БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

факультет радиофизики и компьютерных технологий

Создание UML-диаграмм для графического описания ИС.

Проектирование модели данных

" Разработка интеллектуальной платформы для верификации мультимедийного контента в реальном времени"

Работу выполнили:

Островский И. В.

Шаковец И. А.

Сергиевич В. Д.

4 курс 5 группа

Минск, 2025

Введение

В данной работе разработана информационная система для **верификации мультимедийного контента в реальном времени**. Цель - описать структуру, поведение и взаимодействие компонентов системы с помощью UML-диаграмм и проанализировать подход к организации данных. На основе утвержденной темы были созданы диаграммы: сценариев использования (Use Case), компонентов (Component), активностей (Activity), последовательностей (Sequence) и модели данных (ER-диаграммы). По данным моделям делаются выводы о подходящем хранилище - в нашем случае **графовой БД** (Neo4j/JanusGraph) с дополнением реляционных и блочных хранилищ. Диаграмма вариантов использования (рисунок ниже) демонстрирует основные функции системы и роли пользователей.

Диаграмма вариантов использования (Use Case Diagram)

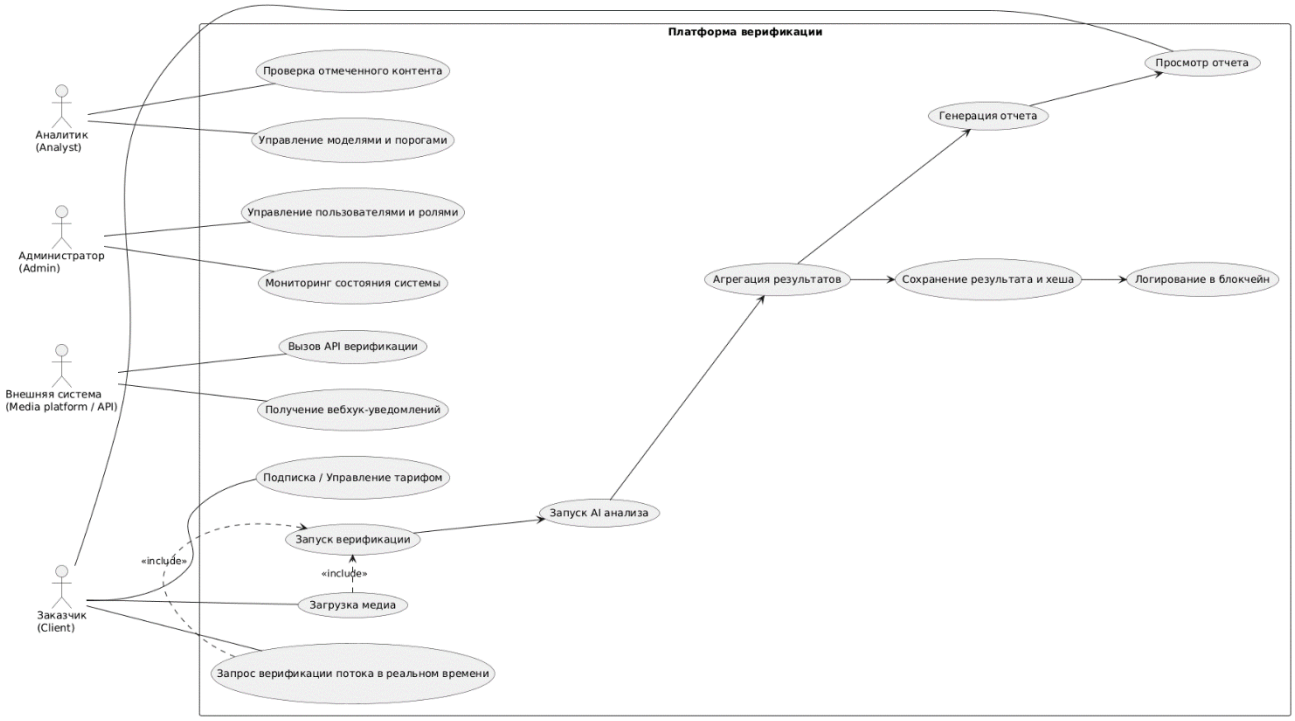
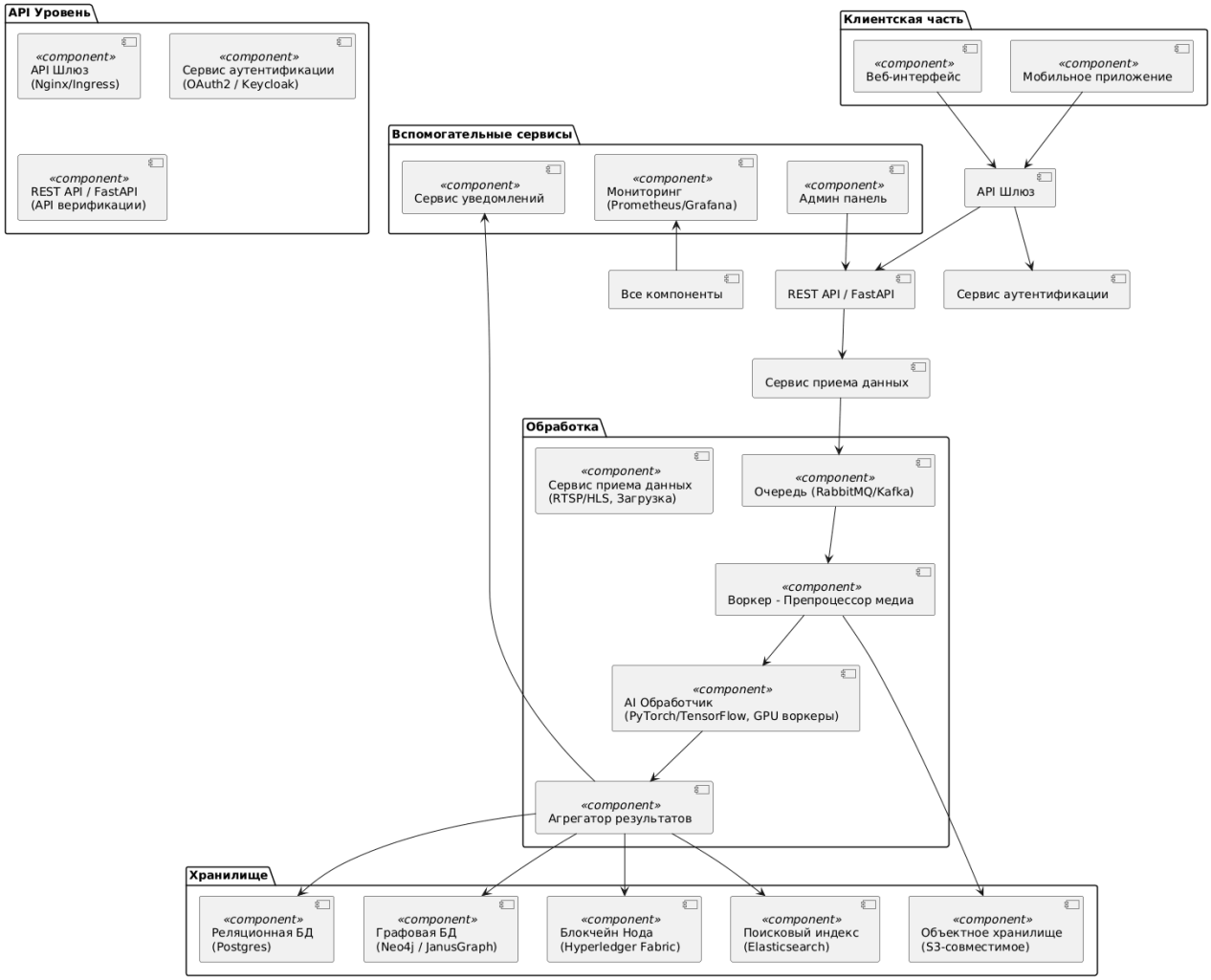


Диаграмма сценариев использования описывает, какой функционал системы доступен каждой роли. В системе выделены четыре главных типа актёров: **Заказчик (Client)**, **Аналитик (Analyst)**, **Администратор (Admin)** и **Внешняя система (Media/API)**. Основные случаи использования (use cases) сгруппированы по блокам «Загрузка», «Обработка», «Отчетность» и «Администрирование». Например:

