perplexity

검색 키워드:

- "MCP Model Context Protocol official specification"
- "Anthropic MCP GitHub repository latest"
- "MCP web search server implementation"
- "MCP brave search server example"
- "MCP google search server setup"
- "MCP filesystem server for personal drives"
- "MCP server typescript implementation guide"
- "Claude Desktop MCP integration tutorial"

검색 키워드:

- "Claude Canvas feature implementation details"
- "Gemini Notebook canvas functionality"
- "AI writing canvas collaborative editing"
- "Real-time text editing with AI suggestions"
- "Canvas mode UI/UX patterns AI assistants"
- "Collaborative document editing libraries JavaScript"
- "Monaco Editor vs CodeMirror for AI canvas"
- "Draft.js vs Slate.js for rich text editing"

검색 키워드:

- "Electron security best practices 2024"
- "Electron context isolation implementation"
- "Secure IPC communication Electron"
- "Electron preload script security"
- "Desktop app API key storage security"

- "SQLite full-text search FTS5 implementation"
- "Electron file system database encryption"
- "Personal OneDrive API OAuth desktop app"
- "Google Drive API personal storage desktop"
- "Local file encryption AES-256 Node.js"
- "Conversation backup restore strategies"
- "SQLite WAL mode vs normal mode performance"

검색 키워드:

- "MCP web search to research pipeline"
- "AI research assistant workflow patterns"
- "Multi-step research automation"
- "Source citation and reference management"
- "Knowledge graph construction from web search"
- "Research project organization patterns"

검색 키워드:
- "Project-based file organization patterns"
- "Workspace management desktop applications"
- "Project templates and scaffolding"
- "Nested project hierarchy database design"
- "Project sharing and collaboration features"

Based on the comprehensive research conducted, I'll create a detailed report covering all the architectural and implementation aspects of building a comprehensive AI research assistant with MCP integration, secure data management, and collaborative features.

포괄적 AI 연구 어시스턴트 아키텍처: MCP 통합, 보안 데이터 관리 및 협업 기능을 위한 완전한 구현 가이드

1. 개요 및 시스템 아키텍처

1.1 프로젝트 목표

본 연구는 Model Context Protocol (MCP)을 활용한 포괄적인 AI 연구 어시스턴트 시스템의 설계와 구현을 다룹니다. 이 시스템은 다음과 같은 핵심 기능들을 통합합니다:

- **MCP 기반 웹 검색 및 리서치 파이프라인**: 웹 검색, 콘텐츠 추출, 다중 소스 합성을 통한 자동화된 연구 워크플로우[1][2]
- **보안 데이터 저장 및 암호화**: AES-256 암호화와 Electron safeStorage를 활용한 안전한 로컬 데이터 관리[3][4]
- **프로젝트 기반 파일 조직**: 중첩된 계층 구조와 워크스페이스 관리를 통한 체계적인 연구 프로젝트 조직[5][6]
- **협업 및 공유 기능**: 클라우드 스토리지 연동과 실시간 협업을 위한 소셜 기능[7][8]

1.2 시스템 아키텍처 개요

시스템은 다음과 같은 주요 컴포넌트들로 구성됩니다:

```
AI Research Assistant
├── Frontend (Electron + React/Vue)
│   ├── Main Window (연구 인터페이스)
│   ├── Canvas Mode (실시간 편집)
│   └── Project Management (워크스페이스)
├── Backend Services
│   ├── MCP Server (검색 및 리서치)
│   ├── Data Manager (암호화 저장)
│   └── Collaboration Service (공유 및 동기화)
├── Database Layer
│   ├── SQLite (로컬 데이터)
│   ├── FTS5 (전문 검색)
│   └── Backup/Restore System
└── External Integrations
    ├── Cloud Storage (OneDrive/Google Drive)
    ├── Search APIs (Google/Brave)
    └── Collaboration Platforms
```

2. MCP (Model Context Protocol) 구현

2.1 MCP 개요 및 중요성

Model Context Protocol은 Anthropic에서 개발한 오픈 소스 프로토콜로, AI 애플리케이션이 외부 데이터 소스와 도구에 접근할 수 있는 표준화된 방법을 제공합니다[^9]. MCP의 핵심 구성요소는 다음과 같습니다:

- **MCP 서버**: API, 데이터베이스, 코드에 대한 브리지 역할을 하며, 데이터 소스를 호스트에 도구로 노출합니다
- **MCP 클라이언트**: 프로토콜을 사용하여 MCP 서버와 상호작용합니다
- **MCP 호스트**: 서버와 클라이언트 간의 통신을 관리하는 시스템입니다

