AI Chat Tools Documentation

Overview

The AI Chat Tools framework provides the AI assistant with advanced capabilities including file system access and code execution. This enables the AI to help with real development tasks while maintaining security through sandboxing and validation.

Features

1. File System Tools

The AI can perform various file system operations within the project directory:

- **read_file**: Read file contents
- **write_file**: Create or overwrite files
- **list_directory**: List directory contents
- **search_files**: Search for files by pattern
- **get_file_info**: Get detailed file metadata

2. Code Execution Tools

The AI can execute code in sandboxed environments:

- **execute_python**: Run Python scripts with timeout and resource limits
- **execute_javascript**: Run JavaScript in a VM sandbox
- **execute_shell**: Execute whitelisted shell commands
- **run_npm_command**: Run npm commands safely
- **analyze_code**: Analyze code for issues and improvements

3. Security Features

All operations are protected by multiple security layers:

- **Path Validation**: Only allowed directories are accessible
- **Command Whitelisting**: Only safe commands can be executed
- **Resource Limits**: Execution time and memory are capped
- **Content Sanitization**: File content is validated before writing
- **Audit Logging**: All operations are logged for review

Architecture

```
src/lib/ai-tools/
├── types.ts                  # Type definitions
├── index.ts                  # Tool registry and exports
├── file-system-tools.ts      # File system operations
├── code-execution-tools.ts   # Code execution operations
├── logger.ts                 # Audit logging
└── security/
    ├── config.ts             # Security configuration
    ├── validator.ts          # Input validation
    └── sandbox.ts            # Sandboxed execution
```

Usage

API Endpoint

The enhanced chat API is available at `/api/ai/tool-chat`:

```
POST /api/ai/tool-chat
{
  "message": "Read the package.json file",
  "enableTools": true,
  "useKnowledge": true,
  "useCodebase": true,
  "conversationHistory": []
}
```

Response Format

```
{
  "response": "Here's the content of package.json...",
  "model": "llama3.2:3b",
  "usedContext": true,
  "toolsEnabled": true,
  "toolCalls": [
    {
      "id": "tool_123",
      "name": "read_file",
      "parameters": { "path": "./package.json" }
    }
  ],
  "toolResults": [
    {
      "id": "tool_123",
      "name": "read_file",
      "success": true,
      "result": { "content": "...", "size": 1234 }
    }
  ]
}
```

Frontend Component

Use the `ToolChatInterface` component:

```
import ToolChatInterface from '@/components/ToolChatInterface'

export default function Page() {
  return <ToolChatInterface />
}
```

Security Configuration

Configuration is stored in `ai-tools-config.json`:

```
{
  "security": {
    "filesystem": {
      "allowedBasePaths": ["./src", "./public"],
      "blockedPaths": ["./node_modules", "./.git"],
      "maxFileSizeMB": 10,
      "allowedExtensions": [".ts", ".js", ".json"],
      "blockedExtensions": [".exe", ".dll"]
    },
    "codeExecution": {
      "allowedLanguages": ["python", "javascript", "bash"],
      "maxExecutionTimeMs": 30000,
      "maxMemoryMB": 512,
      "allowedCommands": ["ls", "cat", "grep"],
      "blockedCommands": ["rm", "sudo", "chmod"],
      "allowNetworkAccess": false
    }
  }
}
```

Tool Definitions

File System Tools

read_file

```
{
  name: 'read_file',
  parameters: {
    path: string,      // File path (required)
    encoding?: string  // Encoding (default: utf8)
  }
}
```

write_file

```
{
  name: 'write_file',
  parameters: {
    path: string,      // File path (required)
    content: string,   // Content to write (required)
    encoding?: string  // Encoding (default: utf8)
  }
}
```

list_directory

```
{
  name: 'list_directory',
  parameters: {
    path: string,        // Directory path (required)
    recursive?: boolean  // Recursive listing (default: false)
  }
}
```

search_files

```
{
  name: 'search_files',
  parameters: {
    path: string,        // Search directory (required)
    pattern: string,     // Search pattern (required)
    recursive?: boolean  // Recursive search (default: true)
  }
}
```