- Заказчик взаимодействует с системой через «Доступ к платформе» - этот прецедент **включает** сценарии «Загрузить медиа», «Запросить проверку потока», «Управление планом подписки» и «Просмотр отчета».
- Процесс верификации описан цепочкой прецедентов: «Запустить верификацию» → «Запуск AI-анализа» → «Агрегация результатов» → «Сохранить результат и хеш» → «Запись в блокчейн».
  Последовательность от загрузки контента до генерации отчёта и просмотра связывает функционал обработки и отчетности.
- Вспомогательные прецеденты (прим. «Просмотр отмеченных записей», «Управление моделями», «Управление пользователями/ролями», «Мониторинг системы») доступны Аналитику и Администратору, показаны пунктирными связями.
  Таким образом, диаграмма демонстрирует, какие сценарии выполняются системой и кто их инициирует. Прецеденты «просто что» делает система, без отображения внутренней реализации (это качественно отражает функциональные требования).

Компонентная диаграмма (Component Diagram)



Компонентная диаграмма показывает архитектурную структуру системы - основные подсистемы и сервисы, а также их связи. В нашей системе выделены слои:

- **Клиентская часть:** Web-интерфейс и мобильное приложение, через которые пользователь обращается к платформе.
- **API-слой:** шлюз (API Gateway на основе Nginx/Ingress), сервис аутентификации (OAuth2/Keycloak), основной REST API (на FastAPI) для обработки запросов верификации.
- **Обработка:** сервис Ingest (приём медиа: RTSP/HLS, файлы) → **Очередь сообщений** (RabbitMQ или Kafka) → **Worker - медиа-препроцессор** (извлекает кадры/аудио) → **AI Processor** (модели на PyTorch/TensorFlow с GPU) → **Result Aggregator** (аггрегирует результаты).
- **Хранилища:** объектное хранилище S3 (оригиналы медиа и отчёты), реляционная СУБД Postgres (примитивные данные о пользователях, контенте, краткие результаты проверок), **графовая БД** (Neo4j/JanusGraph) для метаданных и связей (родство контента, происхождение), узел блокчейна Hyperledger Fabric (запись хешей), поисковый индекс Elasticsearch (быстрый поиск по данным).
- **Вспомогательные сервисы:** сервис нотификаций (отправка email/webhook с отчетами), система мониторинга (Prometheus/Grafana), административная панель (Admin Panel).
  Компоненты соединены по интерфейсам: Web UI и Mobile App обращаются к API Gateway, который коммуницирует с Auth Service и REST API; REST API передаёт файлы в Ingest, Ingest пушит задачи в очередь, которую обрабатывают Worker → AI → Aggregator. Aggregator записывает результаты в Postgres, обновляет связи в графовой БД и отправляет транзакцию в блокчейн; также нотифицирует клиента о готовности отчета. Мониторинговая система отслеживает все компоненты. Такой вид диаграммы даёт **концептуальную картину взаимодействия подсистем и артефактов** и соответствует распределенной архитектуре нашего проекта (микросервисы и различные БД).

Диаграмма активностей (Activity Diagram)

