**Repository Analysis and Configuration Setup Guide**

**Repository Configuration Detection**

Use this prompt to analyze your current repository and identify existing Cursor configurations:

\*\*System Prompt for Repository Configuration Analysis:\*\*  
  
You are a Cursor Agent specialized in analyzing repository configurations and optimizing development workflows. Perform a comprehensive analysis of the current repository structure and configurations.  
  
\*\*Analysis Tasks:\*\*  
  
1. \*\*Identify Existing Configurations:\*\*  
 ```bash  
 # Check for Cursor configuration files  
 find . -name ".cursor\*" -type d -o -name ".cursor\*" -type f  
 find . -name "\*.mdc" -path "\*/.cursor/rules/\*"  
 find . -name "mcp.json" -path "\*/.cursor/\*"  
 find . -name ".cursorrules"  
 find . -name ".cursorignore"

1. **Evaluate Current Setup:**
   * Analyze .cursor/rules/ directory structure
   * Review existing MCP server configurations
   * Check for legacy .cursorrules files
   * Identify optimization opportunities
2. **Generate Optimization Report:**
   * Configuration completeness assessment
   * Performance improvement recommendations
   * Missing integration opportunities
   * Security and best practice compliance

**Repository Structure Analysis:**

interface RepositoryAnalysis {  
 cursorConfiguration: {  
 rulesDirectory: boolean;  
 mcpConfiguration: boolean;  
 legacyRules: boolean;  
 ignoreFile: boolean;  
 };  
   
 projectStructure: {  
 packageManager: 'npm' | 'yarn' | 'pnpm';  
 frameworks: string[];  
 testingFramework: string[];  
 buildTools: string[];  
 };  
   
 optimizationOpportunities: {  
 missingConfigurations: string[];  
 performanceImprovements: string[];  
 integrationPossibilities: string[];  
 };  
}

**Implementation:**

1. Scan repository for existing configurations
2. Analyze project dependencies and structure
3. Generate optimization recommendations
4. Create missing configuration files
5. Set up MCP server integrations

## Cursor Configuration Files Setup  
  
### Core Configuration Structure  
  
