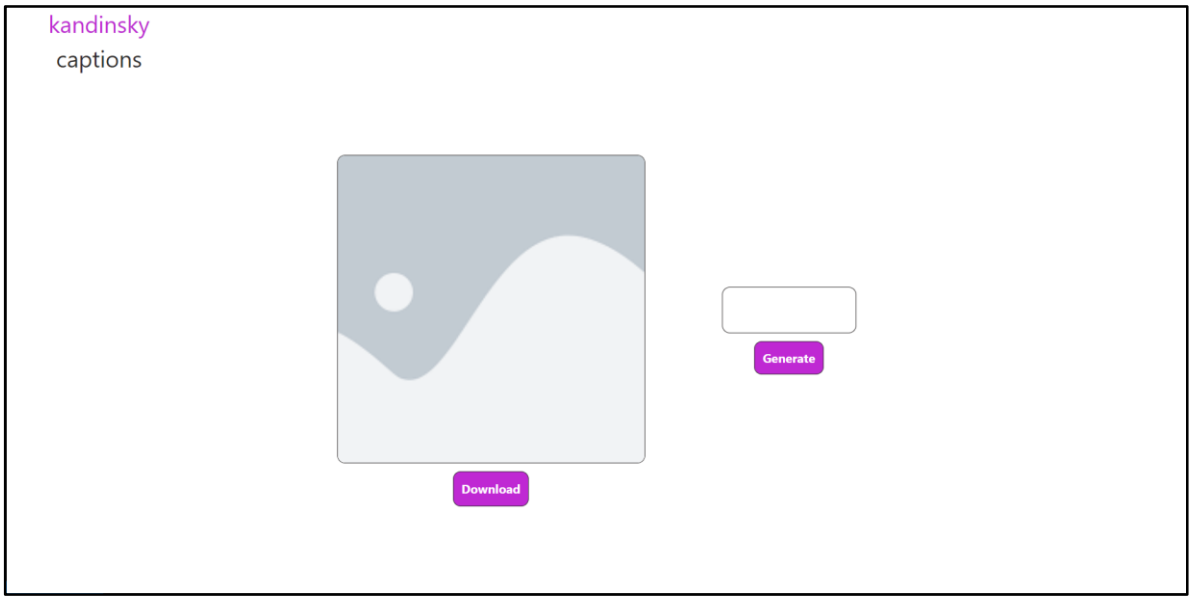

Рисунок 15 – Стартовая страница приложения

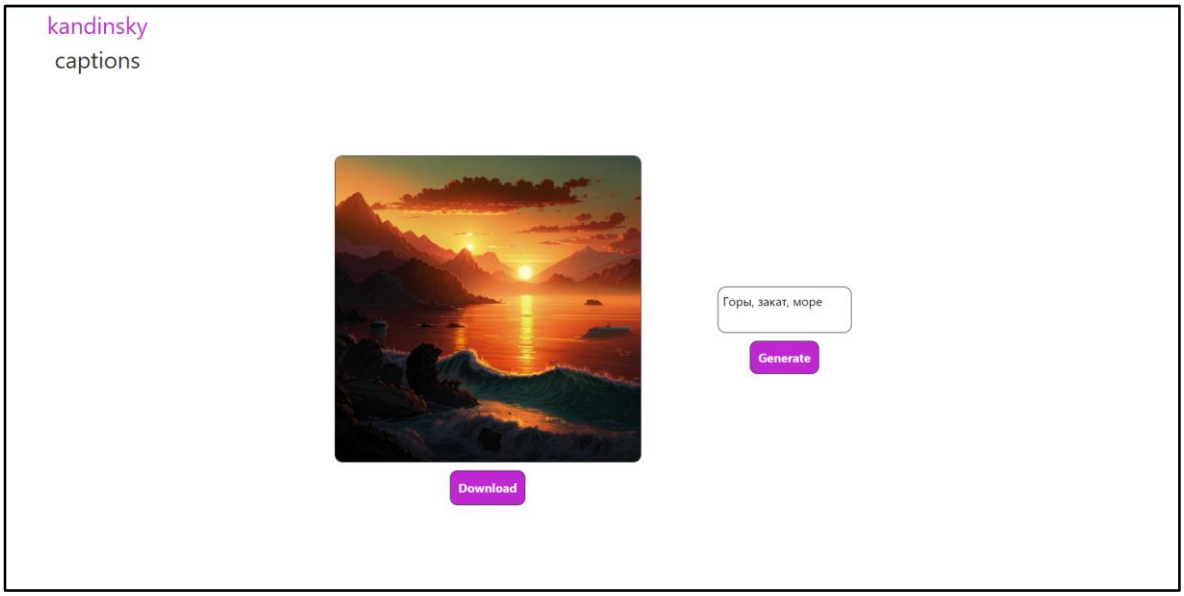

Рисунок 16 – Страница приложения с генерированным изображением

Выводы по четвертой главе

В рамках данной главы была рассмотрена реализация основных компонентов системы «Kandinsky Captions», включая разрабатываемую библиотеку «C2Py». Были описаны ключевые объекты и подходы, используемые при разработке библиотеки, помимо этого был описан реализованный интерфейс для взаимодействия пользователя с библиотекой. Также был описан принцип создания тестового приложения, которое использует ключевые особенности библиотеки. Помимо этого, было представлено описание реализации серверной части приложения, включая описание алгоритмов, используемых для обращения в публичный API сервиса Kandinsky. Также в рамках данной главы описана реализация клиентской части приложения, включая описание всех основных компонентов.

5. ТЕСТИРОВАНИЕ

5.1. Юнит тестирование

Разработка библиотеки «C2Py» проводилась с использованием методологии TDD (Test Driven Development). Данный подход подразумевает первостепенное написание небольших тестов, а после этого код, позволяющий проходить тесты. TDD подразумевает три этапа.

1. Написание теста, который проверяет требуемое поведение программы. Тест не должен пройти, так как реализации соответствующей функциональности еще нет.
2. Написание кода, реализующего минимальную функциональность, необходимую для прохождения теста.
3. Улучшение написанного кода без изменения поведения таким образом, чтобы тест все равно проходил.

Данная методология позволяет создавать более надежное и чистое программное обеспечение, так как каждая функция или изменение обеспечивается тестами, которые могут быстро сигнализировать об ошибке. В рамках TDD код покрывается юнит тестами – тестами, которые проверяют небольшую часть функциональности [28].

В рамках разработки библиотеки было написано 40 юнит тестов, покрывающих основную функциональность. Данные тесты постепенно дополнялись и проверялись с каждым изменением кода. Помимо этого, проверка данных тестов выполнялась автоматически при каждой загрузке кода на платформу GitHub, с помощью среды GitHub Actions (рисунок 17). Для этого были реализованы скрипты для автоматического запуска юнит тестов в среде GitHub Actions.

