

MVP — техническое описание (внутренний документ)

Дата: 03.11.2025

Таймзоны: админка — Europe/Amsterdam; публикации — Europe/Moscow (MSK)

1) Цель MVP (самый простой позитивный кейс)

Задача: автоматически взять одну свежую статью из одного источника (начать с RSS arXiv или eLife/Nature RSS), извлечь метаданные и текст/аннотацию, сгенерировать краткое summary «простыми словами» (≤ 1000 символов), сформировать **черновик** поста и отправить его владельцу в Telegram DM на апрув. Утверждённые черновики публикуются в канал **в 11:00 MSK**. Если черновик не утверждён — в этот день не постим.

Ограничения/правила MVP: - Обход источников **не чаще 1 раза в сутки**, ночное окно (03:00–05:00 MSK). При сбое — **1 ретрай** через 5–10 часов. - Формат поста: заголовок, summary (≤ 1000 симв.), затем метаданные вне лимита: ссылка на оригинал (обязательно), **DOI/журнал/год только при наличии DOI**, авторы (до 3, затем *et al.*). Без эмодзи/хэштегов.
- Все публикации — через **черновики**: апрув обязателен. Апрув до **11:00 MSK** → пост сегодня в 11:00; позже — перенос на следующий день (11:00).

2) Технологический стек и ключевые библиотеки

Язык: TypeScript / Node.js LTS

Хранилище: PostgreSQL (метаданные), локальное файловое хранилище (PDF/артефакты), Redis (кэш), RabbitMQ (очереди).

Контейнеризация: Docker + Docker Compose (без Kubernetes)

Служебные библиотеки (предложение): - HTTP/загрузка: `undici` или `got`; ретрай — `retry`/встроенные. - RSS/Atom: `rss-parser`. - HTML-парсинг: `cheerio`. - PDF-текст: `pdf-parse` (OCR выключен по умолчанию).
- Нормализация/язык: `franc` (detector) + простая трансформация (EN→RU при публикации — опционально, флагом).
- Дедупликация: `simhash`/`minhash` (например, `simhash-ts`) + ключи на DOI/заголовок+авторы. - БД: `Prisma` ORM (альт.: TypeORM). Миграции — `prisma migrate`. - Очереди: `amqplib` (или `rascal`/Nest RMQ transporter).
- Планировщик: `node-cron` (упрощённо) либо отдельный воркер-сервис с CRON. - Логирование: `pino` + JSON-логи. - Метрики: `prom-client` (Prometheus), Grafana дашборды. - Rate limiting/backoff: `bottleneck`. - Telegram Bot API: `telegraf` (или чистый `node-telegram-bot-api`). - Валидация схем/DTO: `zod`. - LLM API: `openai` (официальный JS SDK).

Суммаризация (по умолчанию — OpenAI API): - Отдельный сервис **Summarizer** обращается к **OpenAI API** (модель задаётся через `OPENAI_MODEL`; по умолчанию — экономичная современная модель). Жёсткий промпт и строгая валидация длины ≤ 1000 символов «простым языком». -

Альтернатива (после MVP): локальный провайдер через `Ollama` (Llama/Mistral и др.) — через адаптер, переключение флагом `PROVIDER=openai|ollama`.

3) Архитектура MVP (модули/сервисы)

Монорепо с несколькими сервисами (каждый — контейнер):

1. **api-gateway** (Fastify/NestJS):

2. REST/JSON для админки: источники, черновики, статус задач.

3. Авторизация Basic Auth (для пет-проекта).

4. **scheduler**:

5. CRON-задачи на ежедневный обход (03:00–05:00 MSK, с джиттером).

6. Планировщик публикаций (ежедневно 11:00 MSK): выбор подтверждённых черновиков за предыдущее календарное сутки.

7. **crawler**:

8. Читает реестр источников, тянет RSS/API/HTML-листы.

9. Находит новые записи → публикует задания `fetch.request` в RMQ.

10. **fetcher**:

11. По URL/API скачивает HTML/PDF/JSON, сохраняет файлы в `/data/articles/<articleId>/...`.

12. Отдаёт сырой контент в очередь `parse.request`.

13. **parser**:

14. Извлекает метаданные (title, authors, DOI, journal, date, license, url, language) и основной текст/аннотацию.

15. Записывает `Article`, `ArticleText` в PostgreSQL. Публикует `dedup.request`.

16. **deduplicator**:

17. Проверяет DOI/ключи, вычисляет текстовый фингерпринт (SimHash/MinHash).

18. Если дубль — прекращает обработку; если новый — `summarize.request`.

19. **summarizer** (HTTP→LLM):

20. Вызывает **OpenAI API** с заданной моделью (`OPENAI_MODEL`), контролирует токены и длину итогового текста (≤ 1000 симв.).

21. Помечает `needs_review` при низкой уверенности/неполноте, сохраняет `Summary` в БД, публикует `publication.draft.request`.

22. **publisher:**

23. Формирует шаблон поста (заголовок, summary, метаданные вне лимита).

24. Отправляет **черновик** владельцу в Telegram DM (preview): кнопки «Утвердить/Отклонить/Редактировать».

25. При апруве — помечает `Publication` как `approved` для планировщика.

26. В 11:00 MSK — отправка в канал + анти-флуд интервал 2-10 сек.

27. **admin-ui** (простая SPA + API):

28. Список источников/черновиков, кнопки модерации, поиск по DOI/авторам/названию/источнику/датам.

