Architecture · Ai

Generated Fri, Nov 6, 2026 1:56 PM · Single page

docs/architecture/ai.md

AI foundation

Provider-agnostic AI layer for Algos CRM. Phase 0 is infrastructure. Phase 1 is Inbox/Lead copilot. Phase 2 is Ask Algos natural-language search. Do not implement Reports AI or Task AI here.

Architecture

```
Controller / future feature job
        ?
        ?
AIContextBuilder   ? tenancy + existing policies
        ?
        ?
PromptAssembler    ? system/user prompts (not in controllers)
        ?
        ?
AIService          ? never throws to callers
        ?
        ??? AIProviderInterface  (OpenAI today, swap later)
        ??? AIResponseParser
        ??? AIUsageLogger ? ai_usage_logs
```

Piece	Path
Config	`config/ai.php`
Service	`app/Services/AI/AIService.php`
Provider contract	`app/Services/AI/Contracts/AIProviderInterface.php`
OpenAI adapter	`app/Services/AI/Providers/OpenAIProvider.php`
Null adapter	`app/Services/AI/Providers/NullAIProvider.php`
Context / authz	`app/Services/AI/Context/AIContextBuilder.php`
Prompts	`app/Services/AI/Prompts/PromptAssembler.php`
Parser	`app/Services/AI/AIResponseParser.php`
Usage log model	`app/Models/AiUsageLog.php`
Queue job	`app/Jobs/AI/ProcessAIRequestJob.php`
Provider registration	`app/Providers/AIServiceProvider.php`

`AIService` always returns an `AIOutcome`. Provider timeouts, HTTP errors, missing keys, and malformed JSON are caught, logged, and turned into a generic user message. CRM screens keep working.

Phase 1 — Sales copilot

User-requested actions only (never on page load):

Action	Route	Permission	Sends messages?

Inbox summary	`POST inbox.ai.summary`	`view.inbox`	No
Suggested reply	`POST inbox.ai.reply`	`reply.inbox`	No — fills a draft only
Lead summary	`POST leads.ai.summary`	`view` on that lead	No
Lead score	`POST leads.ai.score`	`view` on that lead	No
Next best action	`POST leads.ai.next-action`	`view` on that lead	No
Follow-up draft	`POST leads.ai.follow-up`	`view` on that lead	No

Results are cached on `ai_assist_results`. Lead score is also stored on `leads.ai_score` / `ai_score_generated_at` / `ai_score_model` when the user generates it. Identical context is reused until **Regenerate**.

Prompt classes live in `app/Services/AI/Prompts/`. Context is built in `ConversationAiContext` / `LeadAiContext` and only includes records the user can already view.

Phase 2 — Ask Algos

Natural-language search on the existing Global Search page. Keyword search is unchanged.

The model returns an allowlisted filter payload (`entity`, `filters`, `sort`, `limit`). `AskAlgosQueryValidator` rejects unknown fields/operators and SQL-like output. `AskAlgosQueryExecutor` runs Eloquent with existing scopes (`Lead::visibleTo`, `Task::visibleTo`, `CompanyScope`) and `Gate::view` after fetch. **AI-generated SQL is never executed.**

Item	Detail
Route	`POST search.ask` (`view.leads` / customers / tasks / inbox / activity logs)
Entities	leads, customers, tasks, activities (`ActivityLog`), conversations
Dates	Resolved in PHP (`today`, `this_month`, `last_7_days`, …)
Logs	`ai_search_logs` stores question + structured interpretation + status (not conversation bodies). Pruned after `AI_SEARCH_LOG_RETENTION_DAYS` (90).

Explanations are optional and grounded in authorized result titles/counts. If the model invents numbers, the app falls back to "Found N leads matching your request."

Configuration

Master switch is **off** by default (`AI_ENABLED=false`). No provider HTTP calls happen until you enable it and set a key.