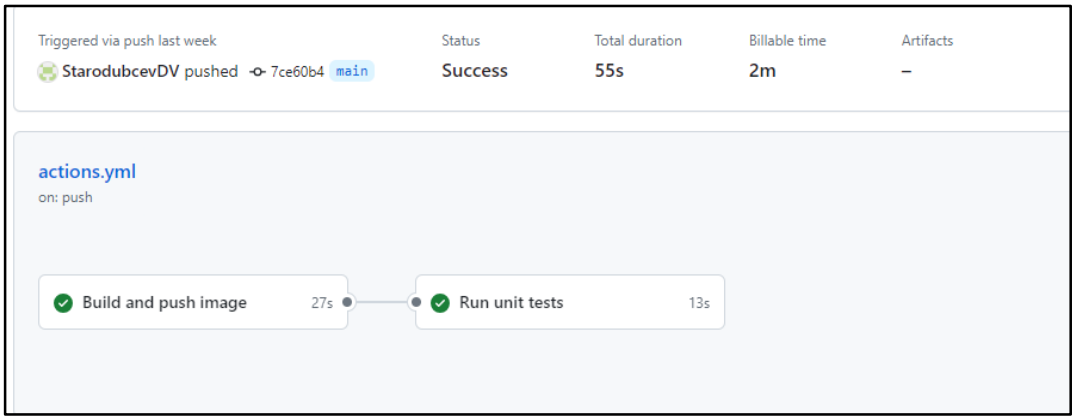

Рисунок 17 – Автоматический запуск юнит тестов.

5.2. Интеграционное тестирование

Интеграционное тестирование – это процесс проверки взаимодействия между различными компонентами или модулями системы для обеспечения их корректной работы в совокупности. Оно направлено на выявление ошибок в процессе интеграции компонентов и их взаимодействия друг с другом [28].

В рамках интеграционного тестирования с использованием тестового приложения, разработанного в рамках реализации библиотеки «C2Py», полученное изображение с C2PA манифестом загружается на сайт верификации C2PA [29]. Благоприятным исходом тестирования является корректное отображение информации на сайте верификации (рисунок 18).

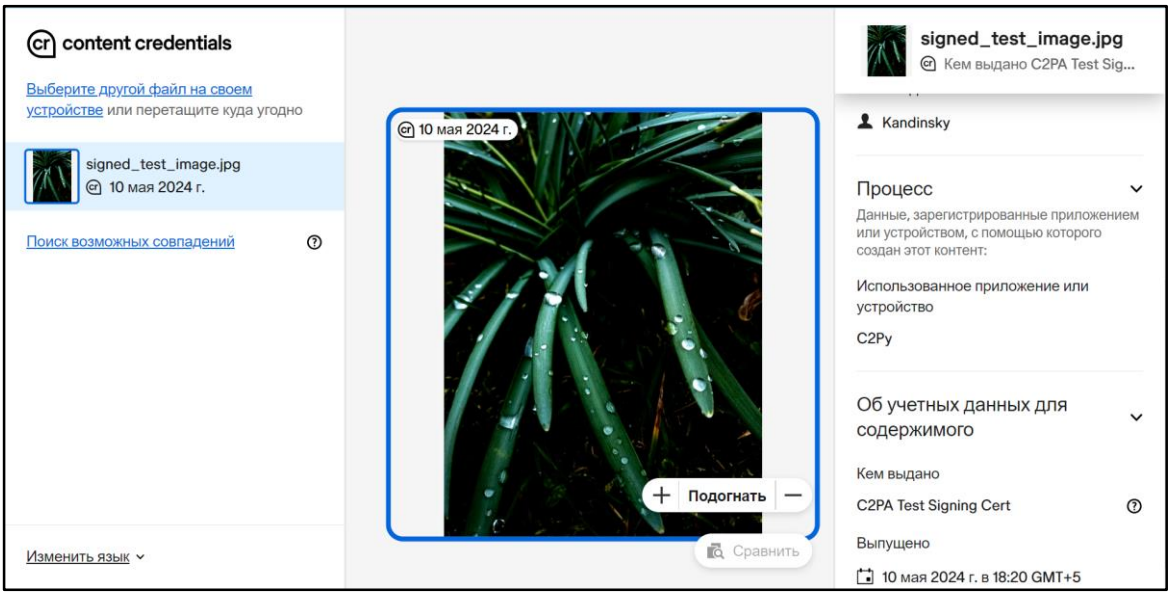

Рисунок 18 – Валидный результат интеграционного тестирования

В ходе интеграционного тестирования выявились проблемы при интеграции манифеста в исходное изображение, после чего это было поправлено. Так произошло, потому что при написании юнит тестов были не предусмотрены некоторые краевые случаи интеграции. После обнаружения ошибки данная функциональность была дополнительно покрыта юнит тестами.

