ChatGPT

План и реализация рефакторинга документации проекта

1. Архитектурные слои проекта

Для начала необходимо чётко определить архитектурные слои системы BioETL и разграничить ответственность каждого слоя. Выделяются следующие ключевые слои:

- **Инфраструктурный слой** – отвечает за среду выполнения и общие технические механизмы проекта. Сюда относятся настройки окружения, зависимости, контейнеризация и CI/CD, а также **кросс-сервисные компоненты**: например, система HTTP-запросов с кэшированием, лимитированием и повторными попытками, механизм управления конфигурациями и секретами, подсистема логирования, инструменты пагинации и др. [1]. Инфраструктурный слой обеспечивает вспомогательные сервисы, не зависящие от предметной области, но необходимые для надёжной работы (например, структурированное логирование, резолвер конфигураций, обеспечение детерминизма запуска и т.п.).

- **Доменный слой** – отражает предметную область (биоактивность соединений) и связанные данные. В этом слое документируются **биологические концепции, терминология и метаданные**, используемые в проекте. В частности, описываются основные сущности доменной модели (например, *activity*, *assay*, *target*, *document*, *test item*) – то есть типы данных, с которыми оперирует система [2]. Также сюда входят описания внешних источников данных (ChEMBL, PubMed, UniProt и др.) и их биологический контекст, словарь используемых терминов (глоссарий) и структура метаданных, сопровождающих наборы данных (например, форматы файлов `meta.yaml`, отчётов качества и другие артефакты данных).

- **Прикладной слой** – отвечает за бизнес-логику обработки данных, реализуемую системой (ETL-процессы, пайплайны). Здесь описываются **пайплайны обработки данных и их компоненты**. Пайплайны (ETL-конвейеры) оркеструют полный цикл обработки – стадии извлечения, трансформации, валидации и сохранения данных. Каждый пайплайн состоит из множества компонентов, выполняющих узкоспециализированные задачи: клиенты внешних API для получения данных, трансформеры для нормализации, валидаторы для проверки данных по схемам, writers для сохранения результатов [3]. Документация прикладного слоя охватывает архитектуру пайплайнов, описание основных абстракций (например, базовый класс PipelineBase, паттерн ABC–Default–Impl), описание ключевых компонентов и сервисов (Unified API Client, Schema Registry, Validation Service и др.), а также руководства для разработчиков по расширению функциональности (например, как добавить новый пайплайн или новый тип компонента).

- **Интерфейсный слой** – обеспечивает взаимодействие пользователя или внешних систем с функциональностью BioETL. В текущем проекте в эту категорию входит, прежде всего, **интерфейс командной строки (CLI)** и вспомогательные утилиты. Документация этого слоя описывает, как запускать пайплайны и пользоваться возможностями системы с точки

зрения конечного пользователя или DevOps-инженера: поддерживаемые CLI-команды, опции, профили конфигурации, а также любые другие способы использования (например, программный API, если предусмотрен, или Jupyter Notebook демонстрации). В репозитории уже имеется раздел CLI (описание архитектуры CLI и перечень команд) [4], который будет отнесён к интерфейсному слою. Если в будущем появится UI или набор Jupyter Notebook'ов для демонстрации, их документация также войдёт в этот слой.

Закрепление слоёв. За каждым слоем будет закреплён отдельный раздел документации, чтобы информация о разнородных аспектах системы не перемешивалась. Такое деление позволит читателю легко находить интересующие его сведения: например, исследователь из области биоинформатики сможет обратиться к доменному слою за пояснениями о данных и терминах, разработчик – к прикладному слою за деталями реализации пайплайнов, а DevOps-инженер – к инфраструктурному слою за инструкциями по развёртыванию и настройке окружения.

2. Анализ текущей документации

Структура и содержание (до рефакторинга). Проект **BioactivityDataAcquisition** на данный момент уже содержит обширную документацию, которая выступает своего рода спецификацией системы – фактически техническим заданием на реализацию [5]. Исходного кода либо нет, либо он неполон, поэтому документация служит **единственным источником истины** о планируемой системе [5]. Структурно документация разделена на несколько категорий: **Overview**, **Guides**, **Reference**, **Architecture**, **Project Rules** и т.д. [6]. В этих разделах описаны ключевые концепции и компоненты системы – например, базовый класс PipelineBase, контракты ABC, клиенты внешних API, схемы данных – а также даны руководства для разработчиков (как добавить новый пайплайн, новый ABC) и описаны архитектурные решения и дорожная карта развития [6]. Основной README предоставляет навигацию по этим документам. Таким образом, на высоком уровне текущая документация охватывает многие аспекты: введение в проект, глоссарий, инструкции по запуску, детали реализации компонентов, описания схем данных, CLI и др.

Выявленные проблемы. Анализ существующих документов показал ряд существенных недостатков, которые необходимо устранить при рефакторинге:

- **Дублирование информации** – Одни и те же концепции описаны в нескольких местах с разной степенью детализации. Например, архитектура базового пайплайна (PipelineBase) повторно описана в документах разного раздела [7], что ведёт к избыточности и потенциальным противоречиям.
- **Отсутствие сквозных сценариев** – Нет цельного end-to-end описания процесса: нигде не прослеживается полный путь данных от вызова внешнего API до получения артефактов (файлов результатов, метаданных). Пользователю сложно понять целостную картину прохождения данных через систему [8].
- **Смешение аудиторий** – Документы не всегда явно разделены по целевой аудитории. Новичку в проекте предлагается читать ту же детализацию, что и core-разработчику, и наоборот – в результате некоторые руководства имеют чересчур технический уклон, а ключевые обзоры могут упускать детали, интересные разработчикам [9].
- **Несогласованность ссылок и структур** – В разных файлах одна и та же сущность или модуль могут именоваться или ссылаться по-разному. Например, часть документов ссылается на компоненты через путь `docs/reference/...`, а часть – через `docs/02-pipelines/...`, что может путать читателя [10]. Структура разделов (например, нумерация префиксов файлов) также неоднородна.

- **Неочевидность статусности API** – Не отмечено, какие части интерфейсов и контрактов считаются финальными и стабильными, а какие являются наброском/проектом. В результате читателю непонятно, на что можно опираться как на неизменную основу, а что ещё в разработке [11].

(Источник: анализ audit-отчёта по документации [8].)

Конкретные примеры подтверждают эти проблемы. **Дублирование:** архитектура пайплайнов (PipelineBase и связанные паттерны) описана трижды в разных разделах документации [7]. **Неполнота:** вводный глоссарий терминов содержит лишь часть терминологии – многие термины, упоминаемые в документах, отсутствуют [12]. **Смешение аудиторий:** в разделе CLI описана архитектура команд, но нет примеров конкретного использования команд, что затрудняет жизнь конечному пользователю [13]. **Несостыковки:** гайд для разработчиков `adding-new-pipeline.md` ссылается на несуществующий путь `docs/reference/pipelines/` [14]. Всё это указывает на необходимость серьезной переработки содержания и структуры документации.

