BackendParserService

September 4, 2026

BackendParserService

Kind: Service

Source: `atloria-monorepo/apps/backend-parser/src/backend-parser.service.ts`

Backend Parser Service

Provides a unified interface for all backend parsers.
Automatically selects the appropriate parser based on file type.

`BackendParserService` provides a single entry point for parsing backend source files in the Atloria monorepo. It detects the input file type, selects the matching backend parser, and returns a consistent parsing result for downstream consumers such as documentation, analysis, or code-generation workflows.

Methods

Method	Signature	Returns	Description
`onModuleInit`	`onModuleInit()`	`Promise<void>`	
`onModuleDestroy`	`onModuleDestroy()`	`Promise<void>`	
`initialize`	`initialize(config: ParserConfig)`	`Promise<void>`	Initialize all parsers with configuration
`getParsers`	`getParsers()`	`IParser[]`	Get all registered parsers
`getParser`	`getParser(name: string)`	`IParser	undefined`	
`findParserForFile`	`findParserForFile(filePath: string)`	`IParser	undefined`	
`parseFile`	`parseFile(filePath: string, content: string)`	`Promise<ParseResult>`	Parse a single file using the

Method	Signature	Returns	Description
			appropriate parser
`parseFiles`	`parseFiles(files: ParseFileInput[])`	`Promise<ParseResult[]>`	Parse multiple files
`parseFilesGrouped`	`parseFilesGrouped(files: ParseFileInput[])`	`Promise<ParseResult[]>`	Parse files grouped by parser type for efficiency
`getSupportedPatterns`	`getSupportedPatterns()`	`string[]`	Get supported file patterns from all parsers
`getParserMetadata`	`getParserMetadata()`	`Array<{`	

```
name: string;
version: string;
supportedPatterns: string[];
```

}>` | Get parser metadata for registration |

Dependencies

- `NestjsParser`
- `ExpressParser`
- `FastAPIParser`
- `DjangoParser`

Where it refuses work

- `BackendParserService` stops the work with an early return when `!parser`.

Diagram

```
sequenceDiagram
  participant Client as Calling Service
  participant ParserService as BackendParserService
  participant Selector as Parser Selection
  participant Parser as Backend-Specific Parser
```

```
  Client->>ParserService: parse(filePath, sourceCode)
  ParserService->>Selector: Determine parser from file type
  Selector-->>ParserService: Matching parser
  ParserService->>Parser: parse(sourceCode, filePath)
  Parser-->>ParserService: Parsed entity metadata
  ParserService-->>Client: Unified parse result
```

Usage

```
import { BackendParserService } from './backend-parser.service';

async function parseBackendFile(
  parserService: BackendParserService,
  filePath: string,
  sourceCode: string,
) {
  const result = await parserService.parse(filePath, sourceCode);

  console.log(result);
  return result;
}

// In a NestJS service or controller, inject BackendParserService
// through the constructor and pass the target file contents.
```

AI Coding Instructions

- Route all backend source parsing through `BackendParserService` rather than instantiating individual parsers directly.
- Add support for new backend file types by registering a dedicated parser and updating the service's file-type selection logic.
- Keep parser outputs consistent so consumers can handle results without knowing which parser processed the file.
- Handle unsupported extensions and malformed source code explicitly; return actionable errors instead of silently selecting a fallback parser.
- Preserve NestJS dependency-injection patterns when adding parser dependencies or extending this service.

Relationships

- DEPENDS_ON → `nestjsparser`

- DEPENDS_ON → `expressparser`
- DEPENDS_ON → `fastapiparser`
- DEPENDS_ON → `djangoparser`

Referenced By

- `AppModule` (MODULE_PROVIDES)
- `AppModule` (MODULE_EXPORTS)