5.3. Ручное тестирование

Ручное тестирование – это процесс проверки программного обеспечения с использованием людей (тестировщиков), которые вручную выполняют тестовые случаи без автоматизации. В ходе такого тестирования тестировщики выполняют различные действия, такие как ввод данных, проверка вывода, интеракцию с пользовательским интерфейсом и так далее, чтобы убедиться в правильном функционировании программы [30].

В рамках ручного тестирования была проверена функциональность клиентской части приложения, а именно корректность отправки запросов и отображение результата, скачивание изображения и проверка его на сайте верификации C2PA. Были выделены следующие случаи для тестирования.

1. Ввод запроса на русском языке без использования цифр и специальных символов. Нажатие на кнопку «Generate» Ожидаемое поведение: в поле для отображения изображения появилось изображение, соответствующее запросу.
2. Ввод запроса на английском языке без использования цифр и специальных символов. Нажатие на кнопку «Generate» Ожидаемое поведение: в поле для отображения изображения появилось изображение, соответствующее запросу.
3. Ввод запроса на русском языке с использованием цифр и специальных символов. Нажатие на кнопку «Generate» Ожидаемое поведение: в поле для отображения изображения появилось изображение, соответствующее запросу.

4. Ввод и отправка запроса на генерацию. Изображение сгенерировалось и отобразилось. Нажатие на кнопку «Download». Ожидаемое поведение: скачивание сгенерированного изображения прошло успешно.

5. Повторное нажатие на кнопку «Download». Ожидаемое поведение: скачивание сгенерированного при прошлом запросе изображения прошло успешно.

6. Ввод и отправка запроса на генерацию. Изображение сгенерировалось и отобразилось. Нажатие на кнопку «Download». Ожидаемое поведение: скачивание сгенерированного изображения прошло успешно.

7. Обновление страницы. Нажатие на кнопку «Download». Ожидаемое поведение: скачивание изображения не произошло.

8. Ввод и отправка запроса на генерацию. Изображение сгенерировалось и отобразилось. Нажатие на кнопку «Download». Загрузка скачанного изображения на сайт верификации C2PA. Ожидаемое поведение: сайт верификации выдаст валидную информацию об изображении.

Все описанные тестовые случаи прошли успешно.

Выводы по пятой главе

В данной главе были описаны подходы к тестированию разработанной библиотеки «C2Py» и системы «Kandinsky Captions». В рамках тестирования использовались несколько распространенных методик, таких как юнит тестирование, интеграционное тестирование и ручное тестирование. В ходе тестирования были обнаружены ошибки в работе библиотеки «C2Py», что позволило исправить недочеты в работе системе на этапе разработки.

ЗАКЛЮЧЕНИЕ

В рамках данной работы было разработано приложение для генерации валидированных изображений на основе стандарта C2PA с использованием генеративных нейронных сетей.

1. Произведен обзор литературы и существующих приложений по предметной области.
2. Выполнено проектирование архитектуры библиотеки для стандарта C2PA.
3. Разработана библиотека на языке программирования Python, реализующая функциональность стандарта C2PA.
4. Выбрана модель генеративной нейронной сети.
5. Выполнено проектирование архитектуры приложения.
6. Разработано приложения для генерации изображений с использование генеративных нейронных сетей.
7. Проведено тестирование приложения.

В будущем планируется продолжать разработку и улучшение библиотеки «C2Py», в частности расширение возможности встраивания C2PA манифеста в различные форматы данных, доработка автоматизации создания релизов и выпуск пакетов. Также после доведения библиотеки до определенного уровня качества, планируется открытие исходного кода в публичный доступ.

ЛИТЕРАТУРА

1. C2PA. [Электронный ресурс] URL: https://c2pa.org/ (дата обращения: 01.02.2024 г.).

2. Riley J. Understanding Metadata. What is metadata, and what is for? // National Information Standards Organization (NISO), 2017. – 49p.

