Detailed plan to finalize the Meta + LinkedIn MVP

1) Security and configuration

- Goals
  - Centralized secrets, safer auth, strict input validation.
- Key tasks
  - Integrate AWS Secrets Manager or Vault for provider secrets and `JWT_SECRET`.
  - Enforce JSON schema validation on all POST bodies.
  - Add API key rotation and JWT expiry checks.
- Where
  - `infra/docker/docker-compose.yml` (env sourcing), `apps/api/src/*`, `packages/config/env.ts`.
- Acceptance
  - No secrets in env files or code; all POST endpoints reject invalid inputs; rotation runbook.

2) OAuth, state, and account targeting

- Goals
  - Secure and predictable OAuth with explicit page/company selection.
- Key tasks
  - Persist CSRF state/nonce in Redis; verify on callback.
  - Add multi-account selection: persist chosen Facebook Page/IG account/LinkedIn company per schedule.
  - Normalize callbacks in `oauth.controller.ts` and `platform.controller.ts`.
- Where

    - `apps/api/src/oauth.controller.ts`, `apps/api/src/platform.controller.ts`, Redis use in `TokenService`.
- Acceptance
    - Callbacks reject invalid state; user selects target page/company; target stored with `schedules`.

3) Token lifecycle hardening

- Goals
    - Reliable refreshes, full audit visibility, safe cache behavior.
- Key tasks
    - Expand audit in `TokenService` and surface via an endpoint.
    - Backoff/jitter on refresh failures; alert after N failures/hour.
    - Invalidate caches on disconnect; handle revoked scopes gracefully.
- Where
    - `apps/api/src/token.service.ts`, `DbService` audit writes, new `GET /tokens/audit` route.
- Acceptance
    - Token refresh SLOs met; audits queryable; revocation leads to user-facing actionable errors.

4) Content workflow guardrails and permissions

- Goals
    - Prevent invalid schedules and enforce roles cleanly.
- Key tasks
    - Validate media requirements by platform (e.g., IG needs media) on schedule.
    - Expand `RolesGuard` to enforce editor vs admin actions consistently.
    - Add richer IG presets validation (story/reel/carousel) in `ContentAdaptationService`.
- Where

    - `apps/api/src/content.controller.ts`, `apps/api/src/guards/roles.guard.ts`, `apps/api/src/content-adaptation.service.ts`.
- Acceptance
    - Invalid content cannot be scheduled; permissions enforced on all content endpoints.

5) Scheduling and jobs reliability

- Goals
    - Full control over schedules and resilient job execution.
- Key tasks
    - Add reschedule/cancel endpoints and state transitions.
    - Generate and enforce idempotency keys per schedule on worker side.
    - Introduce provider-aware retry configs (max attempts, jitter, cooldown).
- Where
    - `apps/api/src/content.controller.ts`, `apps/api/src/queue.service.ts`, `apps/worker/worker.js`.
- Acceptance
    - Schedules can be rescheduled/cancelled; no duplicate publishes; retries match provider limits.

6) Publishing robustness and targeting

- Goals
    - Explicit target account and stronger media handling.
- Key tasks
    - Persist target page/company on schedule; propagate to `PlatformPublishService`.
    - Add MIME detection, size limits, and large media handling for images.
    - Normalize error messages; map provider codes to actionable errors.
- Where

    - `apps/api/src/platform-publish.service.ts`, `apps/api/src/db.service.ts`, `apps/api/src/services/media-processing.service.ts`.
- Acceptance
    - Publishing goes to the selected account; invalid media is blocked early; errors are user-readable.

7) Webhook processing and DLQ

- Goals
    - Action useful events; robust failure handling and replay.
- Key tasks
    - Implement processors: comments/messages/status updates (Meta, LinkedIn).
    - Add DLQ management endpoints and Grafana panel for DLQ depth.
    - Emit metrics for failures and signature mismatches.
- Where
    - `apps/worker/worker.js`, `apps/api/src/webhooks.controller.ts`, `apps/api/src/db.service.ts`.
- Acceptance
    - Events drive stored updates; DLQ visible and replayable; signature failures alert.

