Generated Project Fusion File

Project: project-fusion v0.0.1

Generated: 16/08/2025 23:47:54 UTC−4

UTC: 2025-08-17T03:47:54.051Z

Files: 22

Generated by: project-fusion

📁 Table of Contents

📄 CHANGELOG.md

TODO

CLAUDE.md

```
# Project Fusion - AI Context

> 📖 **For Human Development**: See [DEVELOPMENT.md](./DEVELOPMEN

## Project Overview
Project Fusion merges multiple project files into a single file f

## Essential Architecture
- **TypeScript 5.9.2** ESM project with strict type checking
- **CLI tool** built with Commander.js that generates .txt and .m
- **Configuration-driven** with Zod validation and default fallba
- **Multi-format output** with syntax highlighting and filtering

## Core Files Structure
```
src/
├── cli.ts   # CLI entry point
├── clicommands.ts   # Command implementations
├── fusion.ts   # Core fusion logic
├── types.ts   # Type definitions (branded types)
├── schema.ts   # Zod validation schemas
├── utils.ts   # File operations & utilities
└── index.ts   # Main exports
```

## Key Commands
```bash
npm run build           # Build TypeScript → JavaScript
npm run typecheck       # Type checking only
project-fusion init     # Initialize config
project-fusion fusion   # Run fusion process
```

Testing Directory
⚠️ Important: All testing and temporary files MUST be created
- Package testing: `temp/package/`
```

````
- File generation tests: `temp/test-files/`
- Any temporary artifacts: `temp/artifacts/`

The `temp/` directory is gitignored and safe for any testing acti

## Configuration Schema
```typescript
{
  schemaVersion: 1
  copyToClipboard: boolean
  generatedFileName: string
  generateHtml: boolean
  generateMarkdown: boolean
  generatePdf: boolean
  generateText: boolean
  maxFileSizeKB: number
  parseSubDirectories: boolean
  parsedFileExtensions: {
    web: string[]       // .js, .ts, .tsx, .vue, etc.
    backend: string[]   // .py, .go, .java, .rs, etc.
    config: string[]    // .json, .yaml, .toml, etc.
    cpp: string[]       // .c, .cpp, .h, .hpp
    scripts: string[]   // .sh, .bat, .ps1
    godot: string[]     // .gd, .tscn, .tres
    doc: string[]       // .md, .rst, .adoc
  }
  rootDirectory: string
  ignorePatterns: string[]
  useGitIgnoreForExcludes: boolean
}
```

## Core Workflow
1. Load `project-fusion.json` config with Zod validation
2. Scan files by extensions, apply .gitignore + custom ignore pat
3. Generate dual output:
   - `project-fusioned.txt` - Plain text with separators
   - `project-fusioned.md` - Markdown with syntax highlighting +

## Key Implementation Details
- **Branded types** (FilePath) prevent string confusion
- **Discriminated unions** (FusionResult) for type-safe error han
- **ESM modules** with strict TypeScript
- **Configuration fallbacks** - uses defaults if config missing/i

## Quick Reference
- **Add extensions**: Update `src/schema.ts` + `src/utils.ts` def
- **Add commands**: Register in `src/cli.ts`, implement in `src/c
- **Modify output**: Edit `src/fusion.ts` processing logic
````

CONTRIBUTING.md

```
# Contributing to Project Fusion

Thanks for your interest! This guide explains how to propose chan

## Code of Conduct
Be respectful and constructive. By participating, you agree to fo

## How to contribute
1. **Fork** the repo and create a branch: `feat/<short-name>` or
2. See [DEVELOPMENT.md](./DEVELOPMENT.md) for detailed developmen
3. Open a Pull Request with a clear description and checklist.

## PR checklist
- [ ] Feature/bugfix tested
- [ ] No regressions
- [ ] Docs/README updated if necessary
- [ ] Notable changes added to `CHANGELOG.md`
```

DEVELOPMENT.md

```
# Project Fusion - Development Guide

> 📋 **For Claude AI Context**: See [CLAUDE.md](./CLAUDE.md) for

## 🚀 Development Workflow

### Initial Setup
```bash
git clone https://github.com/the99studio/project-fusion.git
cd project-fusion
npm install
npm run build
```
```

```

### Claude Code Integration
The project includes `.claude/settings.local.json` which configur

**Allowed Operations:**
- NPM commands: install, build, typecheck, test, clean, pack
- Project CLI: `project-fusion` and `node dist/cli.js` commands
- Git operations: status, diff, log, branch, add, commit, push, p
- Safe file operations: Limited to `temp/` directory for rm/cp op
- Search capabilities: find, grep, rg, ls, cat, head, tail for co
- Package management: npm list, outdated, view

**Security Features:**
- File deletions restricted to `temp/` directory only
- No arbitrary Node.js code execution (only specific CLI commands
- Explicit deny list for dangerous operations (sudo, eval, etc.)
- No system-wide file modifications allowed

These permissions eliminate repetitive authorization prompts whil

### Testing the CLI
Use VS Code launch configurations (F5) for easy testing:
- **"Fusion (Default)"** - Default behavior (runs fusion)
- **"Fusion (Web)"** - Test web extensions only
- **"Help"** - Test CLI help
- **"Init"** - Test project initialization

### Testing with Real Package
For testing as if it were the real published package, see the [NP

## 📦 NPM Package Management

### Pre-Publication Testing

Use the **"Test NPM Package"** launch configuration in VS Code (F
- Builds the project
- Creates and extracts test package to `temp/package/`
- Installs dependencies and tests CLI functionality

#### Manual Package Verification
```bash
# Preview what will be published
npm pack --dry-run

# Create test package (if not using VS Code)
npm pack  # Creates project-fusion-x.x.x.tgz
```
```

Testing with Real Package Installation
```bash
# Install the test package globally
npm install -g ./temp/package/ # start line with sudo if you need

# Test commands (acts like real published package)
project-fusion --help
project-fusion --version
project-fusion init
project-fusion # Default: runs fusion

# Uninstall when done testing
npm uninstall -g project-fusion # start line with sudo if you nee
```

Publication Process

```bash
# 1. Final verification
npm pack --dry-run

# 2. Simulate publication (verifies authentication, package valid
npm publish --dry-run

# 3. Create npm account and login (first time only)
# Visit https://www.npmjs.com/signup to create account
npm login

# 4. Publish to npm
npm publish

# 5. Verify publication
npm view project-fusion
```

🛠️ Development Patterns

Adding New File Extensions
1. Update `src/schema.ts` - add to `ParsedFileExtensionsSchema`
2. Update default config in `src/utils.ts`
3. Test with various projects

Adding New CLI Commands
1. Register command in `src/cli.ts` (Commander.js)
2. Implement in `src/clicommands.ts`
3. Update help text and documentation

Modifying Fusion Output
1. Edit `src/fusion.ts` processing logic

````
2. Update types in `src/types.ts` if needed
3. Test both .txt and .md output formats

## 🧪 Testing Strategy

### Manual Testing Checklist
- [ ] `npm run build` - clean build
- [ ] `npm run typecheck` - no type errors
- [ ] CLI help works: `project-fusion --help`
- [ ] Init works: `project-fusion init`
- [ ] Fusion works: `project-fusion fusion`
- [ ] Extension filtering works
- [ ] .gitignore integration works
- [ ] Output files are properly formatted
- [ ] Package builds and installs correctly

### Test Projects
Use these types of projects for testing:
- **Node.js/TypeScript** (like this project)
- **Python projects** (test backend extensions)
- **React/Vue projects** (test web extensions)
- **Mixed projects** (multiple extension types)

## 🔧 Troubleshooting

### Common Issues

**Build Errors:**
```bash
npm run clean && npm run build
```

**Package Contains Wrong Files:**
- Check `package.json` `files` field
- Use `npm pack --dry-run` to verify

**TypeScript Errors:**
```bash
npm run typecheck
# Fix errors in src/ files
```

## 📁 Directory Structure

```
project-fusion/
├── src/                    # TypeScript source
|   ├── cli.ts              # CLI entry point
|   ├── clicommands.ts      # Command implementations
````
````

```
|   ├── fusion.ts          # Core fusion logic
|   ├── types.ts           # Type definitions
|   ├── schema.ts          # Zod schemas
|   ├── utils.ts           # Utilities
|   └── index.ts           # Main exports
├── dist/                  # Compiled JavaScript (gitignored)
├── temp/                  # Testing directory (gitignored)
├── CLAUDE.md              # AI context (essential info)
├── DEVELOPMENT.md         # This file (human development)
├── package.json           # NPM configuration
└── tsconfig.json          # TypeScript configuration
```

🔗 Important Files

- **CLAUDE.md** - Essential project context for AI assistance
- **package.json** - NPM package configuration and scripts
- **tsconfig.json** - TypeScript compilation settings
- **.gitignore** - Git ignore patterns (includes `temp/`)
- **.vscode/launch.json** - VS Code debugging/testing configurati

📄 package.json

```
{
    "name": "project-fusion",
    "version": "0.0.1",
    "description": "CLI tool for merging project files into a sin
    "main": "dist/index.js",
    "types": "dist/index.d.ts",
    "type": "module",
    "bin": {
        "project-fusion": "dist/cli.js"
    },
    "exports": {
        ".": {
            "types": "./dist/index.d.ts",
            "import": "./dist/index.js"
        },
        "./package.json": "./package.json"
    },
    "files": [
```

```
        "dist/**/*",
        "README.md",
        "LICENSE",
        "CHANGELOG.md"
    ],
    "sideEffects": false,
    "scripts": {
        "build": "tsc",
        "clean": "rm -rf dist",
        "dev": "tsc --watch",
        "prepublishOnly": "npm run clean && npm run build",
        "test": "vitest",
        "test:coverage": "vitest run --coverage",
        "test:ui": "vitest --ui",
        "typecheck": "tsc --noEmit"
    },
    "keywords": [
        "cli",
        "code",
        "collaboration",
        "files",
        "fusion",
        "merge",
        "sharing"
    ],
    "author": "the99studio",
    "license": "MIT",
    "engines": {
        "node": ">=18.0.0"
    },
    "repository": {
        "type": "git",
        "url": "https://github.com/the99studio/project-fusion.git
    },
    "bugs": {
        "url": "https://github.com/the99studio/project-fusion/iss
    },
    "homepage": "https://github.com/the99studio/project-fusion#re
    "dependencies": {
        "chalk": "^5.5.0",
        "clipboardy": "^4.0.0",
        "commander": "^14.0.0",
        "fs-extra": "^11.3.1",
        "glob": "^11.0.3",
        "ignore": "^7.0.5",
        "puppeteer": "^24.16.2",
        "zod": "^4.0.17"
    },
    "devDependencies": {
```

```
        "@types/fs-extra": "^11.0.4",
        "@types/node": "^24.2.1",
        "@vitest/coverage-v8": "^2.1.9",
        "typescript": "^5.9.2",
        "vitest": "^2.1.6"
    }
}
```

📄 project-fusioned.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-sc
    <title>Project Fusion - project-fusion v0.0.1</title>
    <style>
        body { font-family: -apple-system, BlinkMacSystemFont, 'S
        .header { border-bottom: 2px solid #eee; padding-bottom:
        .file-section { margin-bottom: 40px; border: 1px solid #d
        .file-title { background: #f5f5f5; margin: -20px -20px 20
        pre { background: #f8f9fa; padding: 15px; border-radius:
        code { font-family: 'Monaco', 'Menlo', 'Ubuntu Mono', mon
        .toc { background: #f8f9fa; padding: 20px; border-radius:
        .toc ul { margin: 0; padding-left: 20px; }
        .toc a { text-decoration: none; color: #0366d6; }
        .toc a:hover { text-decoration: underline; }
    </style>
</head>
<body>
    <div class="header">
        <h1>Generated Project Fusion File</h1>
        <p><strong>Project:</strong> project-fusion v0.0.1</p>
        <p><strong>Generated:</strong> 16/08/2025 23:47:54 UTC-4<
        <p><strong>UTC:</strong> 2025-08-17T03:47:54.051Z</p>
        <p><strong>Files:</strong> 22</p>
        <p><strong>Generated by:</strong> <a href="https://github
    </div>
    <div class="toc">
        <h2>📁 Table of Contents</h2>
        <ul>
```

```
            <li><a href="#changelog-md">CHANGELOG.md</a></li>
            <li><a href="#claude-md">CLAUDE.md</a></li>
            <li><a href="#contributing-md">CONTRIBUTING.md</a></l
            <li><a href="#development-md">DEVELOPMENT.md</a></li>
            <li><a href="#package-json">package.json</a></li>
            <li><a href="#project-fusioned-html">project-fusioned
            <li><a href="#readme-md">README.md</a></li>
            <li><a href="#src-benchmark-ts">src/benchmark.ts</a><
            <li><a href="#src-cli-ts">src/cli.ts</a></li>
            <li><a href="#src-clicommands-ts">src/clicommands.ts<
            <li><a href="#src-fusion-ts">src/fusion.ts</a></li>
            <li><a href="#src-index-ts">src/index.ts</a></li>
            <li><a href="#src-schema-ts">src/schema.ts</a></li>
            <li><a href="#src-types-ts">src/types.ts</a></li>
            <li><a href="#src-utils-ts">src/utils.ts</a></li>
            <li><a href="#tests-formats-test-ts">tests/formats.te
            <li><a href="#tests-integration-test-ts">tests/integr
            <li><a href="#tests-schema-test-ts">tests/schema.test
            <li><a href="#tests-utils-test-ts">tests/utils.test.t
            <li><a href="#todo-md">TODO.md</a></li>
            <li><a href="#tsconfig-json">tsconfig.json</a></li>
            <li><a href="#vitest-config-ts">vitest.config.ts</a><
        </ul>
    </div>    <div class="file-section" id="changelog-md">
        <div class="file-title">
            <h2>📄 CHANGELOG.md</h2>
        </div>
        <pre><code class="markdown">TODO</code></pre>
    </div>

