CompetitorResearchService

September 4, 2026

CompetitorResearchService

Kind: Service

Source: `atloria-monorepo/apps/api/src/documentation/services/competitor-research.service.ts`

Main orchestrator for competitor research
Combines AI discovery + web crawling

`CompetitorResearchService` is the main orchestration layer for competitor research in the backend. It coordinates AI-based competitor discovery with web crawling to gather and normalize data about identified competitors. This service typically sits above lower-level AI/crawler providers and returns consolidated research results to calling modules/controllers.

Methods

Method	Signature	Returns	Description
`researchCompetitors`	`researchCompetitors(projectType: string, projectDescription: string, options: {		

```
  maxCompetitors?: number;
  useCache?: boolean;
  crawlConcurrency?: number;
})` | `Promise<CompetitorResearchResult>` | Research competitors for a project 1. |
```

`formatForAIContext`	`formatForAIContext(research: CompetitorResearchResult)`	`string`	Get competitor documentation content as a formatted string Ready to be included in AI prompts
`clearCache`	`clearCache()`	`Promise<void>`	Clear all cached research
`onModuleDestroy`	`onModuleDestroy()`	`unknown`	Cleanup on destroy

Dependencies

- `CompetitorDiscoveryService`
- `WebCrawlerService`

Where it refuses work

- `CompetitorResearchService` stops the work with an early return when `!this.redis`, in 3 places.
- `CompetitorResearchService` stops the work with an early return when `research.crawledPages.length === 0`.
- `CompetitorResearchService` stops the work with an early return when `!cached`.

When something fails

- `CompetitorResearchService` handles failure in 4 places: it logs it and continues in 3, and turns it into a return value in 1.

Diagram

```
sequenceDiagram
  autonumber
  actor Client
  participant CRS as CompetitorResearchService
  participant AI as AI Discovery Provider
  participant Crawler as Web Crawler
  participant Store as Persistence/Repository

  Client->>CRS: runResearch(input)
  CRS->>AI: discoverCompetitors(input)
  AI-->>CRS: competitorCandidates[]
  loop for each competitor
    CRS->>Crawler: crawl(competitor.url)
    Crawler-->>CRS: pages/content/metadata
    CRS->>CRS: extract + normalize signals
  end
  CRS->>Store: saveResearchResult(result)
  Store-->>CRS: savedResult
  CRS-->>Client: researchResult
```

Usage

```ts
import { Injectable } from '@nestjs/common';
import { CompetitorResearchService } from './documentation/services/competitor-research.service

@Injectable()
export class ResearchRunner {
  constructor(private readonly competitorResearch: CompetitorResearchService) {}

  async run() {
    // Shape depends on your implementation; keep it aligned with the DTO/interface used by the
    const input = {
      companyName: 'Atloria',
      domain: 'atloria.com',
      market: 'B2B SaaS',
      maxCompetitors: 10,
    };

    const result = await this.competitorResearch.runResearch(input);

    // e.g., competitors with evidence/links and extracted positioning signals
    return {
      count: result.competitors?.length ?? 0,
      competitors: result.competitors,
      generatedAt: result.generatedAt,
    };
  }
}
```

AI Coding Instructions

- Keep `CompetitorResearchService` focused on orchestration: delegate AI discovery and crawling details to dedicated providers/services rather than embedding implementation logic here.
- Treat external calls (LLM + crawling) as unreliable: add timeouts, retries/backoff, and graceful partial results instead of failing the entire run on one competitor.
- Normalize outputs at the boundary: ensure AI-discovered competitors and crawled artifacts are mapped into consistent internal DTOs before persistence/return.
- Avoid unbounded loops and crawling: enforce limits (`maxCompetitors` , max pages/depth, allowed domains) to prevent runaway requests and high costs.
- Maintain clear integration points: when changing discovery prompts/schema or crawler extraction, update the parsing/validation in this service to keep downstream consumers stable.

Relationships

- DEPENDS_ON → `CompetitorDiscoveryService`
- DEPENDS_ON → `WebCrawlerService`