8) Analytics and insights (MVP scope)

- Goals
    - Basic insights surfaced for posts/pages/companies.
- Key tasks
    - Add API routes to fetch and cache Meta page insights and LinkedIn post/company stats.
    - Store snapshots daily for simple time series.
- Where

    - `apps/api/src/platforms/meta/meta-client.service.ts`, `apps/api/src/platforms/linkedin/linkedin-client.service.ts`, `apps/api/src/platform.controller.ts`, `DbService`.
- Acceptance
    - API returns reach/engagement metrics; basic daily charts possible.

9) Observability and metrics

- Goals
    - Traceable requests and measurable worker health.
- Key tasks
    - JSON structured logs with correlation IDs in API and Worker.
    - Prometheus metrics: queue depth, job latency, success/failure, token refresh outcomes.
    - Grafana dashboards + alerts (job failure rate, token refresh failures, 429 spikes).
- Where
    - `apps/api/src/main.ts` (logger), `apps/worker/worker.js`, `infra/observability/*`.
- Acceptance
    - Dashboards online; alerts fire on defined thresholds; logs searchable in ELK.

10) Multi-tenancy safety

- Goals
    - Remove `org_chauncey`, enforce tenant isolation.
- Key tasks
    - Extract `organizationId` from JWT context; thread through controllers/services.
    - Add tenant filters to DB queries; prevent cross-tenant access.
- Where
    - `apps/api/src/*controller.ts`, `apps/api/src/db.service.ts`, `apps/api/src/guards/gateway-auth.guard.ts`.

- Acceptance
    - All queries scoped by tenant; no hardcoded org; tests for tenant isolation pass.

11) Rate limiting and abuse prevention

- Goals
    - Fair usage per tenant and per route.
- Key tasks
    - Rate limit keys include tenant and auth type; expand route-specific limits for publish/webhooks.
    - Circuit breaker around provider calls when sustained 429/5xx occur.
- Where
    - `apps/api/src/middleware/rate-limit.middleware.ts`, `apps/api/src/platform-publish.service.ts`.
- Acceptance
    - Tenant-scoped limits; circuit breaker opens on provider saturation and recovers with backoff.

12) CI/CD and quality gates

- Goals
    - Safer merges and deploys.
- Key tasks
    - CI: lint, type-check, tests, secret scanning, Docker build.
    - E2E tests: OAuth mocks, publish path, webhook signature, schedule idempotency.
    - Load tests for publish bursts, webhook storms.
- Where
    - `infra/ci/github-actions.yml`, `apps/api/__tests__/*`, `test/api-integration-test.js`.
- Acceptance

    - CI must pass to merge; core flows covered by E2E; performance SLOs documented.

13) Documentation and runbooks

- Goals
    - Clear ops and user workflows.
- Key tasks
    - Connect → approve → schedule → publish guide with screenshots.
    - Token troubleshooting, rate-limit handling, DLQ replay runbook.
    - Update Postman collection and envs.
- Where
    - `doc/runbook.md`, `doc/diagrams.md`, `doc/postman-collection.json`.
- Acceptance
    - New engineer can operate flows and recover incidents using docs alone.
- Suggested sequence
    - Week 1: Security/config, OAuth/state, token lifecycle; multi-tenancy scoping.
    - Week 2: Content guardrails, scheduling controls, publishing robustness, webhooks processors.
    - Week 3: Analytics wiring, observability and dashboards, rate limiting & circuit breakers.
    - Week 4: CI/CD gates, tests, documentation, polish and UAT.
- Fast wins to tackle first
    - Remove hardcoded `org_chauncey`; thread `organizationId` from JWT.
    - Enforce state/nonce storage and validation in OAuth callbacks.
    - Add schedule target account fields and enforce on publish.
    - Add JSON schema validation for `content`, `schedule`, and `posts` endpoints.

- Emit Prometheus metrics for queue/job outcomes and token refreshes.