2.2 웹 검색 MCP 서버 구현

2.2.1 기본 서버 구조 (TypeScript)

```
// src/search-server.ts
import { Server } from "@modelcontextprotocol/sdk/server/index.js";
import { StdioServerTransport } from "@modelcontextprotocol/sdk/server/stdio.js";
import {
  CallToolRequestSchema,
  ListToolsRequestSchema,
  Tool,
} from "@modelcontextprotocol/sdk/types.js";
import axios from "axios";

class WebSearchMCPServer {
  private server: Server;
  private braveApiKey: string;
  private googleApiKey: string;
  private googleCseId: string;

  constructor() {
    this.server = new Server(
      {
        name: "web-search-server",
        version: "1.0.0",
      },
      {
        capabilities: {
          tools: {},
        },
      }
    );

    this.braveApiKey = process.env.BRAVE_API_KEY || "";
    this.googleApiKey = process.env.GOOGLE_API_KEY || "";
    this.googleCseId = process.env.GOOGLE_CSE_ID || "";

    this.setupToolHandlers();
  }

  private setupToolHandlers(): void {
    // 도구 목록 핸들러
    this.server.setRequestHandler(ListToolsRequestSchema, async () => {
      return {
        tools: [
          {
            name: "brave_search",
            description: "Brave 검색 API를 사용하여 웹 검색을 수행합니다",
            inputSchema: {
              type: "object",
              properties: {
                query: {
                  type: "string",
                  description: "검색할 쿼리 문자열",
                },
                count: {
                  type: "number",
                  description: "반환할 결과 수 (기본값: 10)",
                  default: 10,
                },
                offset: {
                  type: "number",
                  description: "결과 오프셋 (기본값: 0)",
                  default: 0,
                },
              },
              required: ["query"],
            },
          },
          {
            name: "google_search",
            description: "Google Custom Search API를 사용하여 웹 검색을 수행합니다",
            inputSchema: {
              type: "object",
              properties: {
                query: {
                  type: "string",
                  description: "검색할 쿼리 문자열",
                },
                num: {
                  type: "number",
                  description: "반환할 결과 수 (기본값: 10)",
                  default: 10,
                },
                start: {
                  type: "number",
                  description: "검색 시작 인덱스 (기본값: 1)",
                  default: 1,
                },
              },
              required: ["query"],
            },
          },
          {
            name: "scrape_content",
            description: "지정된 URL에서 콘텐츠를 스크레이핑합니다",
            inputSchema: {
              type: "object",
              properties: {
                url: {
                  type: "string",
                  description: "스크레이핑할 URL",
                },
                selector: {
                  type: "string",
                  description: "CSS 선택자 (선택사항)",
                },
              },
              required: ["url"],
            },
          },
        ],
      };
    });

    // 도구 호출 핸들러
    this.server.setRequestHandler(CallToolRequestSchema, async (request) => {
      const { name, arguments: args } = request.params;

      switch (name) {
        case "brave_search":
          return await this.performBraveSearch(args);
        case "google_search":
          return await this.performGoogleSearch(args);
        case "scrape_content":
          return await this.scrapeContent(args);
        default:
          throw new Error(`Unknown tool: ${name}`);
      }
    });
  }

  private async performBraveSearch(args: any) {
    const { query, count = 10, offset = 0 } = args;

    try {
      const response = await axios.get("https://api.search.brave.com/res/v1/web/search",
        headers: {
          "X-Subscription-Token": this.braveApiKey,
          "Accept": "application/json",
        },
        params: {
          q: query,
          count,
          offset,
          search_lang: "ko",
          country: "KR",
        freshness: "pd", // 최근 일주일
      },
    });

    const results = response.data.web?.results || [];

    return {
      content: [
        {
          type: "text",
          text: JSON.stringify({
            query,
            total_results: results.length,
            results: results.map((result: any) => ({
              title: result.title,
              url: result.url,
              description: result.description,
              published: result.age,
              snippet: result.extra_snippets?.[^0] || result.description,
            })),
          }, null, 2),
        },
      ],
    };
  } catch (error) {
    throw new Error(`Brave Search API 오류: ${error.message}`);
  }
}

private async performGoogleSearch(args: any) {
  const { query, num = 10, start = 1 } = args;

  try {
    const response = await axios.get("https://www.googleapis.com/customsearch/v1", {
      params: {
        key: this.googleApiKey,
        cx: this.googleCseId,
        q: query,
        num,
        start,
        hl: "ko",
        gl: "kr",
      },
    });

    const items = response.data.items || [];

    return {
      content: [
        {
          type: "text",
          text: JSON.stringify({
            query,
            total_results: response.data.searchInformation?.totalResults || 0,
            search_time: response.data.searchInformation?.searchTime || 0,
            results: items.map((item: any) => ({
              title: item.title,
              url: item.link,
              snippet: item.snippet,
              displayLink: item.displayLink,
              formattedUrl: item.formattedUrl,
            })),
          }, null, 2),
        },
      ],
    };
  } catch (error) {
    throw new Error(`Google Search API 오류: ${error.message}`);
  }
}

private async scrapeContent(args: any) {
  const { url, selector } = args;

  try {
    // Puppeteer 또는 Playwright를 사용한 웹 스크레이핑
    const { chromium } = await import('playwright');
    const browser = await chromium.launch({ headless: true });
    const page = await browser.newPage();

    await page.goto(url, { waitUntil: 'networkidle' });

    let content: string;
    if (selector) {
      content = await page.textContent(selector) || "";
    } else {
      // 기본적으로 메인 콘텐츠 추출
      content = await page.evaluate(() => {
        // 불필요한 요소들 제거
        const scripts = document.querySelectorAll('script, style, nav, header, footer,
        scripts.forEach(el => el.remove());

        return document.body.innerText;
      });
    }

    await browser.close();

    return {
      content: [
        {
          type: "text",
          text: JSON.stringify({
            url,
            title: await page.title(),
            content: content.substring(0, 10000), // 10,000자로 제한
            extracted_at: new Date().toISOString(),
          }, null, 2),
        },
      ],
    };
  } catch (error) {
      throw new Error(`콘텐츠 스크레이핑 오류: ${error.message}`);
    }
  }

  public async run(): Promise<void> {
    const transport = new StdioServerTransport();
    await this.server.connect(transport);
  }
}

// 서버 실행
if (require.main === module) {
  const server = new WebSearchMCPServer();
  server.run().catch(console.error);
}
```

