Ниже — «правильная» структура под текущую архитектуру + по каждому узлу/файлу: назначение, процент готовности и что доделать/исправить. Я опираюсь на содержимое архива **BACKMESSAGER** (монорепо на PNPM, Fastify-API, Realtime WS, воркеры, packages, modules, миграции, docker-compose).

* **1) Корень репозитория**

.

├─ package.json

├─ pnpm-lock.yaml

├─ tsconfig.json

├─ .gitignore

├─ .env.example

├─ README.md ← (нужен)

├─ .eslintrc.json ← (нужен)

├─ .prettierrc ← (нужен)

├─ apps/

├─ packages/

├─ modules/

├─ migrations/

└─ infra/

* **package.json** — монорепо (PNPM workspaces), скрипты сборки.  
  **Готовность:** 80%  
  **Что сделать:** Расширить workspaces до ["apps/\*","packages/\*","modules/\*"]; поправить build/dev/start фильтры на ./apps/\* (сейчас у вас встречается server/apps/\*).
* **pnpm-lock.yaml / tsconfig.json / .gitignore** — ок.  
  **Готовность:** 100%
* **.env.example** — dev-значения.  
  **Готовность:** 85%  
  **Что сделать:** пометить критичные переменные (JWT, S3) как *пример*, добавить секции для METRICS\_IPS, DOCS\_IPS, CORS\_ALLOWLIST.
* **README.md** — отсутствует.  
  **Готовность:** 0%  
  **Что сделать:** инструкции запуска (dev/prod), перечень сервисов, переменные, миграции, схема API.
* **.eslintrc.json / .prettierrc** — отсутствуют.  
  **Готовность:** 0%  
  **Что сделать:** добавить конфиги + скрипты lint, format, pre-commit (husky/lint-staged).
* **2) apps/api (Fastify 4, TS)**

apps/api/

├─ package.json

├─ tsconfig.json

├─ Dockerfile

└─ src/

├─ main.ts

├─ plugins/

│ ├─ openapi.ts

│ ├─ ratelimit.ts

│ ├─ security.ts

│ └─ validation.ts

├─ middleware/

│ ├─ auth.guard.ts

│ ├─ role.guard.ts

│ └─ idempotency.ts

└─ routes/

└─ v1/

├─ index.ts

├─ auth.route.ts

├─ users.route.ts

├─ uploads.route.ts

├─ media.route.ts

├─ conversations.route.ts

├─ messages.route.ts

├─ channels.route.ts

├─ posts.route.ts

└─ search.route.ts

* **apps/api/package.json** — зависимости Fastify (@fastify/helmet, cors, rate-limit, swagger, zod и т.д.).  
  **Готовность:** 95%
* **src/main.ts** — инициализация Fastify, регистрация плагинов, роутов, graceful shutdown.  
  **Готовность:** 85%  
  **Что сделать:**
  + Действительно **подключить валидацию Zod**:  
    withTypeProvider<ZodTypeProvider>(), setValidatorCompiler(validatorCompiler), setSerializerCompiler(serializerCompiler).
  + Ограничить доступ к /metrics и /docs (см. security ниже).
  + Вынести CORS\_ALLOWLIST в env и запретить \* с credentials:true.
* **plugins/openapi.ts** — Swagger + zod-to-json-schema, генерация /docs и /openapi.json.  
  **Готовность:** 75%  
  **Что сделать:**
  + Привести **все роуты** к единому формату schema.response (контракты ответов), чтобы спецификация была полной.
  + Добавить глобальный формат ошибок (error.code/message/details).
* **plugins/ratelimit.ts** — per-route лимиты через Redis.  
  **Готовность:** 90%
* **plugins/security.ts** — CORS allowlist, CSP nonce/headers, HSTS, X-CTO, XFO.  
  **Готовность:** 85%  
  **Что сделать:**
  + Явно указать **список разрешённых origin** из ENV (строки/RegExp).
  + Проверить отсутствие origin="\*" при credentials:true.
  + При необходимости включить @fastify/helmet (если не дублируете заголовки вручную).
* **plugins/validation.ts** — сейчас заглушка (проброс схемы «как есть»).  
  **Готовность:** 40%  
  **Что сделать:** заменить на реальный компилятор Zod (см. main.ts).
* **middleware/auth.guard.ts** — проверка JWT с jose, извлечение uid/jti/roles/scope, 401 на ошибках.  
  **Готовность:** 90%  
  **Что сделать:** при необходимости добавьте проверку aud/iss, rbac-решение через роли/скоупы.
* **middleware/role.guard.ts** — привязка RBAC/ABAC (есть, не проверяли логику).  
  **Готовность:** 80%  
  **Что сделать:** унифицировать список ролей и описать в README.
* **middleware/idempotency.ts** — полноценный Idempotency-Key с Redis-lock, TTL, maxStoreBytes.  
  **Готовность:** 90%
* **routes/v1/\*.route.ts** — роуты.
  + **index.ts** — агрегатор v1. **Готовность:** 95%
  + **auth.route.ts** — регистрация/логин/refresh/logout. **Готовность:** 85%
    - **TODO:** единый формат ошибок/ответов, Zod-схемы ответов (не только body).
  + **users.route.ts** — профили/аккаунты. **Готовность:** 80%
    - **TODO:** response-схемы, пагинация cursor, фильтры.
  + **uploads.route.ts** — S3 multipart, complete с идемпотентностью; публикация задач. **Готовность:** 70%
    - **TODO критично:** явные **пределы размера** (sizeBytes upper bound), whitelists **MIME** и расширений; чёткий TTL presigned URL; запрет overwrite; валидация всех входных полей; audit-логи.
  + **media.route.ts** — работа с медиа/метаданными. **Готовность:** 75%
    - **TODO:** отдача только после antivirus=clean; пагинация/фильтры; кэш-заголовки.
  + **conversations/messages/channels/posts/search** — CRUD/листинги. **Готовность:** 70–80%
    - **TODO:** response-схемы, единый формат ошибок, курсорная пагинация.
* **Dockerfile** — multi-stage, USER app, prod deps только.  
  **Готовность:** 95%
* **3) apps/realtime (WebSocket шлюз)**