    <div class="file-section" id="claude-md">
        <div class="file-title">
            <h2>📄 CLAUDE.md</h2>
        </div>
        <pre><code class="markdown"># Project Fusion - AI Context

&gt; 📖 **For Human Development**: See [DEVELOPMENT.md](./DEVELOP

## Project Overview
Project Fusion merges multiple project files into a single file f

## Essential Architecture
- **TypeScript 5.9.2** ESM project with strict type checking
- **CLI tool** built with Commander.js that generates .txt and .m
- **Configuration-driven** with Zod validation and default fallba
- **Multi-format output** with syntax highlighting and filtering

## Core Files Structure
```
```
```

```
src/
├── cli.ts              # CLI entry point
├── clicommands.ts      # Command implementations
├── fusion.ts           # Core fusion logic
├── types.ts            # Type definitions (branded types)
├── schema.ts           # Zod validation schemas
├── utils.ts            # File operations &amp;amp; utilities
└── index.ts            # Main exports
```

Key Commands
```bash
npm run build           # Build TypeScript → JavaScript
npm run typecheck       # Type checking only
project-fusion init     # Initialize config
project-fusion fusion   # Run fusion process
```

Testing Directory
⚠️ Important: All testing and temporary files MUST be created
- Package testing: `temp/package/`
- File generation tests: `temp/test-files/`
- Any temporary artifacts: `temp/artifacts/`

The `temp/` directory is gitignored and safe for any testing acti

Configuration Schema
```typescript
{
  schemaVersion: 1
  copyToClipboard: boolean
  generatedFileName: string
  generateHtml: boolean
  generateMarkdown: boolean
  generatePdf: boolean
  generateText: boolean
  maxFileSizeKB: number
  parseSubDirectories: boolean
  parsedFileExtensions: {
    web: string[]       // .js, .ts, .tsx, .vue, etc.
    backend: string[]   // .py, .go, .java, .rs, etc.
    config: string[]    // .json, .yaml, .toml, etc.
    cpp: string[]       // .c, .cpp, .h, .hpp
    scripts: string[]   // .sh, .bat, .ps1
    godot: string[]     // .gd, .tscn, .tres
    doc: string[]       // .md, .rst, .adoc
  }
  rootDirectory: string
  ignorePatterns: string[]
```

```
  useGitIgnoreForExcludes: boolean
}
```

Core Workflow
1. Load `project-fusion.json` config with Zod validation
2. Scan files by extensions, apply .gitignore + custom ignore pat
3. Generate dual output:
   - `project-fusioned.txt` - Plain text with separators
   - `project-fusioned.md` - Markdown with syntax highlighting +

Key Implementation Details
- **Branded types** (FilePath) prevent string confusion
- **Discriminated unions** (FusionResult) for type-safe error han
- **ESM modules** with strict TypeScript
- **Configuration fallbacks** - uses defaults if config missing/i

Quick Reference
- **Add extensions**: Update `src/schema.ts` + `src/utils.ts` def
- **Add commands**: Register in `src/cli.ts`, implement in `src/c
- **Modify output**: Edit `src/fusion.ts` processing logic</code>

            <h2> CONTRIBUTING.md</h2>

        <pre><code class="markdown"># Contributing to Project Fus

Thanks for your interest! This guide explains how to propose chan

Code of Conduct
Be respectful and constructive. By participating, you agree to fo

How to contribute
1. **Fork** the repo and create a branch: `feat/<short-name&gt
2. See [DEVELOPMENT.md](./DEVELOPMENT.md) for detailed developmen
3. Open a Pull Request with a clear description and checklist.

PR checklist
- [ ] Feature/bugfix tested
- [ ] No regressions
- [ ] Docs/README updated if necessary
- [ ] Notable changes added to `CHANGELOG.md`
</code></pre>

```
            <h2>📄 DEVELOPMENT.md</h2>
        </div>
        <pre><code class="markdown"># Project Fusion - Developmen

&amp;gt; 📋 **For Claude AI Context**: See [CLAUDE.md](./CLAUDE.md) f

## 🚀 Development Workflow

### Initial Setup
```bash
git clone https://github.com/the99studio/project-fusion.git
cd project-fusion
npm install
npm run build
```

Claude Code Integration
The project includes `.claude/settings.local.json` which configur

Allowed Operations:
- NPM commands: install, build, typecheck, test, clean, pack
- Project CLI: `project-fusion` and `node dist/cli.js` commands
- Git operations: status, diff, log, branch, add, commit, push, p
- Safe file operations: Limited to `temp/` directory for rm/cp op
- Search capabilities: find, grep, rg, ls, cat, head, tail for co
- Package management: npm list, outdated, view

Security Features:
- File deletions restricted to `temp/` directory only
- No arbitrary Node.js code execution (only specific CLI commands
- Explicit deny list for dangerous operations (sudo, eval, etc.)
- No system-wide file modifications allowed

These permissions eliminate repetitive authorization prompts whil

Testing the CLI
Use VS Code launch configurations (F5) for easy testing:
- **&quot;Fusion (Default)&quot;** - Default behavior (runs fusio
- **&quot;Fusion (Web)&quot;** - Test web extensions only
- **&quot;Help&quot;** - Test CLI help
- **&quot;Init&quot;** - Test project initialization

Testing with Real Package
For testing as if it were the real published package, see the [NP

📦 NPM Package Management

Pre-Publication Testing
```

````
Use the **&amp;quot;Test NPM Package&amp;quot;** launch configuration in
- Builds the project
- Creates and extracts test package to `temp/package/`
- Installs dependencies and tests CLI functionality

#### Manual Package Verification
```bash
# Preview what will be published
npm pack --dry-run

# Create test package (if not using VS Code)
npm pack  # Creates project-fusion-x.x.x.tgz
```

#### Testing with Real Package Installation
```bash
# Install the test package globally
npm install -g ./temp/package/ # start line with sudo if you need

# Test commands (acts like real published package)
project-fusion --help
project-fusion --version
project-fusion init
project-fusion # Default: runs fusion

# Uninstall when done testing
npm uninstall -g project-fusion # start line with sudo if you nee
```

### Publication Process

```bash
# 1. Final verification
npm pack --dry-run

# 2. Simulate publication (verifies authentication, package valid
npm publish --dry-run

# 3. Create npm account and login (first time only)
# Visit https://www.npmjs.com/signup to create account
npm login

# 4. Publish to npm
npm publish

# 5. Verify publication
npm view project-fusion
```
````

```
## 🛠️ Development Patterns

### Adding New File Extensions
1. Update `src/schema.ts` - add to `ParsedFileExtensionsSchema`
2. Update default config in `src/utils.ts`
3. Test with various projects

### Adding New CLI Commands
1. Register command in `src/cli.ts` (Commander.js)
2. Implement in `src/clicommands.ts`
3. Update help text and documentation

### Modifying Fusion Output
1. Edit `src/fusion.ts` processing logic
2. Update types in `src/types.ts` if needed
3. Test both .txt and .md output formats

## 🧪 Testing Strategy

### Manual Testing Checklist
- [ ] `npm run build` - clean build
- [ ] `npm run typecheck` - no type errors
- [ ] CLI help works: `project-fusion --help`
- [ ] Init works: `project-fusion init`
- [ ] Fusion works: `project-fusion fusion`
- [ ] Extension filtering works
- [ ] .gitignore integration works
- [ ] Output files are properly formatted
- [ ] Package builds and installs correctly

### Test Projects
Use these types of projects for testing:
- **Node.js/TypeScript** (like this project)
- **Python projects** (test backend extensions)
- **React/Vue projects** (test web extensions)
- **Mixed projects** (multiple extension types)

## 🔧 Troubleshooting

### Common Issues

**Build Errors:**
```bash
npm run clean && npm run build
```

Package Contains Wrong Files:
- Check `package.json` `files` field
- Use `npm pack --dry-run` to verify
```

TypeScript Errors:
```bash
npm run typecheck
# Fix errors in src/ files
```

📁 Directory Structure

```
project-fusion/
├── src/                    # TypeScript source
|   ├── cli.ts              # CLI entry point
|   ├── clicommands.ts      # Command implementations
|   ├── fusion.ts           # Core fusion logic
|   ├── types.ts            # Type definitions
|   ├── schema.ts           # Zod schemas
|   ├── utils.ts            # Utilities
|   └── index.ts            # Main exports
├── dist/                   # Compiled JavaScript (gitignored)
├── temp/                   # Testing directory (gitignored)
├── CLAUDE.md               # AI context (essential info)
├── DEVELOPMENT.md          # This file (human development)
├── package.json            # NPM configuration
└── tsconfig.json           # TypeScript configuration
```

🔗 Important Files

- **CLAUDE.md** - Essential project context for AI assistance
- **package.json** - NPM package configuration and scripts
- **tsconfig.json** - TypeScript compilation settings
- **.gitignore** - Git ignore patterns (includes `temp/`)
- **.vscode/launch.json** - VS Code debugging/testing configurati

            <h2>📄 package.json</h2>

        <pre><code class="json">{
    "name": "project-fusion",
    "version": "0.0.1",
    "description": "CLI tool for merging project f
    "main": "dist/index.js",
    "types": "dist/index.d.ts",
    "type": "module",
    "bin": {
        "project-fusion": "dist/cli.js"

```
    },
    &quot;exports&quot;: {
        &quot;.&quot;: {
            &quot;types&quot;: &quot;./dist/index.d.ts&quot;,
            &quot;import&quot;: &quot;./dist/index.js&quot;
        },
        &quot;./package.json&quot;: &quot;./package.json&quot;
    },
    &quot;files&quot;: [
        &quot;dist/**/*&quot;,
        &quot;README.md&quot;,
        &quot;LICENSE&quot;,
        &quot;CHANGELOG.md&quot;
    ],
    &quot;sideEffects&quot;: false,
    &quot;scripts&quot;: {
        &quot;build&quot;: &quot;tsc&quot;,
        &quot;clean&quot;: &quot;rm -rf dist&quot;,
        &quot;dev&quot;: &quot;tsc --watch&quot;,
        &quot;prepublishOnly&quot;: &quot;npm run clean &amp;&amp
        &quot;test&quot;: &quot;vitest&quot;,
        &quot;test:coverage&quot;: &quot;vitest run --coverage&qu
        &quot;test:ui&quot;: &quot;vitest --ui&quot;,
        &quot;typecheck&quot;: &quot;tsc --noEmit&quot;
    },
    &quot;keywords&quot;: [
        &quot;cli&quot;,
        &quot;code&quot;,
        &quot;collaboration&quot;,
        &quot;files&quot;,
        &quot;fusion&quot;,
        &quot;merge&quot;,
        &quot;sharing&quot;
    ],
    &quot;author&quot;: &quot;the99studio&quot;,
    &quot;license&quot;: &quot;MIT&quot;,
    &quot;engines&quot;: {
        &quot;node&quot;: &quot;&gt;=18.0.0&quot;
    },
    &quot;repository&quot;: {
        &quot;type&quot;: &quot;git&quot;,
        &quot;url&quot;: &quot;https://github.com/the99studio/pro
    },
    &quot;bugs&quot;: {
        &quot;url&quot;: &quot;https://github.com/the99studio/pro
    },
    &quot;homepage&quot;: &quot;https://github.com/the99studio/pr
    &quot;dependencies&quot;: {
        &quot;chalk&quot;: &quot;^5.5.0&quot;,
```

```
        &quot;clipboardy&quot;: &quot;^4.0.0&quot;,
        &quot;commander&quot;: &quot;^14.0.0&quot;,
        &quot;fs-extra&quot;: &quot;^11.3.1&quot;,
        &quot;glob&quot;: &quot;^11.0.3&quot;,
        &quot;ignore&quot;: &quot;^7.0.5&quot;,
        &quot;puppeteer&quot;: &quot;^24.16.2&quot;,
        &quot;zod&quot;: &quot;^4.0.17&quot;
    },
    &quot;devDependencies&quot;: {
        &quot;@types/fs-extra&quot;: &quot;^11.0.4&quot;,
        &quot;@types/node&quot;: &quot;^24.2.1&quot;,
        &quot;@vitest/coverage-v8&quot;: &quot;^2.1.9&quot;,
        &quot;typescript&quot;: &quot;^5.9.2&quot;,
        &quot;vitest&quot;: &quot;^2.1.6&quot;
    }
}
</code></pre>
    </div>
```


README.md