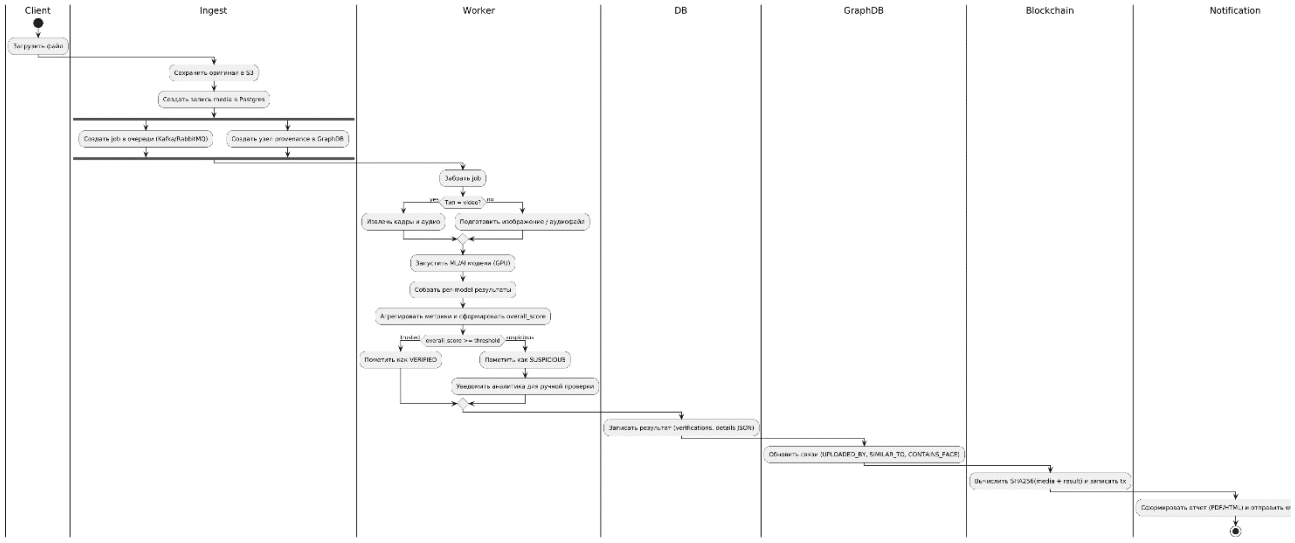


Диаграмма активностей отображает динамику процесса - в нашем случае процесс верификации загруженного медиа. По сути, это блок-схема, показывающая переход потоков управления от одной активности к другой. На диаграмме видны *беговые дорожки* (swimlanes) для роли Client, сервиса Ingest, Worker, баз данных и нотификаций, что наглядно разделяет ответственность. Основной поток выглядит так:

1. **Клиент** загружает файл.
2. **Ingest Service:** сохраняет оригинал в S3; создаёт запись о медиа в Postgres; параллельно добавляет задачу в очередь и создаёт узел происхождения (provenance) в GraphDB.
3. **Worker:** забирает задачу из очереди. Если это видео - извлекает кадры и аудио, иначе готовит изображение/аудиофайл.
4. Запускаются **ML/AI-модели** на GPU. Полученные детекции и баллы по моделям собираются и аггрегируются в итоговый overall_score.
5. Если overall_score >= threshold, контент помечается «**VERIFIED**». Иначе - помечается «**SUSPICIOUS**», и отправляется уведомление аналитика для ручной проверки.
6. **DB:** записываются результаты проверки (в таблицу verifications в Postgres, подробности - в JSONB).
7. **GraphDB:** обновляются связи (например, UPLOADED_BY (кто загрузил), SIMILAR_TO (схожие медиа), CONTAINS_FACE (обнаруженные лица)).
8. **Blockchain:** вычисляется SHA256 от файла и результата, отправляется транзакция в Hyperledger (фиксируется в Blockchain_Logs).
9. **Notification:** формируется отчёт (PDF/HTML) и отправляется клиенту через вебхук или email.
   Таким образом, диаграмма детально иллюстрирует процедуру

верификации - от загрузки до записи в блокчейн и рассылки отчета - наглядно показывая синхронизацию параллельных потоков (создание задачи и узла provenance одновременно) и ветвления (проверка условия на порог).

Диаграмма последовательностей (Sequence Diagram)