apps/realtime/

├─ package.json

├─ Dockerfile

└─ src/

└─ index.ts

* **src/index.ts** — WS-шлюз Fastify + @fastify/websocket, Redis Pub/Sub, авторизация по JWT/HMAC, лимит подписок, heartbeat.  
  **Готовность:** 80%  
  **Что сделать:**
  + Перейти с jsonwebtoken на **jose** (унификация с API).
  + Вынести правила подписки (ACL по топикам), лимит активных подписок через ENV.
  + Rate-limit на subscribe событиях, защита от флуд-подписок.
* **Dockerfile** — ок.  
  **Готовность:** 95%
* **4) apps/workers/\* (фоновые воркеры)**

apps/workers/

├─ video-transcode/

│ ├─ Dockerfile

│ └─ src/index.ts

├─ image-variants/

│ ├─ Dockerfile

│ └─ src/index.ts

├─ search-indexer/

│ ├─ Dockerfile

│ └─ src/index.ts

├─ notify/

│ ├─ Dockerfile

│ └─ src/index.ts

└─ media-gc/

├─ Dockerfile

└─ src/index.ts

* **video-transcode** — ffmpeg → HLS, проверяет antivirus=clean и quarantined=false.  
  **Готовность:** 85%  
  **Что сделать:** idempotency/ретраи, лимиты CPU/IO, fallback-профили (360→480→720→1080), параметризация CRF/preset; контроль диска (tmpdir) и очистка.
* **image-variants** — sharp, генерация вариантов.  
  **Готовность:** 85%  
  **Что сделать:** лимиты по входному размеру, ретраи, профили (webp/avif), метрики.
* **search-indexer / notify / media-gc** — общая заготовка.  
  **Готовность:** 70–80%  
  **Что сделать:** контракт событий (Redis Streams), отложенные ретраи (DLQ), строгие схемы событий (zod), метрики по операциям.
* **5) packages/\* (инфраструктура/шаред-коды)**

packages/

├─ config/ # env, численные/булевы парсеры, лимиты для воркеров

├─ db/ # postgres client (prepare:true), транзакции, closeDb

├─ redis/ # ioredis, xaddJson/parseEntryJson, init/quit

├─ s3/ # AWS SDK v3, обёртки (из архива — есть)

├─ logger/ # pino/… (есть)

├─ observability/ # Prometheus registry, /metrics сервер, таймеры

└─ validation/ # zod-схемы, типы, PaginationQuery, Cursor

* **config** — отличная типизация настроек приложения/воркеров.  
  **Готовность:** 90%  
  **Что сделать:** добавить METRICS\_IPS, DOCS\_IPS, CORS\_ALLOWLIST, значения по умолчанию и проверки.
* **db** — postgres с подготовленными запросами, транзакции.  
  **Готовность:** 90%
* **redis** — init, pub/sub, Streams утилиты.  
  **Готовность:** 90%
* **s3** — не детализировали, но присутствует.  
  **Готовность:** 85%  
  **Что сделать:** presign с TTL из ENV, строгие ACL, запрет публичных бакетов по умолчанию.
* **observability** — registry, хендлер /metrics, тайминги postgres/redis/http.  
  **Готовность:** 90%  
  **Что сделать:** экспорт лейблов app/service, быстрый healthz/readyz в API, **ограничить доступ к /metrics**.
* **validation** — базовые zod-схемы (Id, Handle, PaginationQuery).  
  **Готовность:** 90%  
  **Что сделать:** общий **ErrorSchema**, **ApiResponse<T>**, Cursor-пагинация (request/response), common enums.
* **logger** — есть.  
  **Готовность:** 80%  
  **Что сделать:** единая конфигурация pino (json, level per env), маппинг полей ошибок.
* **6) modules/\* (доменная логика)**