3. Ahmad N. The Impact of Social Media Content Marketing (SMCM) towards Brand Health. // The Impact of Social Media Content Marketing (SMCM) towards Brand Health, 2015. – 6 p.

4. Tiwari H. A secure and efficient cryptographic hash function based on NewFORK-256. // Department of Computer Science and Engineering, JIIT (Deemed University), 2012. – 10 p.

5. Aas J. Let's Encrypt: An Automated Certificate Authority to Encrypt the Entire Web. // CCS, 2019. – 15 p.

6. Katz J. Complete characterization of security notions for probabilistic private-key encryption. // STOC: Proceedings of the thirty-second annual ACM symposium on Theory of computing, 2000. – 10 p.

7. Kaur R. Digital Signature. // International Conference on Computing Sciences, 2012. – 20 p.

8. Danezis G., Cristofaro E. LOGAN: Evaluating Privacy Leakage of Generative Models Using Generative Adversarial Networks. // University College London, 2017. – 18 p.

9. German artist refuses award after his AI image wins prestigious photography prize. [Электронный ресурс] URL: https://www.engadget.com/german-artist-refuses-award-after-his-ai-image-wins-prestigious-photography-prize-071322551.html (дата обращения: 03.02.2024 г.).

10. Adobe Firefly. [Электронный ресурс] URL: https://www.adobe.com/products/firefly.html (дата обращения: 10.02.2024 г.).

11. Image Creator. [Электронный ресурс] URL: https://designer.microsoft.com/image-creator (дата обращения: 11.02.2024 г.).

12. Kandinsky. [Электронный ресурс] URL: https://www.sber-bank.com/promo/kandinsky (дата обращения: 13.02.2024 г.).

13. DALL-E. [Электронный ресурс] URL: https://openai.com/research/dall-e (дата обращения: 15.02.2024 г.).

14. Hanna D. The Use of Artificial Intelligence Art Generator «Midjourney» in Artistic and Advertising Creativity. // Journal of Design Sciences and Applied Arts, 2023. – 42 p.

15. Razzhigaev A. Kandinsky: An Improved Text-to-Image Synthesis with Image Prior and Latent Diffusion. // Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, 2023. – 10 p.

16. England P. AMP: Authentication of Media via Provenance. // 12th ACM Multimedia Systems Conference, 2021. – 14 p.

17. Grasso A. Restoring Trust in Online Content and Demystifying Deepfakes with C2PA Credentials. // Digital Business Innovation, 2023. – 7 p.

18. Черемушкин А. В. О содержании понятия «Электронная подписи». // Институт криптографии, связи и информации, 2012. – 17 с.

19. Sonain J. A novel multimedia player for International Standard – JPEG snack. // Journal of imaging, 2023. – 17 p.

20. Vasmani A. Attention is all you need. // 31st Conference on Neural Information Systems, 2017. – 15 p.

21. Куликов С.С. Тестирование программного обеспечения. Базовый курс. 2 издание. // Издательство Четыре Четверти, 2017. – 312 с.

22. Арлоу Д., Нейштадт А. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, 2-е издание. // СПб: Символ-Плюс, 2007. – 624 с.

23. Krug S. Don`t make me think. // New Riders, 2014. – 212 p.

24. Developing inside a Container. [Электронный ресурс] URL: https://code.visualstudio.com/docs/devcontainers/containers (дата обращения: 01.04.2024 г.).

25. JSON – JavaScript. [Электронный ресурс] URL: https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/JSON (дата обращения: 06.04.2024 г.).

26. CBOR – Concise Binary Object Representation. Overview. [Электронный ресурс] URL: https://cbor.io (дата обращения: 06.04.2024 г.).

27. Messe M. REST API. // O`Reilly Media Inc, 2011. – 93 p.

28. Percival H. Test Driven Development with Python. // O`Reilly Media Inc, 2017. – 614 p.

29. Content Credentials. [Электронный ресурс] URL: https://contentcredentials.org/verify (дата обращения: 10.04.2024 г.).

