ChatGPT

Обзор архитектуры и ключевых компонентов в репозитории bioactivity_data_acquisition (ветка main)

Общий обзор

Репозиторий предоставляет **BioETL** – каркас для извлечения, обработки и записи биологической активности из различных источников. В нём используется «три-слойная архитектура»:

Слой	Назначение	Компоненты
Orchestration	Отвечает за выбор, запуск и мониторинг пайплайнов. Предоставляет CLI, методы для регистрации/конфигурации и планирования этапов.	Модули `src/bioetl/cli` (регистрация команд, CLI), `src/bioetl/pipelines` (описание пайплайнов), `src/bioetl/public.py` (единый публичный API) [1].
Domain	Содержит чистые функции и схемы, описывающие формат данных. Не работает с сетью или файловой системой.	Схемы Pandera в `src/bioetl/schemas` (например `chembl_activity_schema.py`), преобразователи и типы данных в `src/bioetl/domain` [2].
Infrastructure	Реализует работу с внешними сервисами (HTTP-API ChEMBL, UNPD и др.), кэширование, запись результатов и логирование.	Клиенты в `src/bioetl/clients` и `src/bioetl/core`, система кэширования и хранилища в `src/bioetl/storage` и `src/bioetl/core/http` [3].

Ниже приведено подробное описание ключевых элементов каждого слоя.

1. Слой Orchestration (управление пайплайнами)

Общий API и абстракции

Файл `src/bioetl/public.py` экспортирует основные классы и функции, представляющие стабильный интерфейс для пользователей библиотеки: `UnifiedPipelineBase`, `UnifiedAPIClient`, `UnifiedLogger` и загрузчик конфигурации. Пайплайны регистрируются через функцию `register_pipeline`, чтобы CLI знал, какие команды создавать. Это обеспечивает слабую связанность между определением пайплайна и интерфейсом CLI.

Базовый класс пайплайна

В `src/bioetl/core/pipeline/unified.py` определён абстрактный класс `UnifiedPipelineBase` (наследуется от `PipelineRuntimeBase`). Он задаёт стандартный жизненный цикл ETL-пайплайна:

1. **extract** – извлекает порцию данных;
2. **transform** – преобразует данные;
3. **validate** – проверяет данные с помощью Pandera;
4. **save_results** – записывает данные через инфраструктурные писатели;
5. **build_stage_plan** – конструирует план этапов (extract/transform/validate/save).

Кроме того, `UnifiedPipelineBase` содержит вложенный dataclass `BatchExtractionStats` для статистики извлечения и классы для ЧЕМБЛ-пайплайнов `ChemblPipelineBase` [4], которые выбирают подходящие сервисы и планировщики артефактов.

Контексты и типы

Модуль `src/bioetl/core/pipeline/types.py` определяет множество протоколов и классов, используемых в оркес трации:

- `StageExecutionOptions` – параметры запуска (включить QC, ограничение записей, режим dry-run и др.) [5].
- `StageDescriptor` и `StageCommand` – описания стадий и их обработчиков. План выполняется как последовательность `StageCommand` [6].
- `DataBucket` и `ArtifactStore` – контейнеры для результатов этапов и хранилища артефактов [7].
- Протоколы контекстов (ExecutionContext, ConfigContext, ClientContext и др.), а также адаптер `StageContextAdapter`, объединяющий конфигурационный, клиентский, доменный и инфраструктурный контексты во время выполнения [8]. Благодаря этому каждая стадия получает только необходимые зависимости.