Таким образом, основная цель рефакторинга – **устранить дублирования и противоречия**, **заполнить пробелы** (особенно в сценариях использования и определении терминов), **разделить документацию по аудиториям** и **выровнять структуру**. Ниже представлен план, как этого добиться, распределив материалы по четко определенным архитектурным слоям.

3. Новая структура документации по слоям

Исходя из вышеописанных слоёв архитектуры, предлагается перестроить документацию проекта таким образом, чтобы каждому слою соответствовал свой структурированный раздел. Это может быть реализовано как через новую структуру папок в репозитории, так и через секции в основном README (или на портале документации). Предпочтительно создать явную иерархию в директории `docs/`, отражающую разделение на слои – это упростит навигацию и сопровождение. Ниже представлена предлагаемая структура документации в виде дерева файлов (условные названия файлов и папок):

```
docs/                       # Корневая папка документации
├── index.md                # Главный индекс/обзор документации (введение, навигация)
├── domain/                 # **Доменный слой**: предметная область и данные
│   ├── overview.md         # Обзор предметной области (BioETL Domain Overview)
│   ├── glossary.md         # Глоссарий терминов
│   ├── data-sources.md     # Описание внешних источников (ChEMBL, PubChem, ... их биология)
│   ├── data-model.md       # Описание данных и метаданных (доменная модель, сущности, поля)
│   └── schemas/            # Подпапка со схемами данных (Pandera), если много файлов
│       ├── schema-index.md   # Обзор схем и реестра
│       ├── activity-schema.md # Конкретные схемы по сущностям...
│       └── ...
├── application/            # **Прикладной слой**: архитектура и реализация пайплайнов
```

```
│   ├── architecture.md    # Общий обзор архитектуры приложения (ETL-
архитектура, слои Pipeline/Component/Infrastructure)
│   ├── pipeline-base.md   # Спецификация базового пайплайна (PipelineBase) –
контракт и поведение
│   ├── components.md      # Обзор компонентов ETL (Extractor, Transformer,
Validator, Writer и пр.)
│   ├── pipelines/         # Подпапка с документацией конкретных пайплайнов
│   │   ├── chembl/        # Например, пайплайны для ChEMBL
│   │   │   ├── activity-pipeline.md   # Документация пайплайна Activity
(ChEMBL)
│   │   │   ├── assay-pipeline.md     # Документация пайплайна Assay (ChEMBL)
│   │   │   ├── ... (другие пайплайны ChEMBL)
│   │   │   └── common-components.md  # Общие для ChEMBL компоненты и базовые
классы
│   │   └── pubmed/ ...     # Аналогично для других источников (если есть/
планируются)
│   ├── guides/            # Руководства для разработчиков (раздел
прикладного уровня)
│   │   ├── adding-pipeline.md   # Как добавить новый пайплайн
│   │   ├── adding-component.md  # Как реализовать новый компонент или ABC
│   │   └── data-flow.md         # End-to-end сценарий: как данные проходят
через пайплайн
│   └── api-reference.md   # (Опционально) Сводный API-референс (перечисление
ключевых классов и методов, если нужно)
├── infrastructure/        # **Инфраструктурный слой**: окружение и
технические сервисы
│   ├── environment-setup.md   # Настройка окружения, установка зависимостей,
Docker (если будет), CI/CD
│   ├── configuration.md       # Работа с конфигурацией: профили, резолвер,
переменные среды
│   ├── logging.md             # Структурированное логирование (настройка и
формат логов)
│   ├── devops-guide.md        # CI/CD и развертывание (как запускаются
пайплайны в проде, мониторинг, golden files для QA)
│   └── tools.md               # Вспомогательные инструменты (скрипты
генерации диаграмм, тестирование и т.д.)
├── interface/             # **Интерфейсный слой**: взаимодействие с системой
│   ├── cli-overview.md        # Обзор CLI: архитектура командного интерфейса
│   ├── cli-commands.md        # Подробная справка по CLI: список команд,
опций и примеров
│   ├── running-pipelines.md   # Руководство пользователя: как запустить
пайплайн (quick start)
│   ├── examples.md            # Примеры использования: демонстрационные
сценарии, возможно кейсы/ноутбуки
│   └── output-artifacts.md    # Описание выходных артефактов: структура
каталогов, файлы meta.yaml, отчёты QC и т.п.
└── project/               # Проектные документы (правила, процессы)
    ├── rules.md               # Свод правил проекта (стили кода, оформления,
соглашения)
    ├── contributing.md        # Руководство для контрибьюторов (как
```

```
участвовать, требования к PR)
    └── roadmap.md            # Архитектурные решения и roadmap (ранее docs/
architecture)
```

(Примечание: Дерево представлено для иллюстрации; фактические названия файлов могут различаться. В иерархии отражены слои: domain, application, infrastructure, interface.)

Навигация через README. Основной `README.md` репозитория будет обновлён, чтобы отразить новую структуру. В нём будут разделы-ссылки на каждый слой, аналогично текущему содержанию README, но уже с разделением по слоям. Например, вместо прежних категорий “Overview/Guides/Reference” появятся разделы “Domain Documentation”, “Application (ETL) Documentation”, “Infrastructure”, “Interface (CLI)”, и т.д., внутри которых кратко перечислены соответствующие документы. Это обеспечит обратную совместимость для тех, кто начинает чтение с README, и направит читателя сразу в нужный раздел.

Обоснование структуры. Предложенная структура устраняет пересечения: каждая тема закреплена за одним разделом. Например, все сведения о данных и предметной области – в `docs/domain/`, всё о запуске и CLI – в `docs/interface/` и т.д. Это предотвратит ситуацию, когда одна и та же информация разбросана по разным папкам. Кроме того, структура масштабируется: по мере роста проекта новые пайплайны или компоненты будут добавляться в соответствующие подпапки, сохраняя порядок. Отметим, что в текущей документации уже прослеживались намёки на подобное разделение (например, выделены Infrastructure, CLI, Schemas в оглавлении [15] [4]), но мы делаем эту иерархию более явной и расширяем её доменным и интерфейсным слоями.

4. Переработка содержания документов по слоям

После определения структуры по слоям необходимо переработать содержание существующих документов, **распределив информацию по соответствующим разделам**. Будет проведен перенос и перераспределение текста из старых файлов в новые, с устранением дублирования и актуализацией формулировок. Ниже описано, как изменится содержание для каждого слоя.