30. Соммервилл И. Инженерия программного обеспечения. // Издательский дом «Вильямс», 2002. – 625 с.

ПРИЛОЖЕНИЯ

Приложение А. Исходный код библиотеки «C2Py»

Исходный код библиотеки «C2Py» представлен в листингах 1–5.

Листинг 1 – Реализация класса `Assertion`

```
class Assertion(SuperBox):
    def __init__(self, assertion_type, schema):
        self.type = assertion_type
        self.schema = schema
        self.payload = self.get_payload_from_schema()
        content_box = ContentBox(
                          box_type=
                          get_assertion_content_box_type(assertion_type),
                          payload=self.payload
                          )
        super().__init__(
                    content_type=get_assertion_content_type(assertion_type),
                          label=get_assertion_label(assertion_type),
                          content_boxes=[content_box])

    def get_payload_from_schema(self):
        if get_assertion_content_type(self.type) ==
jumbf_content_types['json']:
            return json_to_bytes(self.schema)
        elif get_assertion_content_type(self.type) ==
jumbf_content_types['cbor']:
            return cbor_to_bytes(self.schema)
        elif get_assertion_content_type(self.type) ==
jumbf_content_types['codestream']:
            return self.schema['payload']
        else:
            return b''

    def get_data_for_signing(self):
        return self.description_box.serialize()+
                        self.serialize_content_boxes()
```

Листинг 2 – Реализация класса `Claim`

```
class Claim(SuperBox):

    def __init__(self, assertion_store, claim_generator="",
manifest_label="", image_format="image/jpeg"):
        self.claim_generator = claim_generator
        self.manifest_label = manifest_label
        self.claim_signature_label =
f"self#jumbf=c2pa/{self.manifest_label}/c2pa.signature"
        self.image_format = image_format
        content_boxes = []
        self.assertion_store = assertion_store
        if self.assertion_store != None:
            content_box = ContentBox(box_type="cbor".encode("utf-8").hex(),
payload=self.generate_claim_schema())
            content_boxes.append(content_box)
        super().__init__(label="c2pa.claim",
content_type=c2pa_content_types["claim"], content_boxes=content_boxes)

    def generate_claim_schema(self):
```

```
        claim_schema = {}
        claim_schema["dc:format"] = self.image_format
        claim_schema["alg"] = "sha256"
        claim_schema['instanceID'] = 'xmp:fakeid:4124fae1-1da7-4a3f-95c8-
d8ae071bd059'
        claim_schema["claim_generator"] = self.claim_generator
        claim_schema["signature"] =
f"self#jumbf=c2pa/{self.manifest_label}/c2pa.signature"
        claim_schema["assertions"] = [
            {
                "url":
f"self#jumbf=/c2pa/{self.manifest_label}/c2pa.assertions/{get_assertion_lab
el(assertion.type)}",
                "alg": "sha256",
                "hash":
hashlib.sha256(assertion.get_data_for_signing()).digest()
            } for assertion in self.assertion_store.assertions
        ]

        return cbor_to_bytes(claim_schema)
```