Variable	Default	Purpose
`AI_ENABLED`	false	Master switch
`AI_PROVIDER`	openai	`openai` or `null`
`AI_API_KEY`	*(empty)*	Active provider API key — **never commit**
`AI_MODEL`	gpt-4o-mini	Default model

Variable	Default	Purpose
`AI_TIMEOUT`	20	HTTP timeout (seconds)
`AI_MAX_TOKENS`	1024	Completion cap
`AI_TEMPERATURE`	0.2	Sampling temperature
`AI_QUEUE`	ai	Queue name for `ProcessAIRequestJob`
`AI_JOB_TIMEOUT`	90	Queue worker timeout (seconds)
`AI_OPENAI_API_KEY`	*(falls back to `AI_API_KEY`)*	OpenAI-specific key
`AI_OPENAI_BASE_URL`	https://api.openai.com/v1	Override for compatible gateways
`AI_OPENAI_MODEL` / `AI_OPENAI_TIMEOUT` / `AI_OPENAI_MAX_TOKENS` / `AI_OPENAI_TEMPERATURE`	*(shared defaults)*	Per-provider overrides
`AI_OPENAI_ORGANIZATION`	*(empty)*	Optional OpenAI org header
`AI_LOG_USAGE`	true	Persist `ai_usage_logs`

Tests force `AI_ENABLED=false` and `AI_PROVIDER=null` in `phpunit.xml`. Automated tests must mock the provider / `Http::fake()` — they must not call a real API.

Provider abstraction

Callers depend on `AIProviderInterface`:

- `generateText(AIRequest $request): AIGenerationResult`
- `generateStructuredResponse(AIRequest $request, array $schema = []): AIGenerationResult`

To add a provider later:

1. Implement the interface under `app/Services/AI/Providers`.
2. Add a case to `App\Enums\AI\AIProvider`.
3. Register it in `AIServiceProvider`.
4. Add config under `config/ai.php` `providers`.

Do not put HTTP or SDK calls in controllers.

Tenancy and RBAC

AI is **not** a permission bypass.

1. `AIContextBuilder::forAuthenticatedUser($user)` sets `CurrentCompany` from `$user->company_id`. It never trusts a client-supplied company id.
2. Model loads use Eloquent + `CompanyScope`. Another tenant's row is invisible (`findAuthorized` → not found).
3. `authorize()` / `findAuthorized()` go through existing policies (`LeadPolicy`, `ConversationPolicy`, …) and `ChecksSameCompany`.
4. `AIService` refuses to run if `AIRequest.companyId` does not match `CurrentCompany`.
5. Super Admins have no company — tenant AI context is refused.
6. Usage logs are `BelongsToCompany`. They are not written if tenant context is missing or mismatched.

Queued work must restore context the same way channel jobs do:

```
$context->forQueuedOperation($request->companyId, $request->userId);
```

`ProcessAIRequestJob` already does this and clears context in `finally`.

Usage logging

`ai_usage_logs` stores metadata only:

- `company_id`, `user_id`, `feature`, `provider`, `model`
- `operation` (`text` / `structured` / `dispatch`)
- `provider_request_id`, token counts, `status`, sanitized `error_message`, `duration_ms`

It does **not** store prompts, completions, or conversation bodies. Error strings are redacted (API keys, `sk-…`, Bearer tokens) and never shown to end users.

How future features should call AI

Keep prompts out of controllers. Example for a later Inbox or Lead feature:

```
$context = app(AIContextBuilder::class)->forAuthenticatedUser($request->user());
$lead = $context->findAuthorized(Lead::class, $leadId, 'view');

$prompts = app(PromptAssembler::class)->compose(
    featureInstructions: 'Summarize this lead for the assigned salesperson.',
    userTemplate: "Name: {{name}}\nStatus: {{status}}",
    variables: ['name' => $lead->name, 'status' => $lead->status],
);

$request = $context->textRequest(
    feature: 'leads.summary',
    systemPrompt: $prompts['system'],
    userPrompt: $prompts['user'],
    metadata: ['lead_id' => $lead->id],
);

$outcome = app(AIService::class)->generateText($request);
// or: app(AIService::class)->dispatch($request);  // expensive work
```

Rules for feature work:

- Add a prompt class under `app/Services/AI/Prompts/` (e.g. `LeadSummaryPrompt`).
- Authorize with `AIContextBuilder` + existing abilities (`view`, `reply`, …). Add new permission slugs only if a new AI action needs its own gate.
- Treat `$outcome->ok === false` as a soft failure (flash the `userMessage`, keep the page working).
- Put slow calls on `AI_QUEUE` (`php artisan queue:work --queue=channels,ai,default`).
- Never log raw chat transcripts onto `ai_usage_logs`.
- Mock `AIProviderInterface` in tests; do not hit the network.

Related

- Multi-tenancy
- RBAC
- Queues
- Architecture overview