```
# Project Fusion

Project Fusion enables efficient project file management by mergi

## Prerequisites

- **Node.js** version 18.0.0 or higher

## Installation

Install Project Fusion globally with npm:

```bash
npm install -g project-fusion
```

Quick Start

1. **Initialize** Project Fusion in your project directory if you
   ```bash
```
```

```
   cd your-project-directory
   project-fusion init
   ```

2. **Create fusion files** containing all your project files (if
   ```bash
   project-fusion fusion
   ```
   This creates two files:
   - `project-fusioned.txt` - Plain text format with clear file s
   - `project-fusioned.md` - Markdown format with syntax highligh

3. **Share the fusion files** for collaboration or analysis (choo

Commands

- `project-fusion init` - Initialize Project Fusion in current di
- `project-fusion fusion` - Create fusion file from project files
- `project-fusion config-check` - Validate configuration and show
- `project-fusion --help` - Show help information

Documentation

- **[CLAUDE.md](./CLAUDE.md)** - AI context and technical documen
- **[DEVELOPMENT.md](./DEVELOPMENT.md)** - Development workflows,
- **[CONTRIBUTING.md](./CONTRIBUTING.md)** - How to contribute to
- **[LICENSE](./LICENSE)** - MIT License terms

Usage Workflow

When sharing your code:

1. Run `project-fusion fusion` to create merged files
2. Choose the appropriate format:
   - **`.txt`** - Universal compatibility with clear HTML-style s
   - **`.md`** - Enhanced readability with syntax highlighting, c
3. Share the fusion file with colleagues or collaborators
4. Use for code review, AI analysis, documentation, or project ov

The fusion files contain all your project files in a single, orga

Configuration

Project Fusion creates a `project-fusion.json` configuration file
- File extensions to include (organized by category: web, backend
- Directories to scan or ignore
- Output file names and locations
- Use of .gitignore patterns
- Clipboard copying behavior

```
### Supported File Extensions

Project Fusion supports 35+ file extensions organized by category
- **Web**: .js, .jsx, .ts, .tsx, .html, .css, .vue, .svelte
- **Backend**: .py, .rb, .java, .cs, .go, .rs, .php
- **Config**: .json, .yaml, .yml, .toml, .xml
- **Scripts**: .sh, .bat, .ps1, .cmd
- **C/C++**: .c, .cpp, .h, .hpp
- **Godot**: .gd, .tscn, .tres, .cfg

The markdown output automatically applies appropriate syntax high

### Performance Features

- **File Size Limiting**: Configure `maxFileSizeKB` in `parsing`
- **Streaming Support**: Large projects are processed with stream
- **Performance Metrics**: Detailed benchmarks logged including t
- **Smart Filtering**: Automatically ignores binary files, images

## Distribution

- **GitHub**: [github.com/the99studio/project-fusion](https://git
- **NPM**: [npmjs.com/package/project-fusion](https://www.npmjs.c

## License

This project is licensed under the MIT License - see the [LICENSE
```

📄 src/benchmark.ts

```
/**
 * Benchmark utilities for performance monitoring
 */
import { performance } from 'perf_hooks';
import process from 'process';

export interface BenchmarkMetrics {
    averageFileProcessingTime: number;
    duration: number;
    filesProcessed: number;
    memoryUsed: number;
```

```
    throughputMBps: number;
    totalSizeMB: number;
}

export class BenchmarkTracker {
    private startTime: number;
    private startMemory: NodeJS.MemoryUsage;
    private fileTimings: number[] = [];
    private filesProcessed = 0;
    private totalBytes = 0;

    constructor() {
        this.startTime = performance.now();
        this.startMemory = process.memoryUsage();
    }

    /**
     * Record file processing metrics
     */
    markFileProcessed(sizeBytes: number, processingTimeMs?: numbe
        this.filesProcessed++;
        this.totalBytes += sizeBytes;
        if (processingTimeMs !== undefined) {
            this.fileTimings.push(processingTimeMs);
        }
    }

    /**
     * Calculate and return performance metrics
     */
    getMetrics(): BenchmarkMetrics {
        const endTime = performance.now();
        const endMemory = process.memoryUsage();

        const duration = (endTime - this.startTime) / 1000; // se
        const memoryUsed = (endMemory.heapUsed - this.startMemory
        const totalSizeMB = this.totalBytes / (1024 * 1024);

        const averageFileProcessingTime = this.fileTimings.length
            ? this.fileTimings.reduce((a, b) => a + b, 0) / this.
            : 0;

        const throughputMBps = duration > 0 ? totalSizeMB / durat

        return {
            duration,
            memoryUsed,
            filesProcessed: this.filesProcessed,
            totalSizeMB,
```

```
            averageFileProcessingTime,
            throughputMBps
        };
    }

    /**
     * Format metrics as human-readable string
     */
    formatMetrics(): string {
        const metrics = this.getMetrics();
        return [
            `Performance Metrics:`,
            `  Duration: ${metrics.duration.toFixed(2)}s`,
            `  Memory Used: ${metrics.memoryUsed.toFixed(2)} MB`,
            `  Files Processed: ${metrics.filesProcessed}`,
            `  Total Size: ${metrics.totalSizeMB.toFixed(2)} MB`,
            `  Average File Processing Time: ${metrics.averageFil
            `  Throughput: ${metrics.throughputMBps.toFixed(2)} M
        ].join('\n');
    }
}
```

src/cli.ts

```
#!/usr/bin/env node
/**
 * Command-line interface for Project Fusion
 */
import { Command } from 'commander';
import pkg from '../package.json' with { type: 'json' };
import {
    runConfigCheckCommand,
    runFusionCommand,
    runInitCommand
} from './clicommands.js';

const program = new Command();

program
    .name('project-fusion')
    .description('Project Fusion - Efficient project file managem
    .version(pkg.version, '-v, --version')
```

```
    .option('--extensions <groups>', 'Comma-separated list of ext
    .option('--root <directory>', 'Root directory to start scanni

program
    .command('fusion')
    .description('Run fusion process to merge project files')
    .action((options, command) => {
        const allOptions = { ...command.parent.opts(), ...options
        runFusionCommand(allOptions);
    });

program
    .command('init')
    .description('Initialize Project Fusion in the current direct
    .option('--force', 'Force initialization even if configuratio
    .action((options) => {
        runInitCommand(options);
    });

program
    .command('config-check')
    .description('Validate project-fusion.json and display active
    .action(() => {
        runConfigCheckCommand();
    });

// Parse command-line options for default fusion command
// Manual option parsing for default command when no explicit com
async function runDefaultCommand() {
    const options: { extensions?: string; root?: string } = {};
    const args = process.argv.slice(2);

    // Simple argument parsing for --extensions and --root flags
    for (let i = 0; i < args.length; i++) {
        if (args[i] === '--extensions' && args[i + 1]) {
            options.extensions = args[i + 1];
            i++; // Skip next argument as it's the value
        } else if (args[i] === '--root' && args[i + 1]) {
            options.root = args[i + 1];
            i++; // Skip next argument as it's the value
        }
    }
    await runFusionCommand(options);
}

// Auto-detect command or default to fusion for better UX
// Smart command detection: use Commander.js for explicit command
const args = process.argv.slice(2);
const hasKnownCommand = args.some(arg =>
```

```
    ['init', 'fusion', 'config-check', '--help', '-h', '--version
);

if (hasKnownCommand) {
    program.parse(process.argv); // Use Commander.js parsing
} else {
    await runDefaultCommand(); // Direct fusion execution with ma
}
```

📄 src/clicommands.ts

```
/**
 * CLI commands implementation
 */
import chalk from 'chalk';
import clipboardy from 'clipboardy';
import fs from 'fs-extra';
import path from 'path';
import { processFusion } from './fusion.js';
import { ConfigSchemaV1 } from './schema.js';
import { Config, FusionOptions } from './types.js';
import { defaultConfig, getExtensionsFromGroups, loadConfig } fro

/**
 * Run the fusion command
 * @param options Command options
 */
export async function runFusionCommand(options: { extensions?: st
    try {
        console.log(chalk.blue('🔄 Starting Fusion Process...'));

        const config = await loadConfig();

        if (options.root) {
            config.rootDirectory = options.root;
            console.log(chalk.yellow(`ℹ️ Using specified director
        }

        // Parse extension groups from command line (comma-separa
        let extensionGroups: string[] | undefined;
        if (options.extensions) {
```

```
            extensionGroups = options.extensions.split(',').map(e
            console.log(chalk.blue(`Using extension groups: ${ext
        }

        const fusionOptions: FusionOptions = { extensionGroups };
        const result = await processFusion(config, fusionOptions)

        if (result.success) {
            console.log(chalk.green(`✅ ${result.message}`));
            console.log(chalk.green(`📄 Generated files:`));

            if (config.generateText) {
                console.log(chalk.cyan(`   - ${config.generatedFi
            }
            if (config.generateMarkdown) {
                console.log(chalk.cyan(`   - ${config.generatedFi
            }
            if (config.generateHtml) {
                console.log(chalk.cyan(`   - ${config.generatedFi
            }
            if (config.generatePdf) {
                console.log(chalk.cyan(`   - ${config.generatedFi
            }

            // Copy fusion content to clipboard if enabled
            if (config.copyToClipboard === true && result.fusionF
                try {
                    const fusionContent = await fs.readFile(resul
                    await clipboardy.write(fusionContent);
                    console.log(chalk.blue(`📋 Fusion content cop
                } catch (clipboardError) {
                    console.warn(chalk.yellow(`⚠️ Could not copy
                }
            }

            console.log(chalk.gray(`📝 Log file available at: ${r
        } else {
            console.log(chalk.red(`❌ ${result.message}`));
            if (result.logFilePath) {
                console.log(chalk.gray(`📝 Check log file for det
            }
        }
    } catch (error) {
        console.error(chalk.red(`❌ Fusion process failed: ${erro
        process.exit(1);
    }
}
```

```
/**
 * Run the init command to initialize the config
 */
export async function runInitCommand(options: { force?: boolean }
    try {
        console.log(chalk.blue('🔄 Initializing Project Fusion...

        const configPath = path.resolve('./project-fusion.json');
        if (await fs.pathExists(configPath)) {
            if (!options.force) {
                console.log(chalk.yellow('⚠️ project-fusion.json
                console.log(chalk.yellow('Use --force to override
                process.exit(1);
            } else {
                console.log(chalk.yellow('⚠️ Overriding existing
            }
        }

        await fs.writeJson(configPath, defaultConfig, { spaces: 4

        console.log(chalk.green('✅ Project Fusion initialized su
        console.log(chalk.blue('📁 Created:'));
        console.log(chalk.cyan('  - ./project-fusion.json'));

        console.log(chalk.blue('\n📝 Next steps:'));
        console.log(chalk.cyan('  1. Review project-fusion.json a
        console.log(chalk.cyan('  2. Run fusion: project-fusion')
    } catch (error) {
        console.error(chalk.red(`❌ Initialization failed: ${erro
        process.exit(1);
    }
}

/**
 * Run the config-check command to validate configuration
 */
export async function runConfigCheckCommand(): Promise<void> {
    try {
        console.log(chalk.blue('🔍 Checking Project Fusion Config

        const configPath = path.resolve('./project-fusion.json');

        // Verify configuration file exists
        if (!await fs.pathExists(configPath)) {
            console.log(chalk.yellow('⚠️ No project-fusion.json f
            console.log(chalk.cyan('   Using default configuratio
```

```
        console.log(chalk.gray('   Run "project-fusion init"

        await displayConfigInfo(defaultConfig, true);
        return;
    }

    // Load and parse configuration
    let configContent: string;
    try {
        configContent = await fs.readFile(configPath, 'utf8')
    } catch (error) {
        console.log(chalk.red(`❌ Cannot read configuration f
        process.exit(1);
    }

    let parsedConfig: any;
    try {
        parsedConfig = JSON.parse(configContent);
    } catch (error) {
        console.log(chalk.red(`❌ Invalid JSON in configurati
        process.exit(1);
    }

    // Validate configuration against schema
    const validation = ConfigSchemaV1.safeParse(parsedConfig)

    if (!validation.success) {
        console.log(chalk.red('❌ Configuration validation fa

        // Show detailed validation errors
        // Display detailed validation errors with helpful co
        validation.error.issues.forEach((issue, index) => {
            const path = issue.path.length > 0 ? issue.path.j
            const value = issue.path.reduce((obj: any, key) =
            console.log(chalk.red(`   ${index + 1}. Path: ${c
            console.log(chalk.red(`      Error: ${issue.messa
            console.log(chalk.red(`      Current value: ${cha
            if (issue.code === 'invalid_type') {
                console.log(chalk.red(`      Expected: ${chal
            }
        });

        console.log(chalk.yellow('\n💡 Suggestions:'));
        console.log(chalk.cyan('   - Check your configuration
        console.log(chalk.cyan('   - Run "project-fusion init
        process.exit(1);
    }
```

```
        console.log(chalk.green('✅ Configuration is valid!'));
        await displayConfigInfo(validation.data, false);