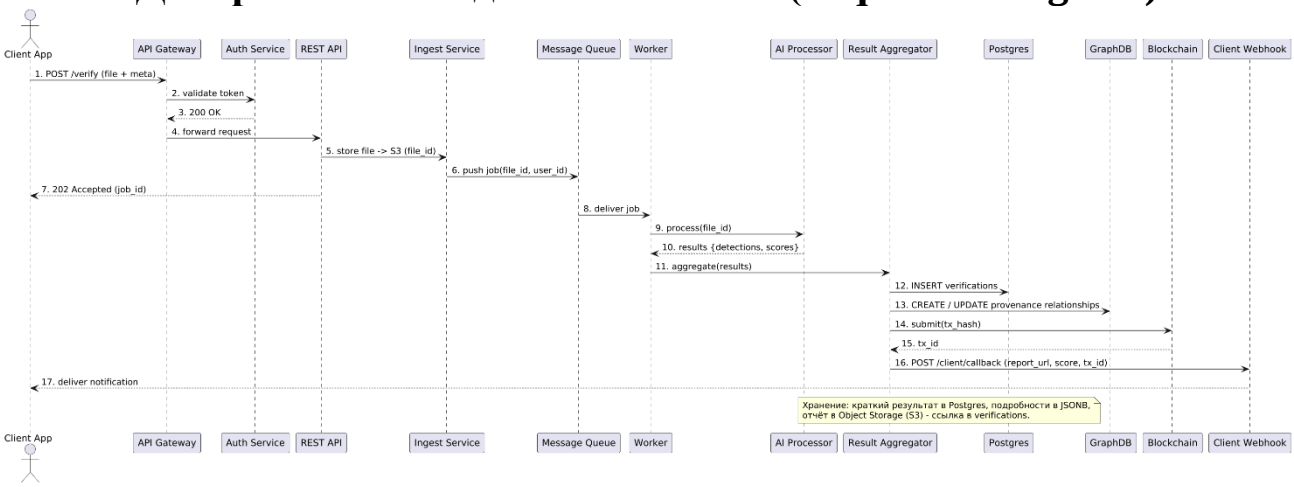


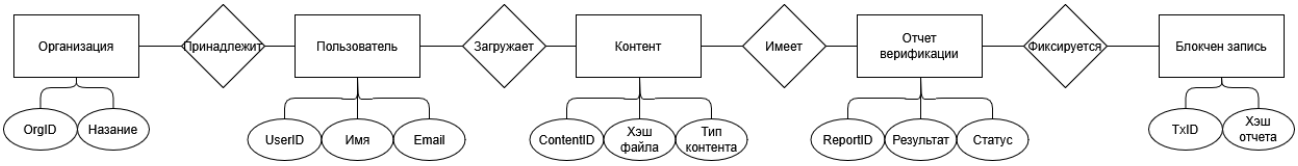
Диаграмма последовательностей показывает обмен сообщениями между компонентами системы во времени. На ней отражен сценарий API-взаимодействия при запросе верификации:

1. **ClientApp → API Gateway:** POST /verify (отправляет файл и метаданные).
2. **API Gateway → Auth Service:** проверка токена безопасности; Auth → Gateway: 200 OK.
3. **API Gateway → REST API:** пересылка запроса к внутреннему сервису.
4. **REST API → Ingest Service:** сохранение файла в S3 (возвращает file_id).
5. **Ingest → Queue:** push job(file_id, user_id).
6. **REST API → ClientApp:** возвращает 202 Accepted (job_id) - асинхронно.
7. **Queue → Worker:** задача доставлена воркеру.
8. **Worker → AI Processor:** process(file_id); AI → Worker: возвращает результаты (detections, scores).
9. **Worker → Result Aggregator:** aggregate(results).
10. **Result Aggregator → Postgres:** INSERT INTO verifications.
11. **Result Aggregator → GraphDB:** CREATE/UPDATE provenance relationships.
12. **Result Aggregator → Blockchain:** submit(tx_hash); Blockchain → Aggregator: возвращает tx_id.
13. **Result Aggregator → Client Webhook:** POST /client/callback с параметрами (report_url, score, tx_id).

14. **Client Webhook → ClientApp:** передает уведомление о завершении с указанием ссылки на отчет.
    Заметим, что **Result Aggregator** сохраняет в Postgres краткий результат, а полные детали (например, JSONB-схему детекций) могут храниться в S3 - ссылка на отчёт сохраняется в базе. Таким образом, последовательность демонстрирует взаимодействие компонентов (API Gateway, Auth, REST API, Ingest, очередь, воркер, AI, агрегатор, БД, граф и блокчейн) для обработки одного запроса верификации.