Code Execution Tools

execute_python

```
{
  name: 'execute_python',
  parameters: {
    code: string,      // Python code (required)
    timeout?: number,  // Timeout in ms (default: 30000)
    args?: string[]    // Command line arguments
  }
}
```

execute_javascript

```
{
  name: 'execute_javascript',
  parameters: {
    code: string,      // JavaScript code (required)
    timeout?: number   // Timeout in ms (default: 30000)
  }
}
```

execute_shell

```
{
  name: 'execute_shell',
  parameters: {
    command: string,   // Shell command (required)
    timeout?: number   // Timeout in ms (default: 30000)
  }
}
```

Example Conversations

Example 1: Reading a File

User: “What’s in the package.json file?”

AI: “Let me read that file for you.”

```
TOOL_CALL: read_file
PARAMETERS:
{
  "path": "./package.json"
}
```

Result: File content is returned and AI provides a summary.

Example 2: Creating a New File

User: “Create a new utility function file”

AI: “I’ll create a new utility file for you.”

```
TOOL_CALL: write_file
PARAMETERS:
{
  "path": "./src/lib/utils/helper.ts",
  "content": "export function helper() { ... }"
}
```

Example 3: Running Code

User: “Calculate the factorial of 10 using Python”

AI: “I’ll run a Python script to calculate that.”

```
TOOL_CALL: execute_python
PARAMETERS:
{
  "code": "import math\nprint(math.factorial(10))"
}
```

Security Best Practices

For Developers

1. **Review Configuration**: Regularly review `ai-tools-config.json`
2. **Monitor Logs**: Check `logs/ai-tools.log` for suspicious activity
3. **Update Whitelist**: Keep command whitelist minimal
4. **Test Security**: Regularly test security boundaries
5. **Audit Access**: Review who has access to the AI tools

For Users

1. **Verify Operations**: Review what the AI plans to do before approval
2. **Check Paths**: Ensure file paths are correct

3. **Review Code**: Check generated code before execution
4. **Report Issues**: Report any security concerns immediately

Troubleshooting

Common Issues

1. “Path is outside allowed directories”

Cause: Trying to access files outside configured paths

Solution: Update `allowedBasePaths` in config or use correct path

2. “Command contains blocked operation”

Cause: Attempting to use a blocked command

Solution: Use an allowed command or request whitelist update

3. “Execution timeout exceeded”

Cause: Code took too long to execute

Solution: Optimize code or increase timeout in config

4. “Cannot connect to Ollama”

Cause: Ollama service is not running

Solution: Start Ollama with `ollama serve`

Debug Mode

Enable detailed logging:

```
// In security/config.ts
general: {
  enableLogging: true,
  logPath: './logs/ai-tools.log'
}
```

View logs:

```
tail -f logs/ai-tools.log
```

API Reference

Tool Execution Context

```
interface ToolExecutionContext {
  workingDirectory: string;
  allowedPaths: string[];
  maxExecutionTime: number;
  maxMemoryMB: number;
  userId?: string;
  sessionId?: string;
}
```

Tool Execution Result

```
interface ToolExecutionResult {
  success: boolean;
  output?: any;
  error?: string;
  executionTime: number;
  resourceUsage?: {
    memoryMB: number;
    cpuPercent: number;
  };
  warnings?: string[];
}
```

Performance Considerations

1. **File Size Limits**: Large files (>10MB) are rejected
2. **Execution Timeouts**: Default 30s, max 120s
3. **Memory Limits**: Default 512MB per execution
4. **Concurrent Executions**: Limited to prevent resource exhaustion
5. **Log Rotation**: Logs are rotated to prevent disk fill

Future Enhancements

- [ ] Database query tools
- [ ] API integration tools
- [ ] Git operations tools
- [ ] Docker container management
- [ ] Real-time collaboration features
- [ ] Advanced code refactoring tools
- [ ] Automated testing tools
- [ ] Deployment automation

Support

For issues or questions:
1. Check this documentation
2. Review logs in `logs/ai-tools.log`
3. Check GitHub issues
4. Contact the development team

License

This tool framework is part of the Sports Bar TV Controller project and follows the same license.