29. **infra:**

- PostgreSQL, RabbitMQ, Redis, Grafana, Prometheus (экспортёр), Nginx (reverse proxy + Basic Auth).

Схема обмена (RMQ): - Exchange `pipeline` (type: topic).

- Routing keys: `fetch.request`, `parse.request`, `dedup.request`, `summarize.request`, `publication.draft.request`, `publication.approved`, `publication.schedule`.

4) Минимальная модель данных (Prisma-стилистика)

- **Source:** id, name, type (rss|api|html), baseUrl, schedule, rateLimit, enabled, priority, defaultTags(json).
 - **Article:** id, externalId (doi|arxiv|pmid), title, url, fulltextUrl, journal, year, license, language, authors(json[3]), createdAt.
 - **ArticleText:** articleId(fk), rawText, cleanedText, ocrUsed(bool).
 - **Summary:** id, articleId(fk), text, lengthType(enum: short), tone(enum: plain), confidence(float), tags(json), warnings(json), createdBy(model/version), createdAt.
 - **Publication:** id, summaryId(fk), channelId, status(enum: draft|approved|scheduled|sent|failed|cancelled), dmPreviewMessageId, approvedBy, approvedAt, scheduledAt, sentAt, retryCount.
-

5) Флоу (последовательность шагов)

1. **Scheduler** публикует `crawl.start` (по каждому включённому источнику).
2. **Crawler** читает RSS/API → новые записи → `fetch.request(url, sourceId)`.
3. **Fetcher** скачивает HTML/PDF → сохраняет → `parse.request(articleId, paths)`.
4. **Parser** извлекает метаданные/текст → пишет в БД → `dedup.request(articleId)`.
5. **Deduplicator** решает: дубль? если нет → `summarize.request(articleId)`.

6. **Summarizer** вызывает локальный LLM → пишет `Summary` → `publication.draft.request(summaryId)`.
7. **Publisher** формирует и отправляет **чертёж** в DM владельцу.
8. **Owner** нажимает «Утвердить» → статус `approved`.
9. **Scheduler** в 11:00 MSK выбирает все `approved` за предыдущее календарное сутки → отправляет в канал (интервал 2–10 сек) → статус `sent`.

Ошибки/повторы: сетевые ошибки → автоматические ретрай с backoff; при падении суточного обхода — один повтор через 5–10 часов.

6) Шаблон поста в канал (MVP)

```
<Заголовок>

<Summary – до 1000 символов, простыми словами>

Оригинал: <url>
Авторы (до 3): <Фамилия И., Фамилия И., Фамилия И., et al.>
DOI/Журнал/Год: <только при наличии DOI>
```

Примечания: - Для препринтов (bioRxiv/arXiv) добавить дисклеймер «preprint, без peer-review».
- Теги не выводим (используются только для внутреннего поиска/фильтров).

7) Развёртывание (Docker Compose)

Сервисы: `api-gateway`, `scheduler`, `crawler`, `fetcher`, `parser`, `deduplicator`,
`summarizer`, `publisher`, `admin-ui`, `postgres`, `rabbitmq`, `redis`, `grafana`,
`prometheus`, `nginx`.

ENV (минимум): - `DATABASE_URL=postgresql://...` - `RABBITMQ_URL=amqp://...` -
`REDIS_URL=redis://...` - `BOT_TOKEN=...` - `OWNER_TELEGRAM_USER_ID=...` -
`CHANNEL_CHAT_ID=...` - `TIMEZONE_PUBLICATION=Europe/Moscow` -
`CRAWL_WINDOW=03:00-05:00` - `OPENAI_API_KEY=sk-...` - `OPENAI_MODEL=...` (например,
экономичная модель для сводок) - `PROVIDER=openai` # для будущего переключения на `ollama`

Файлы статей: `./data/articles` (монтируется в несколько сервисов с `ro` / `rw` где нужно).

8) Мониторинг/логи

- Экспорт метрик по сервисам: время этапов, проценты удач, длина summary, ошибки по типам.
 - Графики в Grafana.
 - JSON-логи с `traceId` и `articleId` по всему пайплайну.
-

9) Критерии готовности MVP

- Поддержан минимум **1** источник (arXiv RSS) + конфиг ещё **2** (например, Nature RSS, PubMed по ключу).
 - $\geq 90\%$ корректное извлечение базовых метаданных на тестовой выборке.
 - «С конца в конец»: от обнаружения до отправки **черновика** в DM и публикации в канал по расписанию 11:00 MSK.
 - Докеризировано; конфиги через `.env`; резервные копии БД ежедневно.
-

10) Нюансы/потенциальные риски

- Парсинг PDF нестабилен: держим фолбэк на аннотацию/абстракт.
 - Лицензии: публикуем только summary + ссылку, без цитирования полного текста.
 - Качество LLM: жёсткий промпт, пост-проверка длины/запрет жаргона; пометка `needs_review` при сомнениях.
 - При использовании OpenAI API: учитывать стоимость и лимиты, хранить ключ в секрете; не отправлять в API лишние персональные/лицензионно чувствительные данные; обработка ошибок/ретраев по кодам и rate-limitам.
-

11) Беклог после MVP (R2/R3 — кратко)

- OCR (включаем флагом), авто-теги, многоязычие, дополнительные источники, EN-канал, улучшенный UI модерации, анти-галлюцинационные проверки, дедуп по нескольким сигналам, алерты.