    } catch (error) {
        console.error(chalk.red(`❌ Config check failed: ${error}
        process.exit(1);
    }
}

/**
 * Display comprehensive configuration summary with preview
 */
async function displayConfigInfo(config: Config, isDefault: boole
    console.log(chalk.blue('\n📋 Configuration Summary:'));

    if (isDefault) {
        console.log(chalk.gray('   (Using default configuration)\
    } else {
        console.log('');
    }

    // Core configuration settings
    console.log(chalk.cyan('🔧 Basic Settings:'));
    console.log(`   Schema Version: ${config.schemaVersion}`);
    console.log(`   Root Directory: ${config.rootDirectory}`);
    console.log(`   Scan Subdirectories: ${config.parseSubDirecto
    console.log(`   Use .gitignore: ${config.useGitIgnoreForExclu
    console.log(`   Copy to Clipboard: ${config.copyToClipboard ?
    console.log(`   Max File Size: ${config.maxFileSizeKB} KB`);

    // File generation options
    console.log(chalk.cyan('\n📄 Output Generation:'));
    console.log(`   Generated File Name: ${config.generatedFileNa
    console.log(`   Generate Text: ${config.generateText ? 'Yes'
    console.log(`   Generate Markdown: ${config.generateMarkdown
    console.log(`   Generate HTML: ${config.generateHtml ? 'Yes'
    console.log(`   Generate PDF: ${config.generatePdf ? 'Yes' :
    console.log(`   Log File: project-fusion.log`);

    // File type configuration
    console.log(chalk.cyan('\n📁 File Extension Groups:'));
    const totalExtensions = getExtensionsFromGroups(config);

    Object.entries(config.parsedFileExtensions).forEach(([group,
        if (extensions) {
            console.log(`   ${group}: ${extensions.length} extens
        }
    });
```

```
    console.log(chalk.gray(`   Total: ${totalExtensions.length} u

    // Pattern exclusions
    console.log(chalk.cyan('\n🚫 Ignore Patterns:'));
    if (config.ignorePatterns.length === 0) {
        console.log('   None defined');
    } else {
        config.ignorePatterns.slice(0, 10).forEach(pattern => {
            console.log(`   ${pattern}`);
        });
        if (config.ignorePatterns.length > 10) {
            console.log(chalk.gray(`   ... and ${config.ignorePat
        }
    }

    // Preview matching files using current configuration
    console.log(chalk.cyan('\n🔍 File Discovery Preview:'));
    try {
        const { glob } = await import('glob');
        const rootDir = path.resolve(config.rootDirectory);

        // Create glob pattern to preview file discovery
        const allExtensionsPattern = totalExtensions.map(ext => e
        const pattern = config.parseSubDirectories
            ? `${rootDir}/**/*@(${allExtensionsPattern.join('|')}
            : `${rootDir}/*@(${allExtensionsPattern.join('|')})`;

        const filePaths = await glob(pattern, {
            nodir: true,
            follow: false
        });

        console.log(`   Pattern: ${pattern}`);
        console.log(`   Files found: ${filePaths.length}`);

        if (filePaths.length > 0) {
            console.log(`   Sample files:`);
            filePaths.slice(0, 5).forEach(file => {
                const relativePath = path.relative(rootDir, file)
                console.log(`     ${relativePath}`);
            });
            if (filePaths.length > 5) {
                console.log(chalk.gray(`     ... and ${filePaths.
            }
        }
    } catch (error) {
        console.log(chalk.yellow(`   Could not preview files: ${e
```

```
    }
}
```

src/fusion.ts

```
/**
 * Fusion functionality - Optimized single-file-in-memory approac
 */
import { createWriteStream } from 'fs';
import fs from 'fs-extra';
import { glob } from 'glob';
import ignoreLib from 'ignore';
import path from 'path';
import puppeteer from 'puppeteer';
import { BenchmarkTracker } from './benchmark.js';
import { Config, FusionOptions, FusionResult, createFilePath } fr
import {
    ensureDirectoryExists,
    formatLocalTimestamp,
    formatTimestamp,
    getExtensionsFromGroups,
    getMarkdownLanguage,
    logConfigSummary,
    readFileContentWithSizeLimit,
    writeLog
} from './utils.js';

/**
 * Process fusion of files - Optimized memory-efficient version
 * @param config Configuration
 * @param options Fusion options
 * @returns Fusion result
 */
export async function processFusion(
    config: Config,
    options: FusionOptions = {}
): Promise<FusionResult> {
    const benchmark = new BenchmarkTracker();

    try {
        // Extract parsing properties directly from flattened con
        const parsing = {
```

```
        rootDirectory: config.rootDirectory,
        parseSubDirectories: config.parseSubDirectories,
        maxFileSizeKB: config.maxFileSizeKB
    };
    const logFilePath = createFilePath(path.resolve('project-
    const fusionFilePath = createFilePath(path.resolve(`${con
    const mdFilePath = createFilePath(path.resolve(`${config.
    const htmlFilePath = createFilePath(path.resolve(`${confi
    const pdfFilePath = createFilePath(path.resolve(`${config
    const startTime = new Date();

    await fs.writeFile(logFilePath, '');
    await logConfigSummary(logFilePath, config);

    const extensions = getExtensionsFromGroups(config, option
    console.log(`Processing ${extensions.length} file extensi

    if (extensions.length === 0) {
        const message = 'No file extensions to process.';
        await writeLog(logFilePath, `Status: Fusion failed\nR
        return { success: false, message, logFilePath };
    }

    // Initialize ignore handler for filtering files based on
    const ig = ignoreLib();
    const rootDir = path.resolve(config.rootDirectory);

    // Load ignore patterns from .gitignore and custom config
    if (config.useGitIgnoreForExcludes) {
        const gitIgnorePath = path.join(rootDir, '.gitignore'
        if (await fs.pathExists(gitIgnorePath)) {
            const gitIgnoreContent = await fs.readFile(gitIgn
            ig.add(gitIgnoreContent);
        }
    }

    if (config.ignorePatterns.length > 0) {
        const patterns = config.ignorePatterns
            .filter(pattern => pattern.trim() !== '' && !patt
            .join('\n');
        ig.add(patterns);
    }

    // Create file discovery pattern based on extensions and
    // Build glob pattern for file discovery: ensure extensio
    const allExtensionsPattern = extensions.map(ext => ext.st
    const pattern = config.parseSubDirectories
        ? `${rootDir}/**/*@(${allExtensionsPattern.join('|')}
        : `${rootDir}/*@(${allExtensionsPattern.join('|')})`;
```

```
        let filePaths = await glob(pattern, {
            nodir: true,
            follow: false
        });
        const originalFileCount = filePaths.length;
        filePaths = filePaths.filter(file => {
            const relativePath = path.relative(rootDir, file);
            return !ig.ignores(relativePath);
        });
        console.log(`Found ${originalFileCount} files, ${filePath

        if (filePaths.length === 0) {
            const message = 'No files found to process.';
            const endTime = new Date();
            await writeLog(logFilePath, `Status: Fusion failed\nR
            return { success: false, message, logFilePath };
        }

        // Extract project metadata for the fusion header
        const projectName = path.basename(process.cwd());
        let packageName = "";
        let projectVersion = "";
        const packageJsonPath = path.join(process.cwd(), 'package
        if (await fs.pathExists(packageJsonPath)) {
            try {
                const packageJson = JSON.parse(await fs.readFile(
                if (packageJson.name) {
                    packageName = packageJson.name;
                }
                if (packageJson.version) {
                    projectVersion = packageJson.version;
                }
            } catch (error) {
                console.warn('Error reading package.json:', error
            }
        }

        // Sort files alphabetically for consistent output across
        filePaths.sort((a, b) => path.relative(rootDir, a).locale

        // Track extension usage for comprehensive reporting
        const foundExtensions = new Set<string>();
        const otherExtensions = new Set<string>();
        const allFilesPattern = config.parseSubDirectories ? `${r
        const allFiles = await glob(allFilesPattern, { nodir: tru

        const allConfiguredExtensions = Object.values(config.pars
        const configuredExtensionSet = new Set(allConfiguredExten
```

```
        // Discover unconfigured extensions for comprehensive rep
        for (const file of allFiles) {
            const relativePath = path.relative(rootDir, file);
            const ext = path.extname(file).toLowerCase();

            // Track extensions found in project but not configur
            if (ext && !ig.ignores(relativePath) && !configuredEx
                otherExtensions.add(ext);
            }
        }

        // Pre-process files: validate sizes and collect metadata
        const maxFileSizeKB = config.maxFileSizeKB;
        const filesToProcess: { path: string; relativePath: strin
        const skippedFiles: string[] = [];
        let skippedCount = 0;
        let totalSizeBytes = 0;

        for (const filePath of filePaths) {
            const relativePath = path.relative(rootDir, filePath)
            const fileExt = path.extname(filePath).toLowerCase();
            foundExtensions.add(fileExt);

            try {
                const stats = await fs.stat(filePath);
                const sizeKB = stats.size / 1024;
                totalSizeBytes += stats.size;

                if (sizeKB > maxFileSizeKB) {
                    skippedCount++;
                    skippedFiles.push(relativePath);
                    await writeLog(logFilePath, `Skipped large fi
                } else {
                    filesToProcess.push({ path: filePath, relativ
                }
            } catch (error) {
                await writeLog(logFilePath, `Error checking file
                console.error(`Error checking file ${filePath}:`,
            }
        }

        // Create output directories for enabled formats
        if (config.generateText) await ensureDirectoryExists(path
        if (config.generateMarkdown) await ensureDirectoryExists(
        if (config.generateHtml) await ensureDirectoryExists(path
        if (config.generatePdf) await ensureDirectoryExists(path.

        // Initialize write streams for concurrent file generatio
        const txtStream = config.generateText ? createWriteStream
```

```
        const mdStream = config.generateMarkdown ? createWriteStr
        const htmlStream = config.generateHtml ? createWriteStrea

        // PDF content collection (generated from HTML via puppet
        let pdfContent = '';

        // Generate format-specific headers with project metadata
        const projectTitle = packageName && packageName.toLowerCa
            ? `${projectName} / ${packageName}`
            : projectName;
        const versionInfo = projectVersion ? ` v${projectVersion}

        const txtHeader = `# Generated Project Fusion File\n` +
            `# Project: ${projectTitle}${versionInfo}\n` +
            `# Generated: ${formatLocalTimestamp()}\n` +
            `# UTC: ${formatTimestamp()}\n` +
            `# Files: ${filesToProcess.length}\n` +
            `# Generated by: project-fusion\n\n`;

        const mdHeader = `# Generated Project Fusion File\n\n` +
            `**Project:** ${projectTitle}${versionInfo}\n\n` +
            `**Generated:** ${formatLocalTimestamp()}\n\n` +
            `**UTC:** ${formatTimestamp()}\n\n` +
            `**Files:** ${filesToProcess.length}\n\n` +
            `**Generated by:** [project-fusion](https://github.co
            `---\n\n## 📁 Table of Contents\n\n`;

        const htmlHeader = `<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-sc
    <title>Project Fusion - ${projectTitle}${versionInfo}</title>
    <style>
        body { font-family: -apple-system, BlinkMacSystemFont, 'S
        .header { border-bottom: 2px solid #eee; padding-bottom:
        .file-section { margin-bottom: 40px; border: 1px solid #d
        .file-title { background: #f5f5f5; margin: -20px -20px 20
        pre { background: #f8f9fa; padding: 15px; border-radius:
        code { font-family: 'Monaco', 'Menlo', 'Ubuntu Mono', mon
        .toc { background: #f8f9fa; padding: 20px; border-radius:
        .toc ul { margin: 0; padding-left: 20px; }
        .toc a { text-decoration: none; color: #0366d6; }
        .toc a:hover { text-decoration: underline; }
    </style>
</head>
<body>
    <div class="header">
        <h1>Generated Project Fusion File</h1>
```

```
        <p><strong>Project:</strong> ${projectTitle}${versionInfo
        <p><strong>Generated:</strong> ${formatLocalTimestamp()}<
        <p><strong>UTC:</strong> ${formatTimestamp()}</p>
        <p><strong>Files:</strong> ${filesToProcess.length}</p>
        <p><strong>Generated by:</strong> <a href="https://github
    </div>
    <div class="toc">
        <h2>📁 Table of Contents</h2>
        <ul>
${filesToProcess.map(fileInfo => `            <li><a href="#${fil
        </ul>
    </div>`;

        if (txtStream) txtStream.write(txtHeader);
        if (mdStream) mdStream.write(mdHeader);
        if (htmlStream) htmlStream.write(htmlHeader);

        // Initialize PDF fallback content
        if (config.generatePdf) {
            pdfContent = `Generated Project Fusion File\n\nProjec
        }

        // Generate table of contents for markdown format
        if (mdStream) {
            for (const fileInfo of filesToProcess) {
                mdStream.write(`- [${fileInfo.relativePath}](#${f
            }
            mdStream.write(`\n---\n\n`);
        }