Листинг 3 – Реализация класса `ClaimSignature`

```
class ClaimSignature(SuperBox):

    def __init__(self, claim, private_key, certificate):
        self.claim = claim
        self.private_key = private_key
        self.certificate = certificate
        content_boxes = []
        if self.claim != None and self.private_key != None and self.certif-
icate != None:
            content_box = ContentBox(box_type='cbor'.encode('utf-8').hex(),
payload=self.create_signature())
            content_boxes.append(content_box)
        super().__init__(content_type=c2pa_content_types['claim_signa-
ture'], label='c2pa.signature', content_boxes=content_boxes)

    # Sign by ps256 algo
    def create_signature(self):
        # -37 stands for PS256 (RSASSA-PSS using SHA-256 and MGF1 with SHA-
256)
        phdr = self.generate_protected_header()
        unprotected_header = {
            'temp_signing_time': str(datetime.datetime.now(pytz.utc)),
        }
        private_key = serialization.load_pem_private_key(self.private_key,
password=None)
        sig_structure_data = cbor2.dumps(cbor2.CBORTag(84, ['Signature1',
phdr, b'', self.claim.serialize()]))
        signature = private_key.sign(
            sig_structure_data,
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=32
            ),
            hashes.SHA256()
        )
        payload = None
        message =  [phdr, unprotected_header, payload, signature]
```

```
        tag = cbor2.CBORTag(18, message)
        cose_tag = cbor2.dumps(tag)
        pad = b'\x00' * (4096 - len(cose_tag))
        payload = cose_tag + pad
        return payload

    def generate_protected_header(self):
        certs_array = []
        certs = x509.load_pem_x509_certificates(self.certificate)
        for cert in certs:
            certs_array.append(cert.public_bytes(Encoding.DER))
        protected_header_map = {1: -37,
                                33: certs_array}
        return cbor2.dumps(protected_header_map)
```

Листинг 4 – Реализация встраивания C2PA манифеста в формат JPG

```
class JpgSegment():
    def __init__(self, payload_length, marker=bytes.fromhex('EB')):
        self.marker = marker
        self.payload_length = payload_length

    def serialize(self, payload):
        serialized_data = b''
        serialized_data += bytes.fromhex('FF') + self.marker
        serialized_data += self.get_segment_length().to_bytes(2, 'big')
        serialized_data += payload
        return serialized_data

class JpgSegmentApp11(JpgSegment):
    def __init__(self, segment_id, sequence_number, payload_length, pay-
load):
        self.ci = bytes.fromhex('JP'.encode('utf-8').hex()) # 2 bytes
        self.en = segment_id                                # 2 bytes
        self.z = sequence_number                            # 4 bytes
        self.app11_payload = payload
        super().__init__(payload_length = self.get_payload_length(pay-
load_length))

    def get_payload_length(self, payload_length):
        return 2 + 2 + 4 + 4 + 4 + payload_length

    def serialize(self):
        _en = self.en.to_bytes(2, 'big')
        _z = self.z.to_bytes(4, 'big')
        app11_payload = self.ci + _en + _z + self.app11_payload
        return super().serialize(app11_payload)

class JpgSegmentApp11Storage():
    def __init__(self, app11_segment_box_length, app11_segment_box_type,
payload):
        self.l_box = app11_segment_box_length
        self.t_box = app11_segment_box_type
        self.payload = payload
        self.serialized_length = 0

    def serialize(self):
        segment_id = 1
        sequence_number = 0
        payload_offset = 0
```

```
        payload_length = self.get_payload_length()
        app11_segments = []
        while (payload_length > 0):
            sequence_number += 1
            chunk_length = JPEG_SEGMENT_MAX_PAYLOAD_LENGTH
            if payload_length < JPEG_SEGMENT_MAX_PAYLOAD_LENGTH:
                chunk_length = payload_length
            app11_segments.append(JpgSegmentApp11(segment_id=segment_id,
                                                  sequence_number=sequence_number,
                                                   payload_length=chunk_length,
                                        payload=self.payload[payload_offset:]))
            payload_length -= chunk_length
            payload_offset += chunk_length
        serialized_storage_data = b''
        for app11_segment in app11_segments:
            serialized_storage_data += app11_segment.serialize()
        self.serialized_length = len(serialized_storage_data)
        return serialized_storage_data
```