Инфраструктурный слой: изменения в документации

Будут собраны воедино все материалы, касающиеся окружения и технических аспектов. В частности:

- **Окружение и установка зависимостей:** Создадим документ `environment-setup.md`, где соберём инструкции из раздела “Getting Started” (если такие имеются) и **Contributing** (в случае существования) о том, как развернуть проект локально. Здесь же опишем системные требования (версия Python, необходимое ПО вроде Graphviz для генерации диаграмм [16], и т.д.), установку библиотек, настройку переменных окружения. Например, если для запуска нужен Docker, опишем процесс сборки/запуска контейнера; если настроен CI (GitHub Actions), дадим ссылку или описание pipeline CI. Сейчас эти сведения частично разбросаны (например, в `README.md` или `docs/guides/getting-started.md`), их нужно централизовать.

- **Конфигурация приложения:** Документ `configuration.md` в инфраструктурном разделе станет **единым источником** информации о профилях конфигурации, резолвере конфигов, работе с переменными окружения и секретами. Ранее концепции конфигурации

описывались в нескольких местах – в пользовательском гайде (`docs/guides/configuration.md`), в справочнике по инфраструктуре (`docs/reference/infrastructure/config/`) и даже в примечаниях внутри pipeline-документов [17]. Мы объединим эти сведения, устранив противоречия. Документ опишет формат YAML-файлов конфигурации, систему переопределения параметров через профили (например, base.yaml + dev.yaml + prod.yaml), а также правила именования полей конфигов. Кроме того, будет приведён пример реального конфигурационного файла (`configs/pipelines/...yaml`) с пояснениями, что значительно прояснит использование (ранее примеров не хватало [18]).

- **Логирование и мониторинг:** Выделим отдельный файл `logging.md` для описания системы логирования. В нём стандартизируем терминологию (в проекте упоминается “UnifiedLogger” и структурированное логирование [19] – это должно быть чётко объяснено). Документ опишет, как настраивается уровень логов, формат вывода, куда пишутся логи (консоль, файлы), приведёт примеры лог-сообщений. Также опишем интеграцию с monitoring/alerting, если планируется (возможно, логирование ошибок и метрик качества). Ранее эти сведения только упоминались вскользь в документации по инфраструктуре, без единого места с примерами.

- **DevOps и CI/CD:** Добавим документ `devops-guide.md` или раздел в `infrastructure.md`, посвящённый сборке и развёртыванию. Опишем, как система используется в режиме производства: например, как выглядят CI-пайплайны (какие проверки выполняются, генерируются ли QC-отчёты и где хранятся), как организован Continuous Integration (можно сослаться на `.github/workflows/` если есть), как выполняется **deterministic build** и golden tests на артефакты (если это часть стратегии качества). Если используются Docker-образы, опишем процесс их сборки. Эта информация важна для инженеров, разворачивающих систему, и ранее не была собрана (частично могла присутствовать в архитектурных решениях). Теперь она будет явно описана в инфраструктурном разделе.

- **Вспомогательные инструменты:** В инфраструктурной документации также перечислим утилиты, поставляемые с проектом. Например, в текущем репо есть скрипты генерации диаграмм классов (`tools/generate_diagrams.py`) и инструкции к ним [20] [21]. Эти инструменты будут описаны в разделе (например, в `tools.md`): для чего предназначены, как использовать (с примерами команд), куда помещают результаты. Также сюда войдут описания стратегии тестирования (если есть autopilot тесты, фикстуры) и прочие dev-tools. Таким образом, инфраструктурная документация даст полную картину того, **как настроена и поддерживается техническая сторона проекта**.

Важно отметить, что после переработки **инфраструктурный слой документации будет предназначен главным образом для разработчиков проекта и DevOps-инженеров**. Он может быть пропущен обычными пользователями-исследователями, которые просто запускают готовые пайплайны, но крайне ценен для тех, кто сопровождает и развертывает систему.

Доменный слой: изменения в документации

Будут существенно дополнены и структурированы материалы по предметной области и данным. Основные изменения:

- **Обзор предметной области:** Добавится документ `overview.md` в папке `domain/`, который простыми словами описывает, **какую проблему решает проект и какие данные обрабатывает**. Здесь будет рассказано, что такое биоактивность соединений, почему важны такие базы данных, как ChEMBL, PubChem, etc., и какие типы сущностей (Compound, Target, Assay, Activity и т.д.) фигурируют. По сути, это расширенная постановка задачи и контекст для исследователей, читающих документацию. Эта часть ранее либо отсутствовала, либо была очень кратко упомянута во введении README. Мы дополним её, чтобы даже неподготовленный читатель из смежной сферы понял, о чём проект.

- **Глоссарий терминов:** Имеющийся `docs/overview/glossary.md` будет перенесён в `domain/glossary.md` и **существенно расширен**. Все специальные термины из документации и кода будут зафиксированы с определениями. Согласно аудиту, сейчас глоссарий содержит не все термины [12] – будем дополнять. Например, добавим отсутствующие понятия: *ETL*, *Pipeline*, *QC (quality control)*, *Activity/Assay/TestItem (с пояснением их смысла в ChEMBL)*, *Batch/Chunk*, *CrossRef*, *InChIKey* и т.д., то есть всё, что может вызвать вопросы. Для каждого термина укажем краткое определение и, при необходимости, ссылку на источник или раздел документации, где он подробнее описан. Глоссарий станет унифицированным справочником – чтобы убрать разночтения, мы в тексте документов будем стараться выделять термин и, при первом появлении, давать на него ссылку-глоссарий.

- **Внешние источники и данные:** Добавим документ `data-sources.md`, посвящённый краткому обзору внешних API/баз данных, из которых агрегируются данные. В нём перечислим подключенные источники (ChEMBL, PubMed, CrossRef, UniProt и др.) и для каждого дадим описание: что это за база, какие типы данных предоставляет, какие ограничения по API, ссылки на официальную документацию этих ресурсов. Это поможет читателям понять происхождение данных и возможно сходить к первоисточникам за деталями. Ранее такая информация подразумевалась, но явно не описывалась.

- **Модель данных и схемы:** Очень важный блок – **описание структур данных**, которые формируются в результате работы пайплайнов. Планируется документ `data-model.md`, где будет описано, какие сущности данных обрабатываются (Activity, Assay, Target и т.д.) и какие атрибуты у них есть. Фактически, это текстовое описание Pandera-схем: для каждой сущности перечислим поля, их типы и смысл. Например: "**Activity**: содержит поля `activity_id` (уникальный идентификатор), `assay_id` (ссылка на эксперимент), `compound_id`, `value`, `unit`, … – описание что означает каждое поле." Такое описание облегчит понимание содержимого выходных наборов данных. Можно включить сюда таблицы или перечисления полей. Частично эти сведения уже присутствуют в виде Pandera-схем (в коде или `docs/reference/schemas/*.md`), но мы представим их в удобочитаемой форме и сгруппируем по доменным объектам. При этом технические детали (например, ссылка на Pandera SchemaDefinition класс) можно вынести в приложение.