        // Stream-process files to maintain low memory footprint
        let processedCount = 0;
        for (const fileInfo of filesToProcess) {
            try {
                const content = await fs.readFile(fileInfo.path,
                const fileExt = path.extname(fileInfo.path).toLow
                const basename = path.basename(fileInfo.path);
                const language = getMarkdownLanguage(fileExt || b
                // Escape HTML entities for safe HTML output
                const escapedContent = content
                    .replace(/&/g, '&amp;')
                    .replace(/</g, '&lt;')
                    .replace(/>/g, '&gt;')
                    .replace(/"/g, '&quot;')
                    .replace(/'/g, '&#39;');

                // Generate plain text format with file separator
                if (txtStream) {
                    txtStream.write(`<!-- ======================
```

```
                txtStream.write(`<!-- FILE: ${fileInfo.relati
                txtStream.write(`<!-- ======================
                txtStream.write(`${content}\n\n`);
            }

            // Generate markdown format with syntax highlight
            if (mdStream) {
                mdStream.write(`## 📄 ${fileInfo.relativePath
                mdStream.write(`\`\`\`${language}\n`);
                mdStream.write(`${content}\n`);
                mdStream.write(`\`\`\`\n\n`);
            }

            // Generate HTML format with styled code blocks
            if (htmlStream) {
                // Create URL-safe anchor ID for navigation
                const fileAnchor = fileInfo.relativePath.repl
                htmlStream.write(`    <div class="file-sectio
                htmlStream.write(`        <div class="file-ti
                htmlStream.write(`            <h2>📄 ${fileIn
                htmlStream.write(`        </div>\n`);
                htmlStream.write(`        <pre><code class="$
                htmlStream.write(`    </div>\n\n`);
            }

            // Collect content for PDF fallback (text-based)
            if (config.generatePdf) {
                pdfContent += `\n${'='.repeat(60)}\nFILE: ${f
            }

            processedCount++;
            benchmark.markFileProcessed(fileInfo.size);
        } catch (error) {
            await writeLog(logFilePath, `Error processing fil
            console.error(`Error processing file ${fileInfo.p
        }
    }

    // Finalize HTML document structure
    if (htmlStream) {
        htmlStream.write(`</body>\n</html>`);
    }

    // Ensure all streams are properly closed before PDF gene
    if (txtStream) {
        await new Promise<void>((resolve, reject) => {
            txtStream.end((err: any) => err ? reject(err) : r
        });
    }
```

```
        if (mdStream) {
            await new Promise<void>((resolve, reject) => {
                mdStream.end((err: any) => err ? reject(err) : re
            });
        }
        if (htmlStream) {
            await new Promise<void>((resolve, reject) => {
                htmlStream.end((err: any) => err ? reject(err) :
            });
        }

        // Generate PDF using puppeteer to convert HTML to PDF
        // Generate PDF using headless browser for high-quality o
        if (config.generatePdf && config.generateHtml) {
            try {
                const browser = await puppeteer.launch({ headless
                const page = await browser.newPage();
                // Wait for all resources to load before PDF gene
                await page.setContent(await fs.readFile(htmlFileP
                await page.pdf({
                    path: pdfFilePath,
                    format: 'A4',
                    margin: { top: '1cm', bottom: '1cm', left: '1
                    printBackground: true // Include CSS backgrou
                });
                await browser.close();
            } catch (error) {
                // Graceful fallback to text-based PDF if puppete
                console.warn(`Warning: PDF generation failed: ${e
                console.warn('Fallback: Creating text-based PDF f
                await fs.writeFile(pdfFilePath, pdfContent, 'utf8
            }
        } else if (config.generatePdf) {
            // Fallback when HTML generation is disabled
            await fs.writeFile(pdfFilePath, pdfContent, 'utf8');
        }

        // Generate detailed completion report with metrics
        const message = `Fusion completed successfully. ${process
        const endTime = new Date();
        const duration = ((endTime.getTime() - startTime.getTime(
        const totalSizeMB = (totalSizeBytes / (1024 * 1024)).toFi

        await writeLog(logFilePath, `Status: Fusion completed suc
        await writeLog(logFilePath, `Start time: ${formatTimestam
        await writeLog(logFilePath, `End time: ${formatTimestamp(
        await writeLog(logFilePath, `Duration: ${duration}s`, tru
        await writeLog(logFilePath, `Total data processed: ${tota
```

```
        // Include performance benchmarks in log
        const metrics = benchmark.getMetrics();
        await writeLog(logFilePath, `\nPerformance Metrics:`, tru
        await writeLog(logFilePath, `  Memory Used: ${metrics.mem
        await writeLog(logFilePath, `  Throughput: ${metrics.thro
        await writeLog(logFilePath, `  Files/second: ${(metrics.f

        await writeLog(logFilePath, `Files found: ${originalFileC
        await writeLog(logFilePath, `Files processed successfully
        await writeLog(logFilePath, `Files skipped (too large): $
        await writeLog(logFilePath, `Files filtered out: ${origin

        await writeLog(logFilePath, `Max file size limit: ${maxFi

        if (skippedFiles.length > 0) {
            await writeLog(logFilePath, `Skipped files:`, true);
            for (const file of skippedFiles.slice(0, 10)) {
                await writeLog(logFilePath, `  ${file}`, true);
            }
            if (skippedFiles.length > 10) {
                await writeLog(logFilePath, `  ... and ${skippedF
            }
        }

        await writeLog(logFilePath, `File extensions actually pro
        const foundExtArray = Array.from(foundExtensions).sort();
        for (const ext of foundExtArray) {
            await writeLog(logFilePath, `  ${ext}`, true);
        }

        const ignoredExtensions = extensions.filter(ext => !Array
        if (ignoredExtensions.length > 0) {
            await writeLog(logFilePath, `Configured extensions wi
            for (const ext of ignoredExtensions.sort()) {
                await writeLog(logFilePath, `  ${ext}`, true);
            }
        }

        if (otherExtensions.size > 0) {
            await writeLog(logFilePath, `File extensions found in
            for (const ext of Array.from(otherExtensions).sort())
                await writeLog(logFilePath, `  ${ext}`, true);
            }
        }

        const generatedFormats = [];
        if (config.generateText) generatedFormats.push('.txt');
        if (config.generateMarkdown) generatedFormats.push('.md')
        if (config.generateHtml) generatedFormats.push('.html');
```

```
        if (config.generatePdf) generatedFormats.push('.pdf');

        return {
            success: true,
            message: `${message} Generated formats: ${generatedFo
            fusionFilePath: config.generateText ? fusionFilePath
            logFilePath
        };
    } catch (error) {
        const errorMessage = `Fusion process failed: ${error}`;
        console.error(errorMessage);

        try {
            const logFilePath = createFilePath(path.resolve('proj
            const endTime = new Date();
            await writeLog(logFilePath, `Status: Fusion failed du

            return {
                success: false,
                message: errorMessage,
                logFilePath,
                error: error as Error
            };
        } catch (logError) {
            console.error('Could not write to log file:', logErro
            return {
                success: false,
                message: errorMessage,
                error: error as Error
            };
        }
    }
}
```

src/index.ts

```
/**
 * Entry point for Project Fusion
 */

export { BenchmarkTracker, type BenchmarkMetrics } from './benchm
export { processFusion } from './fusion.js';
```

```
export * from './schema.js';
export * from './types.js';
export * from './utils.js';
```

src/schema.ts

```
/**
 * Configuration schema definitions for Project Fusion
 */
import { z } from 'zod';

/**
 * Schema for output generation configuration
 */
const OutputConfigSchema = z.object({
    generatedFileName: z.string().default("project-fusioned"),
    copyToClipboard: z.boolean().default(false),
    generateText: z.boolean().default(true),
    generateMarkdown: z.boolean().default(true),
    generateHtml: z.boolean().default(true),
    generatePdf: z.boolean().default(true),
});

/**
 * File extension groups with support for custom categories
 */
const ParsedFileExtensionsSchema = z.object({
    backend: z.array(z.string()).default([".cs", ".go", ".java",
    config: z.array(z.string()).default([".json", ".toml", ".xml"
    cpp: z.array(z.string()).default([".c", ".cc", ".cpp", ".h",
    scripts: z.array(z.string()).default([".bat", ".cmd", ".ps1",
    web: z.array(z.string()).default([".css", ".html", ".js", ".j
    godot: z.array(z.string()).default([".gd", ".cs", ".tscn", ".
    doc: z.array(z.string()).default([".md", ".rst", ".adoc"]),
}).and(z.record(z.string(), z.array(z.string())));

/**
 * Complete configuration schema for version 1
 */
export const ConfigSchemaV1 = z.object({
    schemaVersion: z.literal(1).default(1),
    generatedFileName: z.string().default("project-fusioned"),
```

```
    copyToClipboard: z.boolean().default(false),
    generateText: z.boolean().default(true),
    generateMarkdown: z.boolean().default(true),
    generateHtml: z.boolean().default(true),
    generatePdf: z.boolean().default(true),
    maxFileSizeKB: z.number().default(1024),
    parseSubDirectories: z.boolean().default(true),
    parsedFileExtensions: ParsedFileExtensionsSchema.default({
        backend: [".cs", ".go", ".java", ".php", ".py", ".rb", ".
        config: [".json", ".toml", ".xml", ".yaml", ".yml"],
        cpp: [".c", ".cc", ".cpp", ".h", ".hpp"],
        scripts: [".bat", ".cmd", ".ps1", ".sh"],
        web: [".css", ".html", ".js", ".jsx", ".svelte", ".ts", "
        godot: [".gd", ".cs", ".tscn", ".tres", ".cfg", ".import"
        doc: [".md", ".rst", ".adoc"]
    }),
    rootDirectory: z.string().default("."),
    ignorePatterns: z.array(z.string()).default([
        "project-fusion.json",
        "project-fusion.log",
        "project-fusioned.*",
        "node_modules/",
        "package-lock.json",
        "pnpm-lock.yaml",
        "yarn.lock",
        "dist/",
        "build/",
        "*.min.js",
        "*.min.css",
        ".env",
        ".env.*",
        "*.key",
        "*.pem",
        "**/credentials/*",
        "**/secrets/*",
        "*.log",
        "logs/",
        ".DS_Store",
        "Thumbs.db",
        ".vscode/",
        ".idea/",
        "*.swp",
        "*.swo",
        "*.zip",
        "*.tar",
        "*.tgz",
        "*.gz",
        "*.7z",
        "*.rar",
```

```
        "*.png",
        "*.jpg",
        "*.jpeg",
        "*.gif",
        "*.bmp",
        "*.ico",
        "*.svg",
        "*.webp",
        "*.pdf",
        "*.doc",
        "*.docx",
        "*.xls",
        "*.xlsx",
        "*.ppt",
        "*.pptx",
        "*.mp3",
        "*.mp4",
        "*.avi",
        "*.mov",
        "*.wmv",
        "*.flv",
        "*.wav",
        "*.flac",
        "*.unitypackage",
        "*.uasset",
        "*.fbx",
        "*.obj",
        "*.blend",
        "*.exe",
        "*.dll",
        "*.so",
        "*.dylib",
        "*.a",
        "*.o",
        "*.pyc",
        "*.pyo",
        "*.class",
        "*.jar",
        "*.war"
    ]),
    useGitIgnoreForExcludes: z.boolean().default(true),
});
```

src/types.ts

```
/**
 * Type definitions for the fusion functionality
 */

// Branded type for type-safe file path handling
export type FilePath = string & { readonly __brand: unique symbol

export const createFilePath = (path: string): FilePath => path as

/**
 * Main configuration interface
 */
export interface Config {
    copyToClipboard: boolean;
    generatedFileName: string;
    generateHtml: boolean;
    generateMarkdown: boolean;
    generatePdf: boolean;
    generateText: boolean;
    ignorePatterns: string[];
    maxFileSizeKB: number;
    parseSubDirectories: boolean;
    parsedFileExtensions: {
        backend?: string[];
        config?: string[];
        cpp?: string[];
        doc?: string[];
        godot?: string[];
        scripts?: string[];
        web?: string[];
        [key: string]: string[] | undefined;
    };
    rootDirectory: string;
    schemaVersion: number;
    useGitIgnoreForExcludes: boolean;
}

/**
 * Information about a file for fusion
 */
export interface FileInfo {
    path: FilePath;
    content: string;
```

```
}

/**
 * Options for the fusion process
 */
export interface FusionOptions {
    extensionGroups?: string[];
}

/**
 * Type-safe fusion result with discriminated union for success/f
 */
export type FusionResult =
    | {
        success: true;
        message: string;
        fusionFilePath: FilePath;
        logFilePath: FilePath;
    }
    | {
        success: false;
        message: string;
        logFilePath?: FilePath;
        error?: Error;
    };
```

src/utils.ts

```
/**
 * Utilities for Project Fusion
 */
import fs from 'fs-extra';
import path from 'path';
import { z } from 'zod';
import { ConfigSchemaV1 } from './schema.js';
import { Config, FilePath } from './types.js';


/**
 * Default configuration for Project Fusion
 */
export const defaultConfig = {
```