Модель данных: ER-диаграммы

Концептуальная модель «Сущность-Связь»

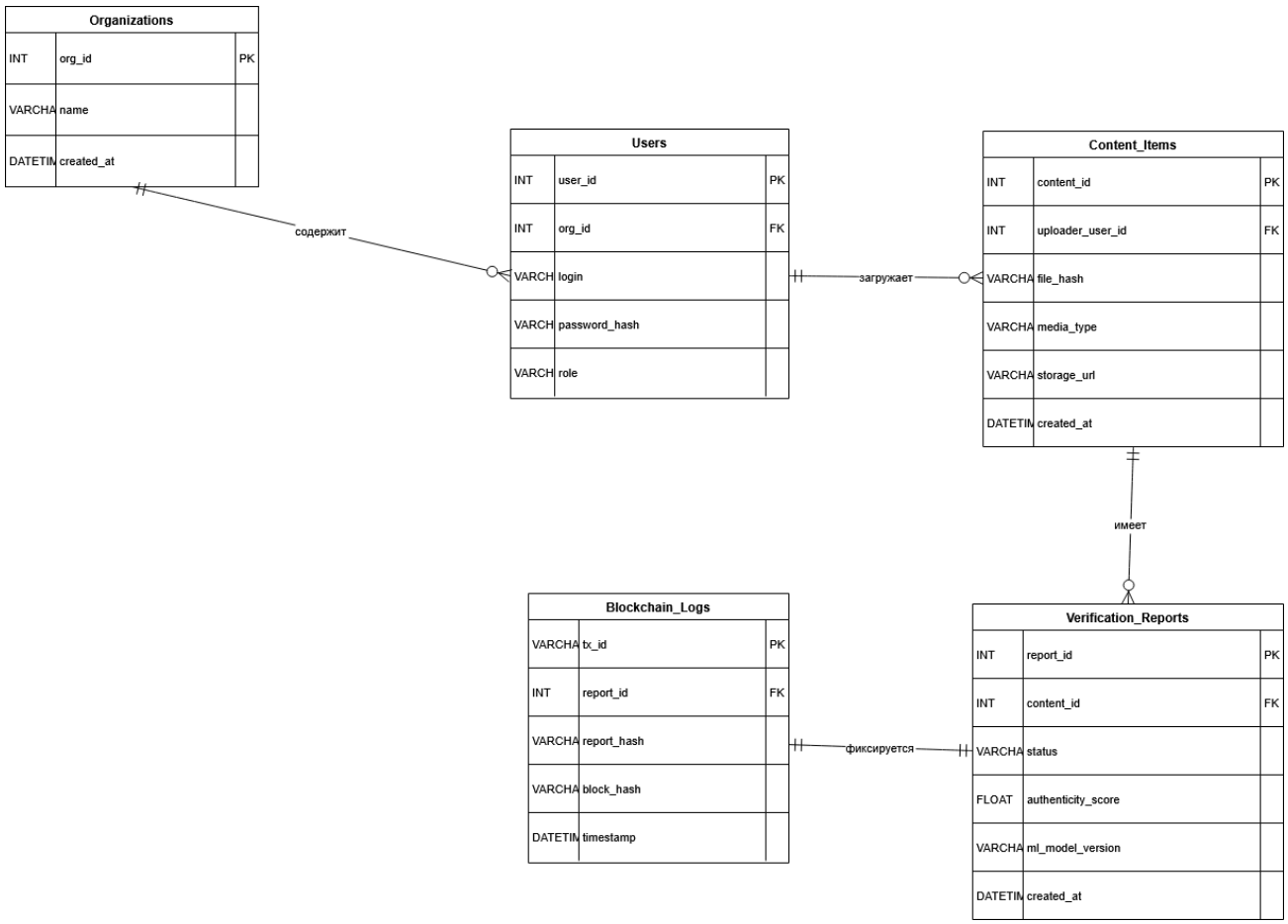


Концептуальная ERD отражает сущности предметной области и их отношения. Выделены основные сущности: **Пользователь (User)**, **Организация (Organization)**, **Контент (Content)**, **Отчет верификации (Verification Report)**, **Запись в блокчейне (Blockchain Record)**. Между ними налажены связи:

- Пользователь *«ПРИНАДЛЕЖИТ»* Организации (каждый пользователь принадлежит одной организации, у организации может быть много пользователей).
- Пользователь *«ЗАГРУЖАЕТ»* Контент (один пользователь может загрузить множество файлов).
- Контент *«ИМЕЕТ»* Отчет верификации (один контент может иметь несколько отчетов, например, повторные проверки).
- Отчет верификации *«ФИКСИРУЕТСЯ»* в одной Записи блокчейна (каждый отчёт соответствует одной транзакции).