2.2.2 연구 파이프라인 MCP 서버

연구 워크플로우를 자동화하는 고급 MCP 서버를 구현합니다[^1]:

```
// src/research-pipeline-server.ts
import { Server } from "@modelcontextprotocol/sdk/server/index.js";
import { StdioServerTransport } from "@modelcontextprotocol/sdk/server/stdio.js";

interface ResearchStep {
  step: string;
  query: string;
  sources: string[];
  findings: string;
  confidence: number;
}

interface ResearchResult {
  topic: string;
  steps: ResearchStep[];
  synthesis: string;
  citations: string[];
  timestamp: string;
}

class ResearchPipelineMCPServer {
  private server: Server;
  private searchServer: WebSearchMCPServer;

  constructor() {
    this.server = new Server(
      {
        name: "research-pipeline-server",
        version: "1.0.0",
      },
      {
        capabilities: {
          tools: {},
        },
      }
    );

    this.searchServer = new WebSearchMCPServer();
    this.setupToolHandlers();
  }

  private setupToolHandlers(): void {
    this.server.setRequestHandler(ListToolsRequestSchema, async () => {
      return {
        tools: [
          {
            name: "deep_research",
            description: "주제에 대한 포괄적인 다단계 연구를 수행합니다",
            inputSchema: {
              type: "object",
              properties: {
                topic: {
                  type: "string",
                  description: "연구할 주제",
                },
                depth: {
                  type: "number",
                  description: "연구 깊이 (1-5, 기본값: 3)",
                  default: 3,
                },
                focus_areas: {
                  type: "array",
                  items: { type: "string" },
                  description: "집중할 영역들 (선택사항)",
                },
                output_format: {
                  type: "string",
                  enum: ["summary", "detailed", "academic"],
                  description: "출력 형식",
                  default: "detailed",
                },
              },
              required: ["topic"],
            },
          },
          {
            name: "build_knowledge_graph",
            description: "연구 결과로부터 지식 그래프를 구축합니다",
            inputSchema: {
              type: "object",
              properties: {
                research_data: {
                  type: "string",
                  description: "연구 데이터 (JSON 형식)",
                },
                relationship_types: {
                  type: "array",
                  items: { type: "string" },
                  description: "추출할 관계 유형들",
                },
              },
              required: ["research_data"],
            },
          },
        ],
      };
    });

    this.server.setRequestHandler(CallToolRequestSchema, async (request) => {
      const { name, arguments: args } = request.params;

      switch (name) {
        case "deep_research":
          return await this.conductDeepResearch(args);
        case "build_knowledge_graph":
          return await this.buildKnowledgeGraph(args);
        default:
          throw new Error(`Unknown tool: ${name}`);
      }
    });
  }

  private async conductDeepResearch(args: any): Promise<any> {
    const { topic, depth = 3, focus_areas = [], output_format = "detailed" } = args;

    const researchResult: ResearchResult = {
      topic,
      steps: [],
      synthesis: "",
      citations: [],
      timestamp: new Date().toISOString(),
    };

    // 1단계: 초기 검색 쿼리 생성
    const initialQueries = this.generateResearchQueries(topic, focus_areas);

    for (let step = 1; step <= depth; step++) {
      const stepResult: ResearchStep = {
        step: `Step ${step}`,
        query: initialQueries[step - 1] || `${topic} 연구 단계 ${step}`,