```
    copyToClipboard: false,
    generatedFileName: "project-fusioned",
    generateHtml: true,
    generateMarkdown: true,
    generatePdf: true,
    generateText: true,
    maxFileSizeKB: 1024,
    parseSubDirectories: true,
    parsedFileExtensions: {
        backend: [".cs", ".go", ".java", ".php", ".py", ".rb", ".
        config: [".json", ".toml", ".xml", ".yaml", ".yml"],
        cpp: [".c", ".cc", ".cpp", ".h", ".hpp"],
        doc: [".md", ".rst", ".adoc"],
        godot: [".gd", ".cs", ".tscn", ".tres", ".cfg", ".import"
        scripts: [".bat", ".cmd", ".ps1", ".sh"],
        web: [".css", ".html", ".js", ".jsx", ".svelte", ".ts", "
    },
    rootDirectory: ".",
    ignorePatterns: [
        "project-fusion.json",
        "project-fusion.log",
        "project-fusioned.*",
        "node_modules/",
        "package-lock.json",
        "pnpm-lock.yaml",
        "yarn.lock",
        "dist/",
        "build/",
        "*.min.js",
        "*.min.css",
        ".env",
        ".env.*",
        "*.key",
        "*.pem",
        "**/credentials/*",
        "**/secrets/*",
        "*.log",
        "logs/",
        ".DS_Store",
        "Thumbs.db",
        ".vscode/",
        ".idea/",
        "*.swp",
        "*.swo",
        // Binary files and archives
        "*.zip",
        "*.tar",
        "*.tgz",
        "*.gz",
```

```
        "*.7z",
        "*.rar",
        // Images
        "*.png",
        "*.jpg",
        "*.jpeg",
        "*.gif",
        "*.bmp",
        "*.ico",
        "*.svg",
        "*.webp",
        // Documents
        "*.pdf",
        "*.doc",
        "*.docx",
        "*.xls",
        "*.xlsx",
        "*.ppt",
        "*.pptx",
        // Media
        "*.mp3",
        "*.mp4",
        "*.avi",
        "*.mov",
        "*.wmv",
        "*.flv",
        "*.wav",
        "*.flac",
        // Game engine assets
        "*.unitypackage",
        "*.uasset",
        "*.fbx",
        "*.obj",
        "*.blend",
        // Compiled/Binary
        "*.exe",
        "*.dll",
        "*.so",
        "*.dylib",
        "*.a",
        "*.o",
        "*.pyc",
        "*.pyo",
        "*.class",
        "*.jar",
        "*.war"
    ],
    useGitIgnoreForExcludes: true,
    schemaVersion: 1
```

```
} as const satisfies Config;


/**
 * Load and validate configuration with fallback to defaults
 * @returns The loaded configuration or default config if invalid
 */
export async function loadConfig(): Promise<Config> {
    try {
        const configPath = path.resolve('./project-fusion.json');

        let configContent: string;
        try {
            configContent = await fs.readFile(configPath, 'utf8')
        } catch (error) {
            return defaultConfig;
        }

        const parsedConfig = JSON.parse(configContent);

        try {
            const validatedConfig = ConfigSchemaV1.parse(parsedCo
            return validatedConfig;
        } catch (zodError: unknown) {
            // Graceful degradation with detailed error reporting
            if (zodError instanceof z.ZodError) {
                console.error('Configuration validation failed (w
                zodError.issues.forEach((issue, index) => {
                    const path = issue.path.length > 0 ? issue.pa
                    const value = issue.path.reduce((obj: any, ke
                    console.error(`  ${index + 1}. Path: ${path}`
                    console.error(`     Error: ${issue.message}`)
                    console.error(`     Current value: ${JSON.str
                    if (issue.code === 'invalid_type') {
                        console.error(`     Expected type: ${(iss
                    }
                });
            } else {
                console.error('Unknown validation error (will use
            }
            return defaultConfig;
        }
    } catch (error) {
        const typedError = error instanceof Error ? error : new E

        console.error('Error loading configuration, will use defa
            message: typedError.message,
            stack: typedError.stack,
            context: 'loadConfig',
```

```
            configPath: path.resolve('./project-fusion.json')
        });

        return defaultConfig;
    }
}

/**
 * Create directory if it doesn't exist
 * @param directory Directory path
 */
export async function ensureDirectoryExists(directory: string): P
    await fs.ensureDir(directory);
}

/**
 * Write content to log file with optional console output
 * @param logFilePath Path to log file
 * @param content Content to log
 * @param append If true, append to existing file
 * @param consoleOutput If true, also display on console
 */
export async function writeLog(
    logFilePath: string,
    content: string,
    append: boolean = false,
    consoleOutput: boolean = false
): Promise<void> {
    try {
        await ensureDirectoryExists(path.dirname(logFilePath));
        if (append) {
            await fs.appendFile(logFilePath, content + '\n');
        } else {
            await fs.writeFile(logFilePath, content + '\n');
        }

        if (consoleOutput) {
            console.log(content);
        }
    } catch (error) {
        console.error('Error writing log:', error);
    }
}


/**
 * Format a timestamp
 * @param date Optional date to format, defaults to current date
 * @returns Formatted timestamp
```

```
 */
export function formatTimestamp(date?: Date): string {
    return (date || new Date()).toISOString();
}

/**
 * Format a local timestamp for display
 * @param date Optional date to format, defaults to current date
 * @returns Formatted local timestamp
 */
export function formatLocalTimestamp(date?: Date): string {
    const now = date || new Date();
    return now.toLocaleString('fr-FR', {
        year: 'numeric',
        month: '2-digit',
        day: '2-digit',
        hour: '2-digit',
        minute: '2-digit',
        second: '2-digit',
        timeZoneName: 'short'
    });
}

/**
 * Read file content
 * @param filePath Path to file
 * @returns File content
 */
export async function readFileContent(filePath: string): Promise<
    try {
        return await fs.readFile(filePath, 'utf8');
    } catch (error) {
        console.error(`Error reading file ${filePath}:`, error);
        throw error;
    }
}

/**
 * Read file with size validation to prevent memory issues
 * @param filePath Path to file
 * @param maxSizeKB Maximum file size in KB
 * @returns File content or null if file exceeds size limit
 */
export async function readFileContentWithSizeLimit(
    filePath: string,
    maxSizeKB: number
): Promise<{ content: string | null; skipped: boolean; size: numb
    try {
        const stats = await fs.stat(filePath);
```

```
        const sizeKB = stats.size / 1024;

        if (sizeKB > maxSizeKB) {
            console.log(`Skipping large file ${filePath} (${sizeK
            return { content: null, skipped: true, size: stats.si
        }

        const content = await fs.readFile(filePath, 'utf8');
        return { content, skipped: false, size: stats.size };
    } catch (error) {
        console.error(`Error reading file ${filePath}:`, error);
        throw error;
    }
}

/**
 * Write detailed configuration summary to log for debugging
 * @param logFilePath Path to log file
 * @param config Configuration to log
 */
export async function logConfigSummary(logFilePath: FilePath, con
    await writeLog(logFilePath, `Configuration Summary:`, true);
    await writeLog(logFilePath, `  Schema Version: ${config.schem
    await writeLog(logFilePath, `  Root Directory: ${config.rootD
    await writeLog(logFilePath, `  Scan Subdirectories: ${config.
    await writeLog(logFilePath, `  Use .gitignore: ${config.useGi
    await writeLog(logFilePath, `  Copy to Clipboard: ${config.co
    await writeLog(logFilePath, `  Max File Size: ${config.maxFil

    // Output files
    await writeLog(logFilePath, `  Generated File Name: ${config.
    await writeLog(logFilePath, `  Generate Text: ${config.genera
    await writeLog(logFilePath, `  Generate Markdown: ${config.ge
    await writeLog(logFilePath, `  Generate HTML: ${config.genera
    await writeLog(logFilePath, `  Generate PDF: ${config.generat

    // File type statistics
    const totalExtensions = getExtensionsFromGroups(config);
    await writeLog(logFilePath, `  Extension Groups: ${Object.key
    await writeLog(logFilePath, `  Total Extensions: ${totalExten

    // Exclusion pattern count
    await writeLog(logFilePath, `  Ignore Patterns: ${config.igno

    await writeLog(logFilePath, ``, true); // Empty line for sepa
}

/**
 * Write content to file
```

```
 * @param filePath Path to file
 * @param content Content to write
 */
export async function writeFileContent(filePath: string, content:
    try {
        await ensureDirectoryExists(path.dirname(filePath));
        await fs.writeFile(filePath, content);
    } catch (error) {
        console.error(`Error writing file ${filePath}:`, error);
        throw error;
    }
}

/**
 * Extract file extensions from configuration groups
 * @param config Config object
 * @param groups Extension groups to include (all if undefined)
 * @returns Array of file extensions
 */
export function getExtensionsFromGroups(
    config: Config,
    groups?: string[]
): string[] {
    // Return all extensions if no specific groups requested
    if (!groups || groups.length === 0) {
        return Object.values(config.parsedFileExtensions)
            .filter((extensions): extensions is string[] => Boole
            .flat();
    }

    // Collect extensions from specified groups with validation
    return groups.reduce((acc: string[], group: string) => {
        const extensions = config.parsedFileExtensions[group];
        if (extensions) {
            acc.push(...extensions);
        } else {
            console.warn(`Warning: Extension group '${group}' not
        }
        return acc;
    }, []);
}

/**
 * Map file extensions to syntax highlighting languages for markd
 * @param extensionOrBasename File extension or special basename
 * @returns Language identifier for syntax highlighting
 */
export function getMarkdownLanguage(extensionOrBasename: string):
    // Comprehensive mapping for syntax highlighting across multi
```

```
    const languageMap: Record<string, string> = {
        // Backend (alphabetized)
        '.cs': 'csharp',
        '.go': 'go',
        '.java': 'java',
        '.kt': 'kotlin',
        '.lua': 'lua',
        '.perl': 'perl',
        '.php': 'php',
        '.pl': 'perl',
        '.py': 'python',
        '.r': 'r',
        '.rb': 'ruby',
        '.rs': 'rust',
        '.scala': 'scala',
        '.swift': 'swift',

        // C/C++ (alphabetized)
        '.c': 'c',
        '.cc': 'cpp',
        '.cpp': 'cpp',
        '.cxx': 'cpp',
        '.h': 'c',
        '.hpp': 'cpp',
        '.hxx': 'cpp',

        // Config (alphabetized)
        '.env': 'bash',
        '.ini': 'ini',
        '.json': 'json',
        '.toml': 'toml',
        '.xml': 'xml',
        '.yaml': 'yaml',
        '.yml': 'yaml',

        // Database
        '.sql': 'sql',

        // Documentation (alphabetized)
        '.md': 'markdown',
        '.mdx': 'markdown',
        '.rst': 'rst',
        '.tex': 'latex',

        // Godot (alphabetized)
        '.cfg': 'ini',
        '.gd': 'gdscript',
        '.import': 'ini',
        '.tres': 'gdscript',
```

```
        '.tscn': 'gdscript',

        // Other (alphabetized)
        '.cmake': 'cmake',
        '.dockerfile': 'dockerfile',
        '.Dockerfile': 'dockerfile',
        '.gql': 'graphql',
        '.gradle': 'gradle',
        '.graphql': 'graphql',
        '.makefile': 'makefile',
        '.Makefile': 'makefile',
        '.proto': 'protobuf',

        // Shell/Scripts (alphabetized)
        '.bash': 'bash',
        '.bat': 'batch',
        '.cmd': 'batch',
        '.fish': 'bash',
        '.ps1': 'powershell',
        '.sh': 'bash',
        '.zsh': 'bash',

        // Web (alphabetized)
        '.css': 'css',
        '.html': 'html',
        '.js': 'javascript',
        '.jsx': 'jsx',
        '.less': 'less',
        '.sass': 'sass',
        '.scss': 'scss',
        '.svelte': 'svelte',
        '.ts': 'typescript',
        '.tsx': 'tsx',
        '.vue': 'vue',

        // Files without extensions (alphabetized by basename)
        '.gitattributes': 'text',
        '.gitignore': 'text',
        '.htaccess': 'apache',
        'Cargo.lock': 'toml',
        'Cargo.toml': 'toml',
        'CMakeLists.txt': 'cmake',
        'dockerfile': 'dockerfile',
        'Dockerfile': 'dockerfile',
        'Gemfile': 'ruby',
        'go.mod': 'go',
        'go.sum': 'text',
        'Jenkinsfile': 'groovy',
        'makefile': 'makefile',
```

```
        'Makefile': 'makefile',
        'nginx.conf': 'nginx',
        'Rakefile': 'ruby',
        'requirements.txt': 'text',
        'Vagrantfile': 'ruby',
    };

    // Case-insensitive lookup with fallback to 'text'
    const lang = languageMap[extensionOrBasename.toLowerCase()] |
    return lang || 'text';
}
```

tests/formats.test.ts

```
import { describe, it, expect, beforeEach, afterEach } from 'vite
import fs from 'fs-extra';
import path from 'path';
import { processFusion } from '../src/fusion.js';
import { Config } from '../src/types.js';

// Test configuration for multiple output formats
const testConfig: Config = {
  schemaVersion: 1,
  generatedFileName: 'test-output',
  copyToClipboard: false,
  generateText: true,
  generateMarkdown: true,
  generateHtml: true,
  generatePdf: true,
  parsedFileExtensions: {
    web: ['.js', '.ts'],
    doc: ['.md']
  },
  parsing: {
    parseSubDirectories: false,
    rootDirectory: '.',
    maxFileSizeKB: 1024
  },
  ignorePatterns: [],
  useGitIgnoreForExcludes: false
};
```

```
const testDir = path.resolve('./temp/test-formats');
const originalCwd = process.cwd();

describe('Multiple Format Generation', () => {
  beforeEach(async () => {
    await fs.ensureDir(testDir);
    process.chdir(testDir);

    // Create test files
    await fs.writeFile(path.join(testDir, 'test.js'), `
console.log('Hello World');
function greet(name) {
  return \`Hello, \${name}!\`;
}
export { greet };
    `.trim());

    await fs.writeFile(path.join(testDir, 'README.md'), `
# Test Project

This is a **test** project with _markdown_ content.