modules/

├─ auth/

│ └─ src/index.ts # Argon2id, SignJWT, refresh-сессии (SHA-256), ротация

├─ users/

├─ posts/

├─ uploads/

├─ conversations/

├─ messages/

├─ channels/

└─ search/

* **auth** — качественно: Argon2id, refresh opaque + SHA-256, ротация/отзыв.  
  **Готовность:** 90%  
  **Что сделать:** лимиты попыток логина (Redis), 2FA при необходимости.
* **uploads** — есть модуль modules/uploads/src/index.ts (инициация/parts/complete/abort, очереди).  
  **Готовность:** 80%  
  **Что сделать:** явные лимиты maxInputMB, фильтры MIME/расширений, тонкий аудит.
* **остальные модули** — каркас/роуты на месте, код домена частично упомянут.  
  **Готовность:** 60–75%  
  **Что сделать:** описать **use-cases** (интерфейсы), схему данных, индексы, транзакционные границы, in/out-схемы (zod).
* **7) migrations/ (PostgreSQL)**

migrations/

├─ 001\_init.sql

├─ 002\_auth\_sessions.sql

├─ 003\_media.sql

├─ 004\_conversations.sql

├─ 005\_messages.sql

├─ 006\_receipts.sql

└─ ... (ещё ~12 файлов всего)

* **Сущности**: users, auth\_sessions (хэш refresh, replaced\_by, expires\_at), media\_\*, conversations/messages, etc.  
  **Готовность:** 85%  
  **Что сделать:**
  + Индексы для горячих запросов (пример: (conversation\_id, created\_at), media\_variants(media\_id, profile), files(sha256)), ON DELETE CASCADE где требуется.
  + Ограничения целостности (CHECKs) и дефолты.
* **8) infra/ (docker-compose)**

infra/

└─ docker-compose.yml

* Поднимает: Postgres, Redis, MinIO, ClamAV, Nginx, API/воркеры/WS; healthchecks; конфигурация USER app.  
  **Готовность:** 90%  
  **Что сделать:** лимиты по ресурсам (CPU/mem) для воркеров, тома для MinIO, сеть/ingress-правила, прокси-кеш Nginx для статичных GET.
* **9) Поперечные темы (что обязательно довести)**
* **Валидация Zod «на входе и на выходе»** — сейчас частично; довести до 100% во всех роут-хэндлерах schema.response.  
  **Статус:** 50% → **Нужно довести до 100%**.
* **Периметр безопасности** — CORS allowlist (без \*+credentials), /metrics и /docs под IP/VPN.  
  **Статус:** 70% → **Нужно довести до 100%**.
* **Единый формат ошибок и пагинации** — error.code/message/details, cursor-пагинация (next\_cursor).  
  **Статус:** 60% → **Нужно довести до 100%**.
* **JWT унификация** — jose и в WS, общие iss/aud, ротация ключей/секретов.  
  **Статус:** 80% → **Нужно довести до 100%**.
* **Uploads безопасность** — лимиты размера, явный белый список MIME/расширений, TTL presign, запрет overwrite, журналирование.  
  **Статус:** 65% → **Нужно довести до 100%**.
* **Developer-опыт** — .eslintrc, .prettierrc, README, скрипты lint|format|typecheck|test.  
  **Статус:** 20% → **Нужно довести до 100%**.
* **Тесты** — unit/integration/contract (OpenAPI).  
  **Статус:** 10% → **Нужно довести до 100%**.
* **10) Быстрый план доведения до «идеала» (с приоритетами)**
* **(P0) Включить Zod-валидацию** в apps/api/src/main.ts и пройтись по всем роутам: дописать schema.response.
* **(P0) Перекрыть /metrics и /docs** по IP/VPN, навести порядок в CORS.
* **(P0) Ужесточить uploads**: maxInputMB, MIME/extension whitelist, TTL presign, audit.
* **(P1) Унификация JWT на jose** и в Realtime; добавить ротацию ключей.
* **(P1) Workspaces**: починить workspaces и фильтры сборки.
* **(P2) Добавить ESLint/Prettier/README**, единый формат ошибок и cursor-пагинацию.
* **(P2) Тесты**: auth-refresh, uploads-idem, media-pipeline, search-index.