- **Документация схем (Pandera):** Возможно, стоит сохранить подробные схемы в виде отдельной секции (например, `domain/schemas/` подпапка). Если в проекте уже есть

файлы вроде `00-document-schema.md`, `01-testitem-schema.md` и т.д., они будут перенесены в доменный раздел. Они послужат расширением к `data-model.md`, давая детальное определение в терминах Pandera (типы данных, ограничения). Важно убедиться, что содержание этих файлов согласовано с общим описанием модели данных. Мы проверим, чтобы не было расхождений (по аудиту, сейчас описания схем были концептуальными и без примеров [22]). В переработке добавим примеры – например, кусочек YAML или JSON, соответствующий схеме, или маленький DataFrame, чтобы читатель видел реальную структуру.

- **Метаданные и артефакты:** Отдельно в доменном разделе (или интерфейсном, см. далее) опишем, какие **метаданные** сопровождают каждый запуск пайплайна. Проект генерирует QC-артефакты – отчёты качества, файлы с метриками и `meta.yaml` [23]. Необходимо объяснить, что содержится в `meta.yaml` (например, версия исходного источника, дата выгрузки, параметры фильтрации и пр.), как выглядят отчёты качества (например, распределение значений, % пропущенных данных). Эти артефакты – часть доменной сущности данных (они описывают данные) и потому логично их описать здесь же. В текущей документации есть раздел Quality Control, но его нужно доработать с точки зрения понятности для пользователя. После рефакторинга любой пользователь сможет понять, **какие файлы он получит на выходе и что в них означает каждый элемент**.

Доменный раздел, таким образом, станет исчерпывающим справочником по данным: **что за данные обрабатываются, откуда они получены, каковы их структура и ограничения, каковы ключевые термины**. Это особенно важно для пользователей-биоинформатиков, которым нужны контекст и уверенность в достоверности данных. Разумеется, доменная документация будет связана с прикладной (через ссылки: например, описание поля `assay_id` может сослаться на раздел про Assay Pipeline), но при этом оставаться понятной автономно.

Прикладной слой: изменения в документации

Будет консолидирована и очищена документация, относящаяся к реализации ETL-пайплайнов и компонентов. Основные направления изменений:

- **Обзор архитектуры:** Документ `architecture.md` (ранее `architecture-overview.md`) станет единым высокоуровневым описанием архитектуры BioETL. В нём структурированно изложим, **как разделяется логика по слоям Pipeline / Component / Infrastructure**, какие паттерны используются (например, ABC-контракты для компонентов, шаблон “ABC + Default + Impl”), и как данные движутся через систему. Ранее эта информация была раздроблена: частично в `docs/overview/architecture-overview.md` (с перечислением слоёв Pipeline/Component/Infrastructure) и частично в `docs/02-pipelines/00-pipeline-base.md` (детали PipelineBase) – причём они частично дублировали друг друга [7]. Теперь мы объединим эти описания: **PipelineBase** и общие стадии Extract-Transform-Validate-Write будут описаны сразу в обзоре архитектуры, единообразно. Также сюда включим общую диаграмму или схему компонентов (например, схему классов основных абстракций) для наглядности. Если нужно, дадим отсылку на подробности в референсах, но без повторов текста. Эта консолидация устранит ситуацию, когда одна и та же концепция (PipelineBase) описана тремя разными способами [24].

- **Спецификации базовых компонентов:** В прикладной секции появятся документы, каждый из которых фокусируется на отдельной группе компонентов или сервисов ядра. Например, `components.md` – обзор всех компонентных абстракций (Extractor, Transformer,

Loader/Writer, Validator, Descriptor и т.д.) с указанием, за что каждая отвечает, и ссылками на их интерфейсы. Отдельно может быть документ по **Unified API Client** (единый HTTP-клиент для внешних API) и по **Validation Service/Schema Registry** – они уже были в reference-разделе [25], но мы убедимся, что их описание не повторяет то, что сказано в `architecture.md`. Возможно, их стоит объединить: например, один документ "Core Services" описывает и реестр схем, и сервис валидации, и UnifiedOutputWriter. Таким образом, **каждый ключевой компонент системы будет описан ровно один раз** в соответствующем документе, с необходимой детализацией. Это позволит убрать дубли (по аудиту, у нас есть параллельные описания BaseExternalDataClient в разных файлах, противоречащие друг другу [26] – после рефакторинга останется одно авторитетное описание актуального клиента).

- **ABC-паттерн и расширение системы:** Особое внимание уделим документированию архитектурного паттерна "ABC/Default/Impl", который широко используется. Сейчас правило использования ABC (Abstract Base Class) разбросано между проектными правилами и гайдлайнами для разработчиков [27]. Мы создадим унифицированное описание этого паттерна (например, в `project/rules.md` или отдельном `architecture.md` разделе) – с объяснением, что все компоненты имеют интерфейс ABC, дефолтную реализацию (DefaultImpl) и конкретные Impl при необходимости. Объясним, для чего это сделано (интерфейсы для гибкости, возможность подменять реализацию). Также отметим, какие контракты на данный момент стабильны. Например, можем пометить, что интерфейс PipelineABC окончательно утверждён и менять его не планируется, а вот интерфейс StageHookABC пока экспериментальный. Такое **явное маркирование стабильных контрактов** внедрим прямо в текст (например, значком или пометкой). Тем самым реализуем рекомендацию аудита [11] [28] – сделать понятным статус каждой части системы.

- **Документация конкретных пайплайнов:** Все материалы, относящиеся к отдельным пайплайнам (например, ChEMBL Activity Pipeline и др.), будут перемещены в структуру `docs/application/pipelines/<provider>/<pipeline>.md`. Мы будем придерживаться стандартизированного шаблона для пайплайн-документации, чтобы единообразно описывать каждый ETL. В репозитории уже есть заготовка шаблона [29] [30] – разделы "Description", "Stages", "Configuration", "Key Methods", "Usage" и т.д. Мы применим этот шаблон ко всем существующим пайплайнам. Например, для **ChEMBL Activity Pipeline** у нас будет документ с описанием назначения этого пайплайна, используемых классов (Extractor, Transformer...), перечислением стадий и ключевых методов (как в текущем `00-activity-chembl-overview.md` [31] [32], но с доработками). Также опишем конфигурацию этого пайплайна (какие параметры из YAML влияют на работу). Аналогично для **Assay Pipeline**, **Target Pipeline** и прочих. Общие для провайдера (ChEMBL) компоненты, такие как ChemblBasePipeline, ChemblClient, будут вынесены либо в отдельный документ "common-components.md", либо описаны в каждом пайплайне с отсылкой на общую секцию. Главное – обеспечить, что читатель, открывая документацию пайплайна, видит **полную картину работы этого пайплайна** и не вынужден прыгать по трём разным файлам. Если нужно, можем внутри пайплайновой документации давать ссылки на соответствующие разделы: например, при упоминании `ActivityExtractor` сослаться на документ "Extraction Stage" или на общий справочник Extractor'ов.