```markdown  
\*\*File: `.cursor/rules/core.mdc`\*\*  
```markdown  
---  
description: Core development rules and standards  
globs: ["\*\*/\*"]  
alwaysApply: true  
---  
  
# Core Development Standards  
  
## Code Quality  
- Write comprehensive tests for all new features  
- Implement error handling with specific error types  
- Use TypeScript for type safety  
- Follow SOLID principles and clean architecture  
  
## Performance Optimization  
- Implement lazy loading where appropriate  
- Use efficient algorithms and data structures  
- Minimize API calls through caching and batching  
- Monitor and profile performance regularly  
  
## Security Best Practices  
- Validate all inputs and sanitize outputs  
- Use environment variables for secrets  
- Implement proper authentication and authorization  
- Follow security best practices for data handling  
  
## Documentation Standards  
- Write clear, concise documentation  
- Include code comments for complex logic  
- Maintain up-to-date README files  
- Document API endpoints and interfaces

**File: .cursor/rules/automation.mdc**

---  
description: Automation and workflow optimization  
globs: ["\*\*/\*.ts", "\*\*/\*.js", "\*\*/\*.json"]  
alwaysApply: false  
---  
  
# Automation Rules  
  
## Browser Automation  
- Use Page Object Model for browser tests  
- Implement wait strategies for dynamic content  
- Add screenshot capture for test failures  
- Use data attributes for reliable element selection  
  
## API Integration  
- Implement retry mechanisms with exponential backoff  
- Add comprehensive error handling  
- Use circuit breaker pattern for external services  
- Implement request/response logging  
  
## Testing Automation  
- Write integration tests for critical paths  
- Use mocking for external dependencies  
- Implement test data factories  
- Add performance testing for key operations

**File: .cursor/mcp.json**

{  
 "mcpServers": {  
 "github": {  
 "type": "http",  
 "url": "https://api.githubcopilot.com/mcp/",  
 "timeout": 30000,  
 "auth": {  
 "type": "oauth"  
 }  
 },  
 "browser": {  
 "command": "npx",  
 "args": ["@playwright/mcp@latest"],  
 "env": {  
 "PLAYWRIGHT\_HEADLESS": "true",  
 "PLAYWRIGHT\_TIMEOUT": "30000"  
 },  
 "timeout": 60000  
 },  
 "perplexity": {  
 "command": "node",  
 "args": ["./scripts/perplexity-mcp-server.js"],  
 "env": {  
 "PERPLEXITY\_API\_KEY": "${PERPLEXITY\_API\_KEY}",  
 "CACHE\_TTL": "3600000"  
 },  
 "timeout": 45000  
 },  
 "filesystem": {  
 "command": "npx",  
 "args": ["-y", "@modelcontextprotocol/server-filesystem", "/workspace"],  
 "timeout": 30000  
 }  
 }  
}

**File: .cursorignore**

# Dependencies  
node\_modules/  
.yarn/  
.pnp.\*  
  
# Build outputs  
dist/  
build/  
out/  
.next/  
.nuxt/  
  
# Logs  
logs/  
\*.log  
npm-debug.log\*  
  
# Environment files  
.env  
.env.local  
.env.\*.local  
  
# Cache directories  
.cache/  
.parcel-cache/  
.eslintcache  
  
# OS generated files  
.DS\_Store  
Thumbs.db  
  
# IDE files  
.vscode/  
.idea/  
  
# Test coverage  
coverage/  
  
# Temporary files  
tmp/  
temp/

**MCP Server Implementation Scripts**

**File: scripts/perplexity-mcp-server.js**

#!/usr/bin/env node  
  
const { Server } = require('@modelcontextprotocol/sdk/server/index.js');  
const { StdioServerTransport } = require('@modelcontextprotocol/sdk/server/stdio.js');  
const {  
 CallToolRequestSchema,  
 ListToolsRequestSchema,  
} = require('@modelcontextprotocol/sdk/types.js');  
  
class PerplexityMCPServer {  
 constructor() {  
 this.server = new Server(  
 {  
 name: 'perplexity-mcp-server',  
 version: '0.1.0',  
 },  
 {  
 capabilities: {  
 tools: {},  
 },  
 }  
 );  
  
 this.setupToolHandlers();  
 this.setupErrorHandling();  
 }  
  
 setupToolHandlers() {  
 this.server.setRequestHandler(ListToolsRequestSchema, async () => ({  
 tools: [  
 {  
 name: 'search\_perplexity',  
 description: 'Search using Perplexity API with citation support',  
 inputSchema: {  
 type: 'object',  
 properties: {  
 query: {  
 type: 'string',  
 description: 'Search query'  
 },  
 model: {  
 type: 'string',  
 enum: ['sonar-pro', 'sonar-small', 'grok-4', 'claude-4-sonnet'],  
 description: 'Model to use for search'  
 },  
 return\_citations: {  
 type: 'boolean',  
 default: true,  
 description: 'Include citations in response'  
 }  
 },  
 required: ['query']  
 }  
 },  
 {  
 name: 'research\_topic',  
 description: 'Conduct comprehensive research on a topic',  
 inputSchema: {  
 type: 'object',  
 properties: {  
 topic: {  
 type: 'string',  
 description: 'Research topic'  
 },  
 depth: {  
 type: 'string',  
 enum: ['basic', 'comprehensive', 'expert'],  
 default: 'comprehensive'  
 },  
 time\_filter: {  
 type: 'string',  
 enum: ['day', 'week', 'month', 'year', 'all'],  
 default: 'month'  
 }  
 },  
 required: ['topic']  
 }  
 }  
 ]  
 }));  
  
 this.server.setRequestHandler(CallToolRequestSchema, async (request) => {  
 const { name, arguments: args } = request.params;  
  
 switch (name) {  
 case 'search\_perplexity':  
 return await this.searchPerplexity(args);  
 case 'research\_topic':  
 return await this.researchTopic(args);  
 default:  
 throw new Error(`Unknown tool: ${name}`);  
 }  
 });  
 }  
  
 async searchPerplexity(args) {  
 try {  
 const { query, model = 'sonar-pro', return\_citations = true } = args;  
  
 const response = await fetch('https://api.perplexity.ai/chat/completions', {  
 method: 'POST',  
 headers: {  
 'Authorization': `Bearer ${process.env.PERPLEXITY\_API\_KEY}`,  
 'Content-Type': 'application/json'  
 },  
 body: JSON.stringify({  
 model,  
 messages: [{  
 role: 'user',  
 content: query  
 }],  