## Features
- Feature 1
- Feature 2
    `.trim());
  });

  afterEach(async () => {
    process.chdir(originalCwd);
    await fs.remove(testDir);
  });

  it('should generate text format when enabled', async () => {
    const config = { ...testConfig, generateText: true, generateM
    const result = await processFusion(config);

    expect(result.success).toBe(true);
    expect(await fs.pathExists('test-output.txt')).toBe(true);
    expect(await fs.pathExists('test-output.md')).toBe(false);
    expect(await fs.pathExists('test-output.html')).toBe(false);
    expect(await fs.pathExists('test-output.pdf')).toBe(false);
  });

  it('should generate markdown format when enabled', async () =>
    const config = { ...testConfig, generateText: false, generate
    const result = await processFusion(config);

    expect(result.success).toBe(true);
```

```
  expect(await fs.pathExists('test-output.txt')).toBe(false);
  expect(await fs.pathExists('test-output.md')).toBe(true);
  expect(await fs.pathExists('test-output.html')).toBe(false);
  expect(await fs.pathExists('test-output.pdf')).toBe(false);
});

it('should generate HTML format when enabled', async () => {
  const config = { ...testConfig, generateText: false, generate
  const result = await processFusion(config);

  expect(result.success).toBe(true);
  expect(await fs.pathExists('test-output.txt')).toBe(false);
  expect(await fs.pathExists('test-output.md')).toBe(false);
  expect(await fs.pathExists('test-output.html')).toBe(true);
  expect(await fs.pathExists('test-output.pdf')).toBe(false);
});

it('should generate PDF format when enabled', async () => {
  const config = { ...testConfig, generateText: false, generate
  const result = await processFusion(config);

  expect(result.success).toBe(true);
  expect(await fs.pathExists('test-output.txt')).toBe(false);
  expect(await fs.pathExists('test-output.md')).toBe(false);
  expect(await fs.pathExists('test-output.html')).toBe(false);
  expect(await fs.pathExists('test-output.pdf')).toBe(true);
});

it('should generate multiple formats when enabled', async () =>
  const result = await processFusion(testConfig);

  expect(result.success).toBe(true);
  expect(await fs.pathExists('test-output.txt')).toBe(true);
  expect(await fs.pathExists('test-output.md')).toBe(true);
  expect(await fs.pathExists('test-output.html')).toBe(true);
  expect(await fs.pathExists('test-output.pdf')).toBe(true);
});

it('should include proper HTML structure', async () => {
  const result = await processFusion(testConfig);

  expect(result.success).toBe(true);
  const htmlContent = await fs.readFile('test-output.html', 'ut

  expect(htmlContent).toContain('<!DOCTYPE html>');
  expect(htmlContent).toContain('<html lang="en">');
  expect(htmlContent).toContain('<title>Project Fusion - test-f
  expect(htmlContent).toContain('📁 Table of Contents');
  expect(htmlContent).toContain('📄 test.js');
```

```
    expect(htmlContent).toContain('📄 README.md');
    expect(htmlContent).toContain('</body>');
    expect(htmlContent).toContain('</html>');
  });

  it('should escape HTML in code content', async () => {
    // Add a file with HTML-like content
    await fs.writeFile('html-test.js', `
const html = '<div>Hello & <span>World</span></div>';
console.log(html);
    `.trim());

    const result = await processFusion(testConfig);

    expect(result.success).toBe(true);
    const htmlContent = await fs.readFile('test-output.html', 'ut

    expect(htmlContent).toContain('&lt;div&gt;Hello &amp; &lt;spa
  });

  it('should include proper PDF content', async () => {
    const result = await processFusion(testConfig);

    expect(result.success).toBe(true);
    const pdfContent = await fs.readFile('test-output.pdf', 'utf8

    expect(pdfContent).toContain('Generated Project Fusion File')
    expect(pdfContent).toContain('Project: test-formats');
    expect(pdfContent).toContain('Generated by: project-fusion');
    expect(pdfContent).toContain('FILE: test.js');
    expect(pdfContent).toContain('FILE: README.md');
    expect(pdfContent).toContain('console.log(\'Hello World\')');
  });

  it('should include proper metadata in generated files', async (
    // Create a package.json with version info
    await fs.writeFile('package.json', JSON.stringify({
      name: 'test-package',
      version: '1.0.0'
    }, null, 2));

    const configWithPackage = {
      ...testConfig,
      parsedFileExtensions: { ...testConfig.parsedFileExtensions,
    };

    const result = await processFusion(configWithPackage);

    expect(result.success).toBe(true);
```

```
    const txtContent = await fs.readFile('test-output.txt', 'utf8
    const mdContent = await fs.readFile('test-output.md', 'utf8')

    expect(txtContent).toContain('# Generated Project Fusion File
    expect(txtContent).toContain('# Project: test-formats / test-
    expect(txtContent).toContain('# Generated by: project-fusion'

    expect(mdContent).toContain('# Generated Project Fusion File'
    expect(mdContent).toContain('**Project:** test-formats / test
    expect(mdContent).toContain('[project-fusion](https://github.
  });
});
```

tests/integration.test.ts

```
import { describe, it, expect, beforeEach, afterEach } from 'vite
import fs from 'fs-extra';
import path from 'path';
import { processFusion } from '../src/fusion.js';
import { defaultConfig } from '../src/utils.js';
import { Config } from '../src/types.js';

describe('integration', () => {
  const testDir = path.join(process.cwd(), 'temp', 'test-integrat
  const originalCwd = process.cwd();

  beforeEach(async () => {
    // Clean up and create test directory
    await fs.remove(testDir);
    await fs.ensureDir(testDir);

    // Change to test directory
    process.chdir(testDir);
  });

  afterEach(async () => {
    // Restore original directory
    process.chdir(originalCwd);

    // Clean up test directory
    await fs.remove(testDir);
  });
```

```
describe('processFusion', () => {
  it('should process fusion successfully with test files', asyn
    // Create test files
    await fs.writeFile('test.js', 'console.log("Hello World");'
    await fs.writeFile('test.ts', 'const message: string = "Typ
    await fs.writeFile('Dockerfile', 'FROM node:18\nCOPY . .\nR

    // Create config for test
    const testConfig: Config = {
      ...defaultConfig,
      parsing: {
        rootDirectory: '.',
        parseSubDirectories: false
      },
      parsedFileExtensions: {
        web: ['.js', '.ts']
      }
    };

    const result = await processFusion(testConfig);

    expect(result.success).toBe(true);
    expect(result.message).toContain('2 files processed');
    expect(result.fusionFilePath).toBeDefined();

    // Check if fusion files were created
    expect(await fs.pathExists(result.fusionFilePath!)).toBe(tr
    expect(await fs.pathExists(result.fusionFilePath!.replace('

    // Check content of fusion file
    const fusionContent = await fs.readFile(result.fusionFilePa
    expect(fusionContent).toContain('test.js');
    expect(fusionContent).toContain('test.ts');
    expect(fusionContent).toContain('console.log("Hello World")
    expect(fusionContent).toContain('const message: string = "T
    expect(fusionContent).not.toContain('Dockerfile'); // Not i
  });

  it('should handle empty directory gracefully', async () => {
    const testConfig: Config = {
      ...defaultConfig,
      parsing: {
        rootDirectory: '.',
        parseSubDirectories: false
      }
    };

    const result = await processFusion(testConfig);
```

```
      expect(result.success).toBe(false);
      expect(result.message).toContain('No files found to process
    });

    it('should respect ignore patterns', async () => {
      // Create test files
      await fs.writeFile('test.js', 'console.log("Hello World");'
      await fs.writeFile('ignored.js', 'console.log("Should be ig

      const testConfig: Config = {
        ...defaultConfig,
        parsing: {
          rootDirectory: '.',
          parseSubDirectories: false
        },
        parsedFileExtensions: {
          web: ['.js']
        },
        ignorePatterns: ['ignored.js']
      };

      const result = await processFusion(testConfig);

      expect(result.success).toBe(true);
      expect(result.message).toContain('1 files processed'); // O

      const fusionContent = await fs.readFile(result.fusionFilePa
      expect(fusionContent).toContain('test.js');
      expect(fusionContent).not.toContain('ignored.js');
    });
  });
});
```

tests/schema.test.ts

```
import { describe, it, expect } from 'vitest';
import { ConfigSchemaV1 } from '../src/schema.js';
import { defaultConfig } from '../src/utils.js';

describe('schema', () => {
  describe('ConfigSchemaV1', () => {
```

```
    it('should validate default config', () => {
      const result = ConfigSchemaV1.safeParse(defaultConfig);
      expect(result.success).toBe(true);
    });

    it('should validate minimal valid config', () => {
      const minimalConfig = {
        schemaVersion: 1,
        generatedFileName: "test-fusion",
        copyToClipboard: false,
        generateText: true,
        generateMarkdown: true,
        generateHtml: false,
        generatePdf: false,
        parsedFileExtensions: {
          web: [".js", ".ts"]
        },
        parsing: {
          rootDirectory: ".",
          parseSubDirectories: true,
          maxFileSizeKB: 1024
        },
        ignorePatterns: [],
        useGitIgnoreForExcludes: true
      };

      const result = ConfigSchemaV1.safeParse(minimalConfig);
      expect(result.success).toBe(true);
    });

    it('should reject config with invalid schema version', () =>
      const invalidConfig = {
        ...defaultConfig,
        schemaVersion: 2
      };

      const result = ConfigSchemaV1.safeParse(invalidConfig);
      expect(result.success).toBe(false);
    });

    it('should reject config with missing required fields', () =>
      const invalidConfig = {
        schemaVersion: 1
        // Missing required fields
      };

      const result = ConfigSchemaV1.safeParse(invalidConfig);
      expect(result.success).toBe(false);
    });
```

```
    it('should reject config with invalid copyToClipboard type',
      const invalidConfig = {
        ...defaultConfig,
        copyToClipboard: "true" // Should be boolean
      };

      const result = ConfigSchemaV1.safeParse(invalidConfig);
      expect(result.success).toBe(false);
    });

    it('should validate config with copyToClipboard true', () =>
      const validConfig = {
        ...defaultConfig,
        copyToClipboard: true
      };

      const result = ConfigSchemaV1.safeParse(validConfig);
      expect(result.success).toBe(true);
    });

    it('should validate config with HTML generation enabled', ()
      const validConfig = {
        ...defaultConfig,
        generateHtml: true
      };

      const result = ConfigSchemaV1.safeParse(validConfig);
      expect(result.success).toBe(true);
    });

    it('should validate config with PDF generation enabled', () =
      const validConfig = {
        ...defaultConfig,
        generatePdf: true
      };

      const result = ConfigSchemaV1.safeParse(validConfig);
      expect(result.success).toBe(true);
    });
  });
});
```

tests/utils.test.ts