CLI и запуск пайплайнов

CLI реализован в пакете `src/bioetl/cli`. Основные файлы:

- `cli_app.py` – инициализирует приложение Typer, определяет команды `list`, `run-chembl-all` и генерирует отдельные команды для каждого зарегистрированного пайплайна. Команда `run-chembl-all` запускает все ЧЕМБЛ-пайплайны, чьи имена заканчиваются на `_chembl`, с общими опциями для выгрузки и валидации [9] [10].
- `cli_command.py` – предоставляет функцию `create_pipeline_command`, которая создаёт Typer-команду для конкретного пайплайна. Она объединяет профили конфигураций, применяет CLI-переопределения, инстанциирует фабрику пайплайна и обрабатывает исключения с выводом кодов выхода [11].
- `cli_registry.py` – регистрирует доступные пайплайны в глобальном реестре. Для ЧЕМБЛ регистрируются пять основных пайплайнов: `activity`, `assay`, `document`, `target` и `testitem` [12]. Это позволяет динамически создавать команды и централизованно управлять доступностью пайплайнов.

Пример наследования: ЧЕМБЛ-пайплайны

Пайплайны для ChEMBL реализуются в `src/bioetl/pipelines/chembl`. Устаревшая реализация `activity` (файл `run.py`) служит примером работы: класс `ChemblActivityPipeline` расширяет `ChemblCommonPipeline` и реализует методы `build_descriptor`, `resolve_chembl_release`, `prepare_run`, `extract`, `transform`, `validate` и `save_results` [13] [14] [15] [16]. Пайплайн сначала извлекает данные из API ChEMBL, затем применяет трансформации, валидирует через Pandera и сохраняет результаты (CSV, отчёты QC). Хотя этот файл относится к ранней версии, он демонстрирует использование базовых классов и сервисов.

2. Слой Domain (обработка данных)

Схемы и валидация

В пакете `src/bioetl/schemas` определены **Pandera-схемы** для каждого типа данных. Например, файл `chembl_activity_schema.py` объявляет класс `ChEMBLActivitySchema` с 57 строго типизированными колонками и их порядком [2]. Схема используется в `UnifiedPipelineBase.validate` для проверки DataFrame перед сохранением.

Чистые трансформации

Модуль `src/bioetl/domain` содержит чистые функции и преобразователи (трансформеры), работающие исключительно с входным DataFrame. Они не обращаются к внешним сервисам и не записывают файлы; это делает слой легко тестируемым. Архитектура разделяет генерацию артефактов (инфраструктура) от бизнес-логики (домен).

3. Слой Infrastructure (внешние сервисы и хранилища)

Unified API Client и HTTP адаптеры

Пакет `src/bioetl/core/http` предоставляет абстракцию для HTTP-клиентов. Класс `UnifiedAPIClient` объединяет исполнителя запросов, построитель запросов и стратегию пагинации. Он предоставляет методы `get_json`, `paginate_json`, `fetch_one`, `fetch_batch` и др., которые изолируют взаимодействие с конкретными API и возвращают Python-структуры [3] [17]. Для итерации по страницам реализовано `paginate_json`, которое прозрачно подгружает очередные записи до тех пор, пока есть данные [18]. Такой подход позволяет легко переключать реализацию транспорта или добавлять кэширование.

Клиенты ChEMBL и фабрики

В каталоге `src/bioetl/clients/chembl` находятся клиенты для API ChEMBL (например, `BaseChemblClient`, `ChemblEntityClient`). Они реализуют протокол `ApiTransportProtocol` и предоставляют единый интерфейс для получения сущностей. Документация `docs/07-chembl-transport-examples.md` описывает рекомендованную транспортную схему: базовый клиент остаётся «чистым» и может адаптироваться, а класс `UnifiedAPIClient` через `ChemblTransportAdapter` обеспечивает более удобное использование, включая кэширование и стандартизацию ответов [19]. Кэширование достигается дополнительным слоем над клиентом (например, `cache_entity_client`), что позволяет повторно использовать результаты и снижает нагрузку на API.

Хранение артефактов

Модуль `src/bioetl/storage` реализует писатели (`WriteService`, `ChemblWriteService`) и планировщики (ArtifactPlanner), которые сохраняют результаты в CSV, YAML и JSON. В `ChemblCommonPipeline` эти компоненты используются для записи итоговых наборов данных и мета-информации: список колонок, количество строк, время выполнения и отчёты QC [20] [21] [22]. Писатели также генерируют манифест с перечнем файлов, что упрощает загрузку результатов во внешние системы.