- **End-to-End сценарий:** Одним из ключевых дополнений станет документ `data-flow.md` (либо раздел внутри `architecture.md`), где мы **опишем полный сценарий прохождения данных через систему**. Этот документ отвечает на вопрос: *"Что происходит, когда пользователь запускает пайплайн?"* – начиная от чтения конфигурации,

затем вызова API, получения данных, их трансформации, валидации, записи и генерации артефактов. Мы возьмём конкретный пример (скажем, пайплайн Activity с небольшим набором ID) и проведём читателя через все стадии, указывая, какие модули срабатывают. В тексте будут упоминаться компоненты (Client, Extractor, Transformer...) в последовательности, возможно с диаграммой потока данных. Уже сейчас в архитектурном обзоре упоминалась общая схема конвейера [33], но она была краткой. Мы расширим её до полноценного раздела. Это поможет устранить выявленный недостаток — **неполноту end-to-end описания** [34]. После добавления этого сценария и примеров использования, читатель (особенно новый участник проекта) сможет лучше понять взаимосвязь всех частей.

- **Руководства для разработчиков:** В приложении (или отдельной подпапке `guides/`) будут обновлены документы наподобие “Как добавить новый пайплайн” (`adding-new-pipeline.md`) и “Как добавить новый ABC/компонент” (`adding-new-abc.md`). Мы приведём их в соответствие с новой архитектурой: исправим пути импортов и ссылок (в аудите указывалось на битые ссылки в этих файлах [35]), уберём избыточное дублирование (например, вместо повторного объяснения принципа ABC – дадим ссылку на раздел с паттерном). В то же время дополним эти гайды примерами кода, где возможно. Ранее код-фрагменты были, но не проверялись из-за отсутствия реализации. Когда код появится, стоит убедиться, что примеры работают, или пометить их как псевдокод. Таким образом, гайдлайны станут практичнее и однозначнее, что важно для core-разработчиков, расширяющих систему.

- **Обновление ссылок и кросс-референций:** Поскольку мы меняем структуру, нужно проследить, чтобы **все ссылки в тексте были обновлены** на новые пути. Например, вместо `docs/02-pipelines/core/04-unified-output-writer.md` будет ссылка на `docs/application/components.md#unified-output-writer` или соответствующий документ. Мы приведём ссылки к единому стилю (относительные пути внутри `docs/`), что решит проблему противоречивых путей [10]. Также настроим перенаправления или пометки устаревших разделов (если, например, кто-то придёт по старому URL, в документе можно оставить уведомление о перемещении раздела).

После этих изменений **документация прикладного слоя станет консистентной и удобной для разработчиков**. Новичок-разработчик сможет прочитать architecture.md для общей картины, затем перейти к нужному компоненту или пайплайну для деталей, не натыкаясь на противоречивые описания. А пользователи-исследователи при желании тоже могут заглянуть сюда, чтобы понять, как “под капотом” реализовано получение и очистка данных, но при этом основные пользовательские вещи будут отделены в интерфейсном разделе (см. ниже).

Интерфейсный слой: изменения в документации

Документация, ориентированная на конечных пользователей и интеграторов, будет пересмотрена с упором на практическое использование системы. Ключевые изменения:

- **Руководство пользователя (Getting Started):** Документ `running-pipelines.md` станет последовательной инструкцией для нового пользователя, желающего запустить существующий пайплайн. Здесь будет описано: как установить/подготовить окружение (с отсылкой в инфраструктурный раздел при необходимости), как настроить конфигурационный файл или воспользоваться дефолтным, и как именно вызвать CLI-команду для запуска. В тексте добавим конкретные примеры команд **CLI**, которых сейчас

не хватает [13]. Например: `bioetl run pipeline --pipeline chembl_activity --output ./results/` – условная команда, и что она сделает. Опишем ожидаемый результат запуска (какие файлы появятся, где лог). Ранее раздел “Running Pipelines” был концептуальным и без конкретики; после изменений это будет пошаговый гайд (“1. Установите зависимости, 2. Сконфигурируйте профиль, 3. Запустите команду, 4. Проверьте выходные данные”).

- **Справочник по CLI:** Сгруппируем всю информацию о командной строке в разделе `cli-overview.md` + `cli-commands.md`. Документ **CLI Overview** кратко опишет архитектуру CLI (что есть единая точка входа `bioetl` с субкомандами, как они устроены, если нужно – что каждая команда под капотом вызывает определённый PipelineRunner и т.п.). Документ **CLI Commands** перечислит **все доступные команды и опции**. Например: `bioetl run pipeline <pipeline_code> [--profile X] [--output Y] [--limit N]` – описание, что делает эта команда; `bioetl list pipelines` – что выводит; `bioetl validate config` – проверяет конфиги и т.д. Если команд немного, можно всё в одном файле; если много – структурируем по подкомандам. Также включим сюда **примеры**: запуск конкретного пайплайна, валидация конфигурации, генерация артефактов. Сейчас подобная информация либо отсутствует, либо разбросана, её консолидация значительно улучшит опыт пользователя CLI.

- **Примеры и кейсы использования:** Добавим документ `examples.md` или несколько тематических примеров, которые показывают, **как система может использоваться на практике**. Например, “Case Study: Импорт данных биоактивности для списка соединений” – где пользователь редактирует YAML-конфиг, запускает пайплайн с опцией `--limit`, получает результат и затем использует его (строит график, анализирует). Для таких примеров можно даже предоставить Jupyter Notebook (в репозитории или в документации), но если выполнять нечего (код отсутствует), то хотя бы описать мысленно. Цель – показать исследователю, **какую пользу он может извлечь** из инструмента. Это отвечает на запрос на добавление end-to-end сценариев использования [36], но уже с точки зрения пользователя (в отличие от data-flow для разработчиков, который мы добавили в прикладной секции). Эти примеры также станут хорошей проверкой целостности: читая их, можно убедиться, что все необходимые сведения (как настроить, где взять идентификаторы, как интерпретировать результат) есть в документации.