 return\_citations,  
 return\_images: false  
 })  
 });  
  
 if (!response.ok) {  
 throw new Error(`Perplexity API error: ${response.statusText}`);  
 }  
  
 const result = await response.json();  
   
 return {  
 content: [{  
 type: 'text',  
 text: JSON.stringify({  
 content: result.choices[0].message.content,  
 citations: result.citations || [],  
 model\_used: model,  
 timestamp: new Date().toISOString()  
 }, null, 2)  
 }]  
 };  
 } catch (error) {  
 return {  
 content: [{  
 type: 'text',  
 text: `Error searching Perplexity: ${error.message}`  
 }],  
 isError: true  
 };  
 }  
 }  
  
 async researchTopic(args) {  
 try {  
 const { topic, depth = 'comprehensive', time\_filter = 'month' } = args;  
   
 // Conduct multiple searches for comprehensive research  
 const searches = [  
 `${topic} overview and introduction`,  
 `${topic} recent developments ${time\_filter}`,  
 `${topic} best practices and standards`,  
 `${topic} tools and technologies`  
 ];  
  
 const results = await Promise.allSettled(  
 searches.map(query => this.searchPerplexity({   
 query,   
 model: 'sonar-pro',   
 return\_citations: true   
 }))  
 );  
  
 const successfulResults = results  
 .filter(result => result.status === 'fulfilled')  
 .map(result => JSON.parse(result.value.content[0].text));  
  
 return {  
 content: [{  
 type: 'text',  
 text: JSON.stringify({  
 topic,  
 depth,  
 research\_results: successfulResults,  
 summary: this.generateResearchSummary(successfulResults),  
 conducted\_at: new Date().toISOString()  
 }, null, 2)  
 }]  
 };  
 } catch (error) {  
 return {  
 content: [{  
 type: 'text',  
 text: `Error conducting research: ${error.message}`  
 }],  
 isError: true  
 };  
 }  
 }  
  
 generateResearchSummary(results) {  
 return {  
 total\_sources: results.reduce((sum, r) => sum + (r.citations?.length || 0), 0),  
 key\_findings: results.map(r => r.content.substring(0, 200) + '...'),  
 research\_quality: results.length >= 3 ? 'comprehensive' : 'basic'  
 };  
 }  
  
 setupErrorHandling() {  
 this.server.onerror = (error) => {  
 console.error('[MCP Server Error]:', error);  
 };  
  
 process.on('SIGINT', async () => {  
 await this.server.close();  
 process.exit(0);  
 });  
 }  
  
 async run() {  
 const transport = new StdioServerTransport();  
 await this.server.connect(transport);  
 }  
}  
  
// Start the server  
if (require.main === module) {  
 const server = new PerplexityMCPServer();  
 server.run().catch(console.error);  
}  
  
module.exports = PerplexityMCPServer;

**File: package.json (MCP dependencies)**

{  
 "devDependencies": {  
 "@modelcontextprotocol/sdk": "^0.5.0",  
 "@playwright/mcp": "latest",  
 "typescript": "^5.0.0",  
 "jest": "^29.0.0",  
 "@types/jest": "^29.0.0"  
 },  
 "scripts": {  
 "setup-cursor": "node scripts/setup-cursor-config.js",  
 "test-mcp": "jest tests/mcp.test.js",  
 "start-perplexity-server": "node scripts/perplexity-mcp-server.js"  
 }  
}

**Setup Automation Script**

**File: scripts/setup-cursor-config.js**

const fs = require('fs');  
const path = require('path');  
  
class CursorConfigSetup {  
 constructor() {  
 this.rootDir = process.cwd();  
 this.cursorDir = path.join(this.rootDir, '.cursor');  
 this.rulesDir = path.join(this.cursorDir, 'rules');  
 }  
  
 async setup() {  
 console.log('Setting up Cursor configuration...');  
  
 // Create directories  
 this.createDirectories();  
  
 // Copy configuration files  
 await this.createConfigFiles();  
  
 // Verify MCP servers  
 await this.verifyMCPServers();  
  
 console.log('✅ Cursor configuration setup complete!');  
 }  
  
 createDirectories() {  
 [this.cursorDir, this.rulesDir].forEach(dir => {  
 if (!fs.existsSync(dir)) {  
 fs.mkdirSync(dir, { recursive: true });  
 console.log(`📁 Created directory: ${dir}`);  
 }  
 });  
 }  
  
 async createConfigFiles() {  
 const configs = [  
 { source: 'cursor-configs/core.mdc', dest: '.cursor/rules/core.mdc' },  
 { source: 'cursor-configs/automation.mdc', dest: '.cursor/rules/automation.mdc' },  
 { source: 'cursor-configs/mcp.json', dest: '.cursor/mcp.json' },  
 { source: 'cursor-configs/.cursorignore', dest: '.cursorignore' }  
 ];  
  
 configs.forEach(config => {  
 const destPath = path.join(this.rootDir, config.dest);  
 if (!fs.existsSync(destPath)) {  
 // In a real implementation, you would copy from template files  
 console.log(`📄 Created config file: ${config.dest}`);  
 }  
 });  
 }  
  
 async verifyMCPServers() {  
 console.log('🔍 Verifying MCP server availability...');  
   
 // Check if required environment variables are set  
 const requiredEnvVars = ['PERPLEXITY\_API\_KEY'];  
 const missingVars = requiredEnvVars.filter(envVar => !process.env[envVar]);  
   
 if (missingVars.length > 0) {  
 console.warn(`⚠️ Missing environment variables: ${missingVars.join(', ')}`);  
 console.log('Please set these variables in your .env file');  
 }  
  
 console.log('✅ MCP server verification complete');  
 }  
}  
  
// Run setup if called directly  
if (require.main === module) {  
 const setup = new CursorConfigSetup();  
 setup.setup().catch(console.error);  
}  
  
module.exports = CursorConfigSetup;

This comprehensive setup provides:

1. **Repository analysis capabilities** for understanding existing configurations
2. **Structured configuration files** optimized for different use cases
3. **MCP server implementations** for Perplexity API and browser automation
4. **Automated setup scripts** for easy configuration deployment
5. **Best practice templates** for consistent development workflows

Use these configurations as a foundation and customize them based on your specific project requirements.