AnalysisService

September 4, 2026

AnalysisService

Kind: Service

Source: `atloria-monorepo/apps/code-analyzer/src/app/analysis/analysis.service.ts`

Analysis orchestrator service

This service coordinates the entire code analysis pipeline:

1. Clone repository (with mono-repo support)
2. Analyze codebase using file optimization techniques
3. Generate UI documentation and element extraction
4. Create Playwright tests
5. Track progress in database
6. Cleanup after completion

`AnalysisService` orchestrates the end-to-end repository analysis workflow for the code analyzer application. It coordinates repository cloning (including monorepos), optimized code analysis, UI documentation and element extraction, Playwright test generation, database-backed progress tracking, and cleanup once processing completes.

Methods

Method	Signature	Returns	Description
`startAnalysis`	`startAnalysis(dto: StartAnalysisDto)`	`Promise<string>`	Start a new code analysis job
`getJobStatus`	`getJobStatus(jobId: string)`	`unknown`	Get analysis job status
`getProjectJobs`	`getProjectJobs(projectId: string)`	`unknown`	Get all jobs for a project

Dependencies

- `PrismaService`
- `GitService`

Where it refuses work

- `AnalysisService` stops the work with `NotFoundException` when `!job`.
- `AnalysisService` stops the work with an early return when `deps['next']`.
- `AnalysisService` stops the work with an early return when `deps['@nestjs/core']`.
- `AnalysisService` stops the work with an early return when `deps['react']`.
- `AnalysisService` stops the work with an early return when `deps['@angular/core']`.
- `AnalysisService` stops the work with an early return when `deps['vue']`.

When something fails

- `AnalysisService` handles failure in 2 places: it turns it into a return value in 1, and lets it reach the caller in 1.

Diagram

```
sequenceDiagram
  participant Client
  participant AnalysisService
  participant Database
  participant RepositoryService
  participant CodeAnalyzer
  participant DocumentationGenerator
  participant PlaywrightGenerator

  Client->>AnalysisService: startAnalysis(request)
  AnalysisService->>Database: Create analysis job / set status
  AnalysisService->>RepositoryService: Clone repository and resolve monorepo
  RepositoryService-->>AnalysisService: Local repository path

  AnalysisService->>CodeAnalyzer: Analyze optimized file set
  CodeAnalyzer-->>AnalysisService: Code analysis results
  AnalysisService->>Database: Update analysis progress

  AnalysisService->>DocumentationGenerator: Generate UI docs and extract elements
  DocumentationGenerator-->>AnalysisService: Documentation artifacts
  AnalysisService->>Database: Update generation progress

  AnalysisService->>PlaywrightGenerator: Generate Playwright tests
```

```
  PlaywrightGenerator-->>AnalysisService: Test artifacts

  AnalysisService->>Database: Mark job completed
  AnalysisService->>RepositoryService: Cleanup cloned repository
  AnalysisService-->>Client: Analysis job ID

  Client->>AnalysisService: getJobStatus(jobId)
  AnalysisService->>Database: Retrieve job status
  Database-->>AnalysisService: Job status
  AnalysisService-->>Client: Job status
```

Usage

```
import { Injectable } from '@nestjs/common';
import { AnalysisService } from './analysis.service';

@Injectable()
export class ProjectAnalysisController {
  constructor(private readonly analysisService: AnalysisService) {}

  async analyzeProject(projectId: string, repositoryUrl: string) {
    const jobId = await this.analysisService.startAnalysis({
      projectId,
      repositoryUrl,
      branch: 'main',
    });

    return {
      jobId,
      message: 'Analysis started successfully',
    };
  }

  async getAnalysisStatus(jobId: string) {
    return this.analysisService.getJobStatus(jobId);
  }

  async listProjectAnalysisJobs(projectId: string) {
    return this.analysisService.getProjectJobs(projectId);
  }
}
```

AI Coding Instructions

- Keep `AnalysisService` focused on orchestration; delegate cloning, parsing, documentation generation, and test generation to specialized services.

- Update the persisted job status at meaningful pipeline boundaries so clients can reliably poll `getJobStatus()`.
- Ensure repository cleanup runs in a `finally` block or equivalent failure-safe path, including when cloning or generation steps fail.
- Preserve monorepo-aware repository resolution when adding analysis stages; downstream tools should receive the resolved package or workspace path.
- Return the analysis job ID immediately from `startAnalysis()` and avoid blocking request handlers on long-running analysis work.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `GitService`

Referenced By

- `AnalysisController` (DEPENDS_ON)
- `AnalysisModule` (MODULE_PROVIDES)
- `AnalysisModule` (MODULE_EXPORTS)