- **Выходные данные и качество:** Добавим документ `output-artifacts.md`, где опишем формат результатов, получаемых пользователем при успешном выполнении пайплайна. Здесь подробно разъясним структуру папки результатов: например, папка с CSV/Parquet файлами основных данных, файл `meta.yaml` с метаданными запуска, папка `qc/` с отчётами качества. Разъясним, как читать `meta.yaml` (что там хранится: версия источника, фильтры, timestamp и пр.), какие показатели могут быть в QC-отчётах (например, число записей, % пропущенных значений, статистики). Эти сведения были частично описаны в разделе QC Artifacts [37], но сейчас мы оформим их с позиции пользователя – **что значат эти артефакты и как их использовать**. Это особенно важно для доверия к данным: исследователь сможет быстро взглянуть на отчёт и понять, насколько качественны полученные данные и соответствуют ли ожиданиям.

- **Программный интерфейс (API):** Если планируется предоставить возможность использовать BioETL как библиотеку (импортировать и вызывать пайплайны из Python-кода, а не через CLI), этот аспект тоже нужно отразить. Например, можно кратко упомянуть в интерфейсной документации, что существует Python API: `from bioetl import`

`run_pipeline` или создание объекта Pipeline и вызов метода `run()`. Пока код не реализован, возможно, это рано детализировать. Но мы оставим пространство для этого: либо раздел “Using as a library” в `interface/examples.md`, либо просто упоминание, что “Также вы можете вызывать пайплайны программно, подробнее см. документацию для разработчиков”.

Таким образом, **интерфейсный слой документации станет понятным руководством для конечных пользователей и интеграторов**. Он будет написан более простым языком (насколько возможно, учитывая предметную область), сфокусирован на “как пользоваться” вместо “как это устроено”. Все упоминания внутренних деталей (классов, модулей) будут либо убраны, либо даны со ссылкой на более техническую часть документации, чтобы не загромождать картину. После такой переработки даже новый пользователь, не знакомый с внутренней архитектурой, сможет установить инструмент, настроить его под свои нужды и запустить пайплайн, опираясь только на раздел интерфейса.

5. Итоговая структура и шаблоны оформления документации

После рефакторинга документация будет оформлена как **целостный технический отчёт**, разбитый на разделы по слоям. Все документы будут написаны в единообразном стиле и формате (Markdown). Мы соблюдём стандартные соглашения оформления, принятые в open-source проектах, чтобы документация выглядела профессионально и её было легко читать и поддерживать.

Файловая структура: Структура папок/файлов приведена выше в виде дерева. Каждому крупному разделу (слою) соответствует отдельная директория, внутри которой файлы разбиты по темам. Такой подход облегчает навигацию и позволяет при необходимости собирать документацию в виде сайта. Например, можно задействовать Sphinx или MkDocs для генерации сайта из этих файлов – структура уже логична и готова для этого.

Формат Markdown и Sphinx: Мы решили использовать Markdown в качестве основного формата документации, так как он простой и поддерживается напрямую на GitHub. При этом для возможности генерации красивого HTML-портала документации будет рассмотрено использование Sphinx с парсером MyST (поддержка Markdown внутри Sphinx) [38]. Это даст нам лучшее из двух миров: писать в Markdown, но иметь мощные возможности Sphinx (генерация оглавлений, перекрёстных ссылок, единого поиска). Настройки Sphinx (conf.py) будут обновлены, чтобы индексировать новые разделы и подключить расширение `myst_parser` [39]. Таким образом, проект сможет публиковать документацию, например, на ReadTheDocs или GitHub Pages, без изменения формата файлов.

Единый стиль заголовков: Во всех файлах документации будет выдержана согласованная иерархия заголовков. Каждый документ начинается с заголовка первого уровня (`# Title`), обычно совпадающего с названием файла и отражающего тему. Внутри документа используются подзаголовки второго-третьего уровня для структуры. Например, в руководстве “Running Pipelines” будут разделы `## Requirements`, `## Steps`, `## Examples`. В описании архитектуры – `## System Components`, `### Pipeline Layer`, `### Component Layer` и т.д. Мы избегаем пропусков уровней и следим, чтобы заголовки были информативными и краткими. Также введём соглашение: один документ – одна тема, чтобы заголовок документа чётко отражал его содержание (например, “Schema Registry” или “ChEMBL Activity Pipeline – Overview”).

Навигационное оглавление (ToC) будет либо вручную составлено в index.md, либо авто-сгенерировано инструментом.

Шаблоны документов: Чтобы гарантировать единообразие, будут разработаны или использованы существующие **шаблоны** для часто повторяющихся типов документов. Например, как уже упоминалось, имеется шаблон для документации пайплайна [40] – мы его локализуем (переведём на русский, при необходимости) и применим ко всем пайплайнам. Аналогично можно завести шаблон для описания нового API-клиента или компонента: структура "Описание – Интерфейс – Методы – Пример использования – Связанная документация". Эти шаблоны могут храниться в `docs/templates/` для будущих контрибьюторов. Благодаря этому новый документ будет создаваться по образцу, и стиль изложения останется единым по всему проекту.

Использование списков и таблиц: В тексте будем активно использовать буллет-листы и таблицы для структурирования информации, где это уместно. Например, перечисление шагов пайплайна (Extract/Transform/...), требуемых зависимостей, полей конфигурации – всё это лучше воспринимается списком, чем сплошным текстом. В уже существующих черновиках документации такие элементы присутствуют (списки, таблицы сравнения) – мы их сохраним и улучшим. Например, таблицу терминов и определений можно оформить как двухколоночную: термин – определение. Таблицы из audit-отчёта будут не нужны пользователю конечному, но метод представления информации из них возьмём на вооружение для других разделов (например, таблица тем покрытия, возможно, трансформируется в checklist прогресса разработки, но это уже внутреннее).

Фрагменты кода и конфигураций: Документация будет включать **блоки кода** там, где необходимо показать пример или шаблон. Это может быть как Python-код (например, пример использования API клиента), так и YAML-конфигурация или shell-команды. Мы будем оформлять их в Markdown-блоки с подсветкой синтаксиса. Например, для демонстрации конфигурации пайплайна предоставим отрывок YAML:

```yaml
# Пример pipeline-конфига (chembl_activity.yaml)
pipeline:
  code: "activity_chembl"
  source: "ChEMBL API v33"
  parameters:
    batch_size: 25   # 25 ID на запрос (ограничение API)
    chunk_size: 10   # 10 батчей объединяются при записи
```

или пример Python-кода для программного запуска:

```python
from bioetl.pipelines import ChemblActivityPipeline

pipeline = ChemblActivityPipeline(config="configs/pipelines/chembl/
activity.yaml", run_id="test_run")
pipeline.run(output_dir="output/test_run", limit=100)
```

Такие примеры делают документацию более практической. Мы удостоверимся, что каждый такой фрагмент сопровождается поясняющим текстом (что он делает, какой ожидается результат). При возможности, кодовые блоки будут проверены на корректность.