Эти отношения обозначены на диаграмме «сущность-связь» и уточнены кратности (1:1, 1:N). Концептуальная модель подчёркивает, что именно **связи между сущностями** являются ключевыми: кто загрузил какой контент, какие отчёты связаны с контентом, и как результат фиксируется в блокчейне.

Логическая ER-диаграмма



Логическая ERD превращает концепции в конкретные таблицы базы данных и их атрибуты. Основные таблицы:

- **Organizations**: поля org_id (PK), name, created_at. Организация содержит пользователей.
- **Users**: user_id (PK), org_id (FK на Organizations), login (уникальный), password_hash, role (admin/user), created_at. Пользователь принадлежит организации.
- **Content_Items**: content_id (PK), uploader_user_id (FK на Users), file_hash (уникальный хеш файла), media_type (video/audio/image), storage_url (ссылка в S3), created_at. Описывает загруженные медиа.
- **Verification_Reports**: report_id (PK), content_id (FK на Content_Items), status (pending/complete/error), authenticity_score (оценка 0.0-1.0), ml_model_version, created_at. Отчеты привязаны к конкретному контенту (отношение 1 к N).
- **Blockchain_Logs**: tx_id (PK - хеш транзакции), report_id (FK на Verification_Reports, связь 1:1), report_hash (хеш отчёта для проверки), block_hash (хеш блока), timestamp. Каждая запись соответствует одному отчету.

Связи FK отражают концептуальную модель: *Organizations 1-N Users, Users 1-N Content_Items, Content_Items 1-N Verification_Reports, Verification_Reports 1-1 Blockchain_Logs*. Такая схема соответствует архитектуре: компоненты записывают данные в эти таблицы. Например, сервис **Result Aggregator** при обработке запроса создаёт запись в Verification_Reports и соответствующую в Blockchain_Logs; сервис **Admin Panel** может читать таблицы Users и Content_Items для отображения информации.

Выбор модели хранения данных

На основе модели данных обоснован выбор **графовой БД** для хранения связей между сущностями. Графовые СУБД явно хранят узлы (сущности) и ребра (отношения) как данные и используют индексы для быстрого перехода по связанным объектам. Это делает их эффективными при выполнении многошаговых запросов по цепочкам связей. В нашей системе существует множество таких связей (например, связь «Пользователь-Контент-Отчет-Блокчейн» и дополнительная топология сходства контента и лиц на изображениях). Графовый подход позволяет эффективно моделировать provenance (UPLOADED_BY, SIMILAR_TO, CONTAINS_FACE) и быстро искать связанные объекты без тяжёлых JOIN'ов, как это было бы в реляционной модели. Как отмечено в литературе, графовая БД - лучший выбор при работе со сложно взаимосвязанными данными, что подтверждает наш вывод. При этом часть информации (записи пользователей, отчётов) хранится также в реляционной БД для удобства хранения табличных атрибутов и транзакционной целостности. Таким образом, система использует **комбинацию моделей**: графовую БД для сетевых отношений и Postgres/S3 для хранения самих объектов и отчётов.

Заключение

Разработанные диаграммы образуют целостную картину системы. Диаграмма сценариев использования определяет основные функции и роли системы; компонентная диаграмма показывает, как эти функции реализованы в архитектуре (микросервисы и базы данных). Диаграммы активностей и последовательностей конкретизируют работу системы при ключевых операциях (например, верификация загруженного медиа), описывая пошаговый поток управления и сообщений между компонентами. Модель данных (ER-диаграммы) обеспечивает базис для хранения информации, необходимой описанным процессам: таблицы и связи соответствуют прецедентам и компонентам (например, Verification_Reports и Blockchain_Logs задействованы в процессе верификации). Все диаграммы взаимосвязаны: сценарии использования задают функциональные требования, которые покрываются

компонентами; сценарии процессов (Activity/Sequence) разъясняют логические шаги внутри этих компонентов; модели данных отражают информацию, с которой эти компоненты работают. Такой комплексный подход гарантирует, что система будет соответствовать как бизнес-требованиям, так и техническим ограничениям выбранной архитектуры.