```
import { describe, it, expect, beforeEach, afterEach } from 'vite
import fs from 'fs-extra';
import path from 'path';
import {
  getMarkdownLanguage,
  getExtensionsFromGroups,
  formatTimestamp,
  formatLocalTimestamp,
  loadConfig,
  writeLog,
  ensureDirectoryExists,
  writeFileContent,
  readFileContent
} from '../src/utils.js';
import { defaultConfig } from '../src/utils.js';

describe('utils', () => {
  describe('getMarkdownLanguage', () => {
    it('should return correct language for file extensions', () =
      expect(getMarkdownLanguage('.ts')).toBe('typescript');
      expect(getMarkdownLanguage('.js')).toBe('javascript');
      expect(getMarkdownLanguage('.py')).toBe('python');
      expect(getMarkdownLanguage('.json')).toBe('json');
    });

    it('should return correct language for files without extensio
      expect(getMarkdownLanguage('Dockerfile')).toBe('dockerfile'
      expect(getMarkdownLanguage('Makefile')).toBe('makefile');
      expect(getMarkdownLanguage('Jenkinsfile')).toBe('groovy');
    });

    it('should return text for unknown extensions', () => {
      expect(getMarkdownLanguage('.unknown')).toBe('text');
      expect(getMarkdownLanguage('UnknownFile')).toBe('text');
    });

    it('should handle case insensitive extensions', () => {
      expect(getMarkdownLanguage('.TS')).toBe('typescript');
      expect(getMarkdownLanguage('.JS')).toBe('javascript');
    });
  });

  describe('getExtensionsFromGroups', () => {
```

```
    it('should return all extensions when no groups specified', (
      const extensions = getExtensionsFromGroups(defaultConfig);
      expect(extensions.length).toBeGreaterThan(0);
      expect(extensions).toContain('.js');
      expect(extensions).toContain('.py');
      expect(extensions).toContain('.json');
    });

    it('should return extensions for specific groups', () => {
      const extensions = getExtensionsFromGroups(defaultConfig, [
      expect(extensions).toContain('.js');
      expect(extensions).toContain('.ts');
      expect(extensions).toContain('.html');
      expect(extensions).not.toContain('.py');
    });

    it('should return extensions for multiple groups', () => {
      const extensions = getExtensionsFromGroups(defaultConfig, [
      expect(extensions).toContain('.js');
      expect(extensions).toContain('.py');
      expect(extensions).toContain('.go');
    });

    it('should handle unknown groups gracefully', () => {
      const extensions = getExtensionsFromGroups(defaultConfig, [
      expect(extensions).toEqual([]);
    });
  });

  describe('formatTimestamp', () => {
    it('should format current date when no date provided', () =>
      const timestamp = formatTimestamp();
      expect(timestamp).toMatch(/^\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\
    });

    it('should format provided date', () => {
      const date = new Date('2025-01-01T12:00:00.000Z');
      const timestamp = formatTimestamp(date);
      expect(timestamp).toBe('2025-01-01T12:00:00.000Z');
    });
  });

  describe('formatLocalTimestamp', () => {
    it('should format current date when no date provided', () =>
      const timestamp = formatLocalTimestamp();
      expect(timestamp).toMatch(/^\d{2}\/\d{2}\/\d{4} \d{2}:\d{2}
    });

    it('should format provided date', () => {
```

```
      const date = new Date('2025-01-01T12:00:00.000Z');
      const timestamp = formatLocalTimestamp(date);
      expect(timestamp).toContain('01/01/2025');
    });
  });

  describe('file operations', () => {
    const testDir = path.resolve('./temp/test-utils');
    const testFile = path.join(testDir, 'test.txt');

    beforeEach(async () => {
      await fs.ensureDir(testDir);
    });

    afterEach(async () => {
      await fs.remove(testDir);
    });

    describe('ensureDirectoryExists', () => {
      it('should create directory if it does not exist', async ()
        const newDir = path.join(testDir, 'new-dir');
        expect(await fs.pathExists(newDir)).toBe(false);

        await ensureDirectoryExists(newDir);
        expect(await fs.pathExists(newDir)).toBe(true);
      });

      it('should not fail if directory already exists', async ()
        await ensureDirectoryExists(testDir);
        // Should not throw
        await ensureDirectoryExists(testDir);
        expect(await fs.pathExists(testDir)).toBe(true);
      });
    });

    describe('writeFileContent', () => {
      it('should write content to file', async () => {
        const content = 'Hello World!';
        await writeFileContent(testFile, content);

        expect(await fs.pathExists(testFile)).toBe(true);
        const readContent = await fs.readFile(testFile, 'utf8');
        expect(readContent).toBe(content);
      });

      it('should create directory if it does not exist', async ()
        const nestedFile = path.join(testDir, 'nested', 'deep', '
        const content = 'Nested content';
```

```
      await writeFileContent(nestedFile, content);
      expect(await fs.pathExists(nestedFile)).toBe(true);
      const readContent = await fs.readFile(nestedFile, 'utf8')
      expect(readContent).toBe(content);
    });
  });

  describe('readFileContent', () => {
    it('should read file content', async () => {
      const content = 'Test content';
      await fs.writeFile(testFile, content);

      const readContent = await readFileContent(testFile);
      expect(readContent).toBe(content);
    });

    it('should throw error for non-existent file', async () =>
      const nonExistentFile = path.join(testDir, 'does-not-exis
      await expect(readFileContent(nonExistentFile)).rejects.to
    });
  });

  describe('writeLog', () => {
    it('should write log content to file', async () => {
      const logFile = path.join(testDir, 'test.log');
      const logContent = 'Log entry';

      await writeLog(logFile, logContent);
      expect(await fs.pathExists(logFile)).toBe(true);
      const content = await fs.readFile(logFile, 'utf8');
      expect(content).toBe(logContent + '\n');
    });

    it('should append log content when append is true', async (
      const logFile = path.join(testDir, 'test.log');
      const firstEntry = 'First entry';
      const secondEntry = 'Second entry';

      await writeLog(logFile, firstEntry);
      await writeLog(logFile, secondEntry, true);

      const content = await fs.readFile(logFile, 'utf8');
      expect(content).toBe(firstEntry + '\n' + secondEntry + '\
    });

    it('should overwrite log content when append is false', asy
      const logFile = path.join(testDir, 'test.log');
      const firstEntry = 'First entry';
      const secondEntry = 'Second entry';
```

```
      await writeLog(logFile, firstEntry);
      await writeLog(logFile, secondEntry, false);

      const content = await fs.readFile(logFile, 'utf8');
      expect(content).toBe(secondEntry + '\n');
    });
  });
});

describe('loadConfig', () => {
  const testDir = path.resolve('./temp/test-config');
  const configFile = path.join(testDir, 'project-fusion.json');

  beforeEach(async () => {
    await fs.ensureDir(testDir);
    // Set working directory to test directory
    process.chdir(testDir);
  });

  afterEach(async () => {
    // Restore original working directory
    process.chdir(path.resolve('./../../'));
    await fs.remove(testDir);
  });

  it('should return default config when no config file exists',
    const config = await loadConfig();
    expect(config).toEqual(defaultConfig);
  });

  it('should load valid config from file', async () => {
    const validConfig = {
      schemaVersion: 1,
      generatedFileName: 'custom-fusion',
      copyToClipboard: true,
      generateText: true,
      generateMarkdown: false,
      generateHtml: true,
      generatePdf: false,
      parsedFileExtensions: {
        web: ['.js', '.ts']
      },
      parsing: {
        parseSubDirectories: false,
        rootDirectory: '.',
        maxFileSizeKB: 512
      },
      ignorePatterns: ['*.log'],
```

```
        useGitIgnoreForExcludes: false
      };

      await fs.writeJson(configFile, validConfig);
      const config = await loadConfig();
      expect(config).toEqual(validConfig);
    });

    it('should return default config for invalid JSON', async ()
      await fs.writeFile(configFile, 'invalid json {');
      const config = await loadConfig();
      expect(config).toEqual(defaultConfig);
    });

    it('should return default config for invalid schema', async (
      const invalidConfig = {
        schemaVersion: 'invalid',
        invalidField: true
      };

      await fs.writeJson(configFile, invalidConfig);
      const config = await loadConfig();
      expect(config).toEqual(defaultConfig);
    });
  });
});
```

📄 TODO.md

```
# TODO - Project Fusion

Basé sur l'analyse du directeur de programmation, voici les tâche

## 🔥 Priorités HAUTES (Bloquantes)

### 1. Unification du modèle de configuration
- [x] **Choix définitif** : Structure aplatie (actuelle) vs imbri
- [x] Mettre à jour `CLAUDE.md` ligne 46 pour correspondre au cod
- [x] Vérifier cohérence entre `src/types.ts`, `src/schema.ts`, `
- [x] Audit complet : chercher toutes les références à l'ancienne
```

```
### 2. Refactor CLI Command Parsing
- [ ] Remplacer le parsing manuel dans `src/cli.ts` (lignes 45-74
- [ ] Implémenter une vraie **commande par défaut** avec Commande
- [ ] Supprimer `runDefaultCommand()` et la logique `hasKnownComm
- [ ] Tester que toutes les options `--extensions`, `--root` fonc

### 3. Configuration TypeScript stricte
- [ ] Ajouter dans `tsconfig.json` :
  - `"noUncheckedIndexedAccess": true`
  - `"exactOptionalPropertyTypes": true`
  - `"verbatimModuleSyntax": true`
  - `"useUnknownInCatchVariables": true`
  - `"noPropertyAccessFromIndexSignature": true`
- [ ] Corriger toutes les erreurs TS qui en résultent

### 4. ESLint & Code Quality
- [ ] Créer `.eslintrc.json` avec config TypeScript stricte
- [ ] Ajouter rules : `no-explicit-any`, `prefer-readonly`, etc.
- [ ] Ajouter script `"lint": "eslint src/**/*.ts"`
- [ ] Corriger tous les warnings ESLint

### 5. Documentation critique manquante
- [ ] Compléter `CHANGELOG.md` (format Keep a Changelog + SemVer)
- [ ] Finaliser le `package.json` (garder la structure moderne av

## ⚠️ Priorités MOYENNES (Important)

### 6. Optimisation des dépendances
- [ ] Rendre `puppeteer` optionnel avec import dynamique
- [ ] Message clair si `puppeteer` absent pour génération PDF
- [ ] Test des imports conditionnels

### 7. Sécurité & Robustesse
- [ ] Vérifier que tous les parcours glob utilisent `follow: fals
- [ ] ~~Corriger logique clipboard~~ ✅ **DÉJÀ CORRECT**
- [ ] Désactiver clipboard en environnement non-interactif (CI)

### 8. Tests d'intégration manquants
- [ ] Tests pour `--extensions` (filtrage)
- [ ] Tests pour `.gitignore` + `ignorePatterns`
- [ ] Tests pour `maxFileSizeKB` (skip gros fichiers)
- [ ] Tests pour parcours non-récursif
- [ ] Tests génération HTML/PDF (avec/sans puppeteer)

### 9. Documentation utilisateur
- [ ] Section **"Examples"** dans README avec captures
- [ ] Section **"Programmatic API"** (usage de `processFusion`)
- [ ] Badges CI et coverage dans README
- [ ] Documenter installation locale (`npm install` sans global)
```

```
### 10. Fichiers projet standards
- [ ] Créer `SECURITY.md`
- [ ] Créer `CODE_OF_CONDUCT.md`
- [ ] Créer `SUPPORT.md`
- [ ] Headers SPDX dans les sources (`// SPDX-License-Identifier:

## 💡 Priorités BASSES (Améliorations DX)

### 11. Types utilitaires avancés
- [ ] `NonEmptyArray<T>` pour groupes d'extensions
- [ ] `const assertions` sur les extensions par défaut
- [ ] `ExtensionGroup`, `FilePath` validation renforcée
- [ ] Hiérarchie `FusionError` avec codes/severity

### 12. API Fluent (bonus DX)
- [ ] `FusionBuilder` par-dessus `processFusion`
- [ ] Pattern method chaining pour configuration

### 13. Tests avancés
- [ ] Property-based tests avec `fast-check`
- [ ] Snapshot tests pour sortie MD/HTML
- [ ] Mocks typés du filesystem

### 14. Features async modernes
- [ ] `AbortController` pour annuler les runs longs
- [ ] Async generators si streaming nécessaire
- [ ] Web Streams pour pipe vers stdout/fichiers

### 15. Outillage de publication
- [ ] Script `npm pack --dry-run` dans DEVELOPMENT.md
- [ ] Script `npm publish --dry-run`
- [ ] Guide complet de publication

---

## 📊 État actuel vs "State of the Art"

| Domaine | État actuel | Cible | Gap |
|---------|-------------|-------|-----|
| **Code TS** | Bon (strict de base) | Excellent | Flags stricts
| **Architecture** | Très bon | Excellent | Unification config |
| **CLI UX** | Bon | Excellent | Default command |
| **Tests** | Base solide | Excellent | Tests intégration |
| **Docs** | Correct | Excellent | API + Examples |
| **npm ready** | Presque | Excellent | CHANGELOG + lint |

## 🎯 Milestone de release
```

```
**v1.0.0** : Compléter toutes les priorités HAUTES + MOYENNES 1-7

**v1.1.0** : Ajouter priorités MOYENNES 8-10 + BASSES sélectionné

---

*Dernière mise à jour : Analyse du directeur de programmation - 2
```

tsconfig.json

```
{
    "compilerOptions": {
        "target": "ES2022",
        "module": "NodeNext",
        "moduleResolution": "NodeNext",
        "esModuleInterop": true,
        "resolveJsonModule": true,
        "strict": true,
        "declaration": true,
        "skipLibCheck": true,
        "forceConsistentCasingInFileNames": true,
        "outDir": "./dist",
        "rootDir": "./src"
    },
    "include": ["src/**/*"],
    "exclude": ["node_modules", "dist"]
}
```

vitest.config.ts

```
import { defineConfig } from 'vitest/config';

export default defineConfig({
  test: {
```

```
    globals: true,
    environment: 'node',
    coverage: {
      provider: 'v8',
      reporter: ['text', 'json', 'html'],
      reportsDirectory: './coverage',
      include: ['src/**/*.ts'],
      exclude: [
        'src/**/*.d.ts',
        'src/cli.ts', // CLI entry point - harder to test
        'node_modules/**'
      ],
      thresholds: {
        global: {
          branches: 80,
          functions: 80,
          lines: 80,
          statements: 80
        }
      }
    }
  },
});
```