Диаграммы и визуализация: Для лучшего восприятия сложных взаимодействий, мы включим диаграммы. Благодаря интеграции Sphinx/Markdown, можно использовать PlantUML или Mermaid прямо в документации. Например, чтобы изобразить зависимость классов или последовательность стадий, добавим Mermaid-диаграмму:

```
flowchart LR
    subgraph Pipeline
        A[Extract Stage] --> B[Transform Stage] --> C[Validate Stage] -->
D[Write Stage]
    end
    E[(External API)] --> A
    D --> F[(Output Files)]
    D --> G[(QC Reports)]
```

(Диаграмма: упрощённый поток данных от внешнего API через стадии пайплайна к выходным файлам.)

Проект уже содержит инструменты для генерации UML-диаграмм классов (с помощью pyreverse) [41]. Мы можем включить полученные SVG/PNG диаграммы в соответствующие разделы (например, диаграмму классов пакета `bioetl.clients` – в документацию клиентов). Если эти изображения будут храниться в репозитории, мы добавим их через Markdown (). Также, при обновлении кода, не забудем обновлять диаграммы, как описано в инструментах [42]. В целом, визуальные элементы значительно повысят качество документации, особенно для архитектурных разделов.

Внутренние и внешние ссылки: Документация будет богата перекрёстными ссылками – это одно из преимуществ структурированного подхода. Мы настроим, чтобы можно было ссылаться на секции других файлов, используя референсы. Например, в тексте про PipelineBase дадим ссылку на “Раздел 2.1 – Архитектура пайплайнов” в architecture.md, либо прямо на класс PipelineBase в reference (если сделаем отдельный reference API). Также будем ссылаться на внешние ресурсы: на документацию API ChEMBL, на статьи о Pandera, на определение терминов в Wikipedia при необходимости. Все ссылки будут оформлены явным образом (Markdown синтаксис или Sphinx roles, если используем его). Это повысит доверие и позволит при желании углубиться.

Язык и терминология: Поскольку документация пишется на русском, мы будем соблюдать аккуратность в употреблении англоязычных терминов. Названия классов, функций, библиотек естественно оставляем на английском (как кодовые идентификаторы), но даём их краткое описание на русском. Специфические термины (pipeline, commit, CLI) возможно оставить транслитерированно или в оригинале, если они общеупотребимы среди русскоязычных разработчиков. Важно – использовать **единые термины во всех разделах**. Например, если мы решили называть Pipeline “конвейер” в одном месте, то всюду придерживаться либо оригинального “пайплайн”, либо перевода, чтобы не было смеси. Будем сверяться с глоссарием и соблюдать его. Стиль изложения – технический, но понятный: короткие фразы, активный залог, по возможности без избыточной канцелярщины. Там где нужно перечисление, лучше дать список, чем прятать в тексте.

Подводя итог, после внедрения новых шаблонов и правил оформления, документация приобретёт профессиональный вид. Она будет легко читаться в виде исходных `.md` файлов на

GitHub, а при генерации через Sphinx (опционально) – превратится в структурированный сайт с оглавлением, что особенно удобно при объёме ~100k слов.

6. Объём и детализация рефакторинга (≥100 000 слов)

Рефакторинг документации предполагается **чрезвычайно объёмным и подробным** – суммарный объём текстов превысит 100 тысяч слов. Такой объём обусловлен стремлением полностью задокументировать все аспекты системы для требовательной аудитории (исследователи в биоинформатике, разработчики). Чтобы достичь этого объёма осмысленно, мы запланировали насыщение каждой секции большим количеством содержательного материала:

- **Расширение пояснительных разделов:** Вводные части (обзор проекта, описание предметной области) будут подробно описывать контекст, включая возможно краткий экскурс в биологию и химию, стоящую за данными. Мы добавим описания для читателей, не знакомых с конкретными базами (что такое ChEMBL, какие данные он содержит). Такое контентное наполнение увеличит объём, одновременно принося пользу в понимании.

- **Детальный разбор данных:** Каждая сущность данных и каждое поле будут объяснены. Например, вместо одной строки “TestItem Schema defines structure of test items” будет таблица или список полей `TestItem` с 5-10 строчными описаниями каждого поля (единицы измерения, диапазоны значений, ссылки на определение в оригинальной базе). Таких деталей много, и они дадут сотни дополнительных слов, делая документацию самодостаточной для анализа данных.

- **Примеры и кейсы:** Мы включим достаточное количество примеров использования – команды, конфиги, фрагменты данных – которые сами по себе увеличивают объём (например, каждый пример YAML или JSON – десятки строк). Но главное – каждый пример будет снабжён пояснением, порой более длинным, чем сам пример. Например, приведя пример команды CLI, мы далее распишем: что она делает, какие бывают варианты её опций, каких ошибок ожидать. Это одновременно обучает пользователя и добавляет текст.

- **Описание каждого компонента и класса:** Reference-часть (спецификации API) тоже займёт значительный объём. Мы опишем методы каждого важного класса: их сигнатуры, параметры, возвращаемые значения, а также семантику (что метод делает, в какой очередности вызывает подметоды). В текущих черновиках уже было начато описание методов Pipeline (например, `run`, `extract`, `transform` с параметрами) [43] [32]. Мы продолжим в том же духе для всех основных компонентов. Хотя такая информация частично дублирует docstring кода, её наличие в общей документации полезно для быстрого обзора. Сотни методов × описание по 1-2 предложения – это тысячи слов текста.

- **Разъяснение решений и рекомендаций:** Помимо описания “что есть”, мы добавим разделы “Почему сделано так” (причины архитектурных решений) и “Как лучше использовать/расширять”. Например, в архитектурном разделе добавим пояснение, почему выбран Pandera для схем (чем он лучше альтернатив для нашего случая), почему используем ABC вместо протоколов, и т.п. В проектных правилах укажем обоснование соглашений (например, почему именование полей строго snake_case). Эти куски текста не обязательны для использования, но ценны для продвинутых читателей и также увеличивают объём.

- **Многословные пояснения там, где нужно избегнуть двусмысленности:** В сложных местах мы не будем экономить на словах. Если понятие трудно объяснить одной фразой, дадим развернутое определение. Если процесс состоит из нескольких шагов, опишем каждый шаг отдельным абзацем. Цель – **максимально подробная, прозрачная документация**, даже если она станет "толстым мануалом". Пользователь всегда может пропустить знакомые секции, но когда возникнет вопрос – подробное объяснение будет под рукой.