Листинг 5 – Реализация интерфейса библиотеки «C2Py»

```
# Function for asserion creation.
def C2Py_GenerateAssertion(assertion_type: C2PA_AssertionTypes, asser-
tion_schema) -> Assertion:
    return Assertion(assertion_type, assertion_schema)

# Function for manifest store generation.
def C2Py_GenerateManifest(assertions: list, private_key: str, certifi-
cate_chain: str) -> ManifestStore:
    manifest = Manifest()
    assertion_store = AssertionStore(assertions=assertions)
    manifest.set_assertion_store(assertion_store)
    claim = Claim(claim_generator='C2Py', manifest_label=manifest.get_mani-
fest_label(), assertion_store=assertion_store)
    manifest.set_claim(claim)
    claim_signature = ClaimSignature(claim, private_key=private_key, cer-
tificate=certificate_chain)
    manifest.set_claim_signature(claim_signature)
    return ManifestStore([manifest])

# Function for emplacing manifest to source data
def C2Py_EmplaceManifest(format_type: C2PA_ContentTypes, content_bytes:
bytes, c2pa_offset: int, manifest: ManifestStore) -> bytes:
    if format_type == C2PA_ContentTypes.jpg:
        c2pa_jpg_app11_storage = JpgSegmentApp11Storage(app11_seg-
ment_box_length=manifest.get_length(),
                                                        app11_seg-
ment_box_type=manifest.get_type(),
                                                        payload=manifest.seri-
alize())
        return content_bytes[:c2pa_offset] + c2pa_jpg_app11_storage.serial-
ize() + content_bytes[c2pa_offset:]
    else:
        print(f'Unsupported content type {format_type}!')
        return b''
```

Приложение Б. Исходный код системы

Исходный код системы «Kandinsky Captions» представлен в листингах 6–7.

Листинг 6 – Реализация модуля взаимодействия с Kandinsky API

```
class Text2ImageAPI:
    def __init__(self, api_key, secret_key):
        self.URL = 'https://api-key.fusionbrain.ai/'
        self.AUTH_HEADERS = {
            'X-Key': f'Key {api_key}',
            'X-Secret': f'Secret {secret_key}',
        }

    def get_model(self):
        response = requests.get(self.URL + 'key/api/v1/models', head-
ers=self.AUTH_HEADERS)
        data = response.json()
        return data[0]['id']

    def generate(self, prompt, model, images=1, width=1024, height=1024):
        params = {
            "type": "GENERATE",
            "numImages": images,
            "width": width,
            "height": height,
            "generateParams": {
                "query": f"{prompt}"
            }
        }
        data = {
            'model_id': (None, model),
            'params': (None, json.dumps(params), 'application/json')
        }
        response = requests.post(self.URL + 'key/api/v1/text2image/run',
headers=self.AUTH_HEADERS, files=data)
        data = response.json()
        return data['uuid']

    def check_generation(self, request_id, attempts=10, delay=10):
        while attempts > 0:
            response = requests.get(self.URL + 'key/api/v1/text2image/sta-
tus/' + request_id, headers=self.AUTH_HEADERS)
            data = response.json()
            if data['status'] == 'DONE':
                return data['images']
            attempts -= 1
            time.sleep(delay)
        return None
```

Листинг 7 – Реализация компонентов клиентской части системы «Kandinsky Captions»

```
function GeneratePhoto({
    setUrl,}: {
    setUrl: (url: string) => unknown}) {
  const [value, setValue] = useState("");
  return (
    <div className='generate-photo'>
```

```
          <textarea
            className='input generate-photo__textarea'
            value={value}
            onChange={(e)=> setValue(e.target.value)}
          />
        <button
            className='button generate-photo__button'
            onClick={() => generateImage(value)}
        >
                Generate
        </button>
    </div>)
  async function generateImage(value) {
    const { data } = await axios.post('http://127.0.0.1:5000/image/', {
      text: value,
    });
    setUrl(`${data}`);}}
export default GeneratePhoto
function ResultPhoto({
    url,
}: {
    url: string;
}) {
  return (
    <div className='result-photo'>
        <img
            className='input result-photo__img'
            src={url || placeholderPath}
            alt="image"
        />
        <a
            className='button result-photo__download'
            href={url}
            download='PHOTO'
        >
            Download
        </a>
    </div>
  )
}

export default ResultPhoto
```