Логирование и настройки

Логирование централизовано в `src/bioetl/core/logging.py`, где настраивается `UnifiedLogger` с поддержкой форматирования, контекстных метаданных и динамического контроля уровня логов. Конфигурационные модели (`src/bioetl/core/config/models.py`) позволяют описывать параметры запуска (например, путь хранения, настройки клиента, профили) и загружать их из YAML. Это упрощает переиспользование пайплайнов в разных окружениях.

Общая архитектура и выводы

- **Модульность и расширяемость**. Архитектура BioETL делит систему на независимые слои, что облегчает расширение. Пайплайны наследуются от базового класса, описывают свои `extract` и `transform` и могут регистрироваться для CLI. Добавление нового источника данных требует лишь реализации клиента и схемы.

- **Инверсия зависимостей через контексты**. Использование протоколов и адаптеров (ExecutionContext, ArtifactContext) позволяет передавать только необходимые зависимости на каждом этапе. Это снижает связанность и упрощает тестирование.

- **Чистота доменного слоя**. Всё взаимодействие с внешним миром вынесено в инфраструктуру, а доменный слой содержит только чистые преобразования. Это позволяет повторно использовать функции и схемы независимо от транспорта.

- **Удобный CLI**. Реализация через Typer обеспечивает автогенерацию команд и аргументов. Профили конфигурации и переопределения позволяют гибко настраивать запуск без изменения кода.

- **Сфокусированность на ChEMBL**. Хотя каркас можно расширять, текущая реализация ориентирована на загрузку данных ChEMBL. Существуют различные пайплайны (activity, assay, document, target, testitem), которые используют общие базовые классы и сервисы. Документация и примеры помогают разобраться с подключением, кэшированием и сохранением результатов.

В целом `bioactivity_data_acquisition` реализует современный ETL-каркас с разделением обязанностей между оркестрацией, доменом и инфраструктурой. Пайплайны легко расширять, а применение Pandera и Typer позволяет обеспечить валидацию и удобный CLI.

---

[1] README.md
https://github.com/SatoryKono/bioactivity_data_acquisition/blob/main/README.md

[2] chembl_activity_schema.py
https://github.com/SatoryKono/bioactivity_data_acquisition/blob/main/src/bioetl/schemas/chembl_activity_schema.py

[3] [17] [18] api_client.py
https://github.com/SatoryKono/bioactivity_data_acquisition/blob/main/src/bioetl/core/http/api_client.py

[4] unified.py
https://github.com/SatoryKono/bioactivity_data_acquisition/blob/main/src/bioetl/core/pipeline/unified.py

[5] [6] [7] [8] types.py
https://github.com/SatoryKono/bioactivity_data_acquisition/blob/main/src/bioetl/core/pipeline/types.py

[9] [10] cli_app.py
https://github.com/SatoryKono/bioactivity_data_acquisition/blob/main/src/bioetl/cli/cli_app.py

[11] cli_command.py
https://github.com/SatoryKono/bioactivity_data_acquisition/blob/main/src/bioetl/cli/cli_command.py

[12] cli_registry.py
https://github.com/SatoryKono/bioactivity_data_acquisition/blob/main/src/bioetl/cli/cli_registry.py

[13] [14] [15] [16] run.py
https://github.com/SatoryKono/bioactivity_data_acquisition/blob/main/src/bioetl/pipelines/chembl/activity/run.py

[19] 07-chembl-transport-examples.md
https://github.com/SatoryKono/bioactivity_data_acquisition/blob/main/docs/07-chembl-transport-examples.md

[20] [21] [22] base.py
https://github.com/SatoryKono/bioactivity_data_acquisition/blob/main/src/bioetl/pipelines/chembl/common/base.py