Таким образом, планируемый объём ≥100k слов – это не искусственное раздувание, а естественный результат **комплексного раскрытия темы**. Документация фактически превратится в полноценную книгу/справочник по проекту BioETL. Для сравнения, 100 тыс. слов – это около 300-350 печатных страниц текста. Мы осознаём масштаб и готовы к тому, что это потребует времени и усилий. Зато итоговый продукт будет чрезвычайно полезен сообществу: он закроет большинство возможных вопросов без необходимости лазить в исходный код или догадки.

Конкретные цифры по объёму по разделам: Доменный слой может занять ~20-25k слов (учитывая описание данных, терминов, источников), Прикладной слой – ~30k (архитектура + компоненты + пайплайны + гайды), Инфраструктура – ~15k (окружение, конфиг, логгинг, CI), Интерфейс – ~15k (CLI, примеры, outputs), остальное – правила, глоссарий, повторяющиеся элементы. Эти оценки приблизительны, но демонстрируют распределение. Мы будем следить, чтобы объём шёл от содержания, не размазывать текст искусственно. При необходимости, привлечём дополнительных экспертов (например, доменных специалистов) для наполнения конкретными фактами (это тоже добавит полезный контент).

Также важно организовать такой объём – поэтому разделение на небольшие файлы (как описано в структуре) критично: один файл ~2000-5000 слов легче читается, чем монолит в 100k. Система навигации и индексов не даст потеряться в объёме.

7. Стиль оформления и поддержка документации

Финальная документация будет оформлена в строгом соответствии с требованиями к техническим документам. Мы применим следующие практики для поддержания качества и актуальности:

- **Markdown как основной формат:** Все документы пишутся в Markdown (с расширениями для диаграмм). Этот формат прост для ревью через GitHub и позволяет легко отслеживать изменения. Мы избегаем проприетарных форматов – документация хранится рядом с кодом, под версионным контролем, что обеспечивает её синхронизацию с развитием проекта.

- **Интеграция с Sphinx (при необходимости):** Как упоминалось, при таком объёме выгодно использовать генератор документации. Мы настроим конфигурацию Sphinx для проекта, чтобы он воспринимал `.md` файлы (с помощью MyST) и генерировал из них сайт/PDF. Это позволит, например, автоматически генерировать оглавление и единый глоссарий. Секция "Glossary" может быть помечена как Sphinx glossary, тогда по тексту термины можно будет помечать и получать всплывающие подсказки – удобство для читателя. Также Sphinx поможет поддерживать **сквозные ссылки**: можно использовать рефы по имени секции, что надёжнее, чем хардкодить относительные пути. Мы обязательно опишем в

`README.md` или `docs/README.md`, как собрать документацию локально с помощью Sphinx (т.е. какие команды выполнить).

- **Автоматизация проверок:** В процесс CI включим шаги, проверяющие документацию – хотя бы линтер Markdown (на предмет неправильных ссылок, длины строк, форматирования заголовков). Это не даст случайно влить изменения, ломающие стиль. Возможно, используем инструменты вроде Markdownlint или doc8. Кроме того, можно настроить проверку ссылок (чтобы все указанные внутренние ссылки действительно существуют). Таким образом, поддержка качества документации станет частью процесса разработки.

- **Обновление документации вместе с кодом:** В проектных правилах зафиксируем принцип: *любое изменение кода, влияющее на поведение или интерфейс, должно сопровождаться обновлением документации*. В CONTRIBUTING.md пропишем, что PR должны содержать правки в документацию, если они затрагивают функциональность. Это предотвратит рассинхронизацию кода и описания. Также определим ответственных (например, tech writer или архитектор проекта) за периодический аудит документации – чтобы не допустить устаревания.

- **Версионирование документации:** Если проект будет выпускать версии, мы рассмотрим версионирование документации (например, отдельная ветка или тег для каждой версии, либо использование ReadTheDocs версионности). Пока код не выпущен, документируем "версию 0.x (design)", но в будущем отметим, начиная с какой версии какие функции доступны. Это особенно важно, когда документ описывает планы (на случай, если не всё реализовано одновременно).

- **Лицензирование и ссылки на репозиторий:** В документации явно укажем лицензию проекта (если не указано в README, добавим) и права на сам текст (обычно документация покрывается той же лицензией, либо Creative Commons). Также обеспечим легкую навигацию из документации к коду: например, для классов укажем путь в репозитории, можно даже вставить ссылки на исходники на GitHub. Для этого Sphinx отлично подходит (domain `:py:class:` ссылки). Если решим использовать такие возможности, опишем их в разделе "Как читать документацию".

- **Многоязычность (при необходимости):** Документация пишется на русском, но возможно в будущем потребуется английская версия для широкой аудитории. Структура и формат, которые мы внедряем, заложат основу для перевода. Мы будем по возможности отделять контент от формата (Sphinx это тоже умеет – i18n). Пока это не в приоритете, но документация будет написана достаточно нейтрально, чтобы её можно было при нужде перевести без сильных потерь (избегаем игр слов, жаргона, слишком локальных ссылок).

В заключение, после выполнения всех указанных работ проект **BioactivityDataAcquisition** получит тщательно реорганизованную документацию, понятную и исследователям биоинформатики, и разработчикам, и DevOps-специалистам. Этот технический отчёт охватывает архитектуру по слоям, детали реализации, инструкции по эксплуатации и контекст предметной области. Благодаря структурированности (слои архитектуры) и глубокому содержанию (100k+ слов с примерами, схемами, пояснениями) документация станет надежной основой для развития проекта и обучения новых участников. Мы обеспечим её соответствие лучшим практикам оформления, что повысит читабельность и удобство сопровождения. Реализация данного плана

сделает документацию не просто приложением к коду, а полноценной **научно-технической монографией о системе BioETL**, чем смогут воспользоваться все заинтересованные стороны.

---

1 3 19 33 architecture-overview.md
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/overview/architecture-overview.md

2 glossary.md
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/overview/glossary.md

4 15 25 README.md
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/README.md

5 6 7 8 9 10 11 12 13 14 17 18 22 24 26 27 28 34 35 36 content-audit-report.md
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/content-audit-report.md

16 20 21 41 42 08-clients-diagrams.md
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/02-pipelines/clients/08-clients-diagrams.md

23 37 INDEX.md
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/qc/INDEX.md

29 30 40 pipeline-doc-template.md
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/templates/pipeline-doc-template.md

31 32 43 00-activity-chembl-overview.md
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/02-pipelines/chembl/activity/00-activity-chembl-overview.md

38 39 Markdown — Sphinx documentation
https://www.sphinx-doc.org/en/master/usage/markdown.html