        sources: [],
        findings: "",
        confidence: 0,
      };

      // 각 단계에서 다중 소스 검색
      const searchResults = await this.performMultiSourceSearch(stepResult.query);
      stepResult.sources = searchResults.sources;

      // 콘텐츠 추출 및 분석
      const extractedContent = await this.extractAndAnalyzeContent(searchResults.urls);
      stepResult.findings = this.synthesizeFindings(extractedContent);
      stepResult.confidence = this.calculateConfidence(extractedContent);

      researchResult.steps.push(stepResult);

      // 다음 단계 쿼리 개선
```

```
      if (step < depth) {
        initialQueries[step] = this.refineQuery(stepResult.findings, topic);
      }
    }

    // 최종 합성
    researchResult.synthesis = this.synthesizeResearch(researchResult.steps);
    researchResult.citations = this.extractCitations(researchResult.steps);

    return {
      content: [
        {
          type: "text",
          text: this.formatResearchOutput(researchResult, output_format),
        },
      ],
    };
  }

  private generateResearchQueries(topic: string, focusAreas: string[]): string[] {
    const baseQueries = [
      `${topic} 개요 정의`,
      `${topic} 최신 연구 동향`,
      `${topic} 실제 사례 응용`,
      `${topic} 향후 전망 과제`,
      `${topic} 비교 분석`,
    ];

    if (focusAreas.length > 0) {
      return focusAreas.map(area => `${topic} ${area} 세부 분석`);
    }

    return baseQueries;
  }

  private async performMultiSourceSearch(query: string) {
    // Brave와 Google 검색을 병렬로 수행
    const [braveResults, googleResults] = await Promise.all([
      this.searchServer.performBraveSearch({ query, count: 5 }),
      this.searchServer.performGoogleSearch({ query, num: 5 }),
    ]);

    const braveData = JSON.parse(braveResults.content[^0].text);
    const googleData = JSON.parse(googleResults.content[^0].text);

    return {
      sources: [
        ...braveData.results.map((r: any) => r.url),
        ...googleData.results.map((r: any) => r.url),
      ],
      urls: [
        ...braveData.results.map((r: any) => r.url),
        ...googleData.results.map((r: any) => r.url),
      ].slice(0, 10), // 상위 10개 URL만 사용
    };
  }
```

```
private async extractAndAnalyzeContent(urls: string[]) {
  const contentPromises = urls.map(async (url) => {
    try {
      const result = await this.searchServer.scrapeContent({ url });
      return JSON.parse(result.content[^0].text);
    } catch (error) {
      console.warn(`콘텐츠 추출 실패: ${url}`, error);
      return null;
    }
  });

  const contents = await Promise.all(contentPromises);
  return contents.filter(Boolean);
}

private synthesizeFindings(contents: any[]): string {
  // 간단한 키워드 기반 합성 (실제로는 LLM API 사용 권장)
  const allText = contents.map(c => c.content).join(" ");
  const sentences = allText.split(/[.!?]+/).filter(s => s.trim().length > 50);

  // 중요도 기반 문장 선별 (TF-IDF 또는 유사한 알고리즘 사용 가능)
  const importantSentences = sentences
    .slice(0, 10)
    .map(s => s.trim())
    .filter(s => s.length > 0);

  return importantSentences.join(". ");
}

private calculateConfidence(contents: any[]): number {
  // 소스의 수, 내용의 일관성 등을 기반으로 신뢰도 계산
  const sourceCount = contents.length;
  const avgContentLength = contents.reduce((acc, c) => acc + c.content.length, 0) / sou

  let confidence = Math.min(sourceCount / 10, 1) * 0.5; // 소스 수 기반 (최대 50%)
  confidence += Math.min(avgContentLength / 5000, 1) * 0.3; // 내용 길이 기반 (최대 30%)
  confidence += 0.2; // 기본 신뢰도

  return Math.round(confidence * 100);
}

private refineQuery(findings: string, originalTopic: string): string {
  // 이전 결과를 바탕으로 다음 쿼리 개선
  const keywords = this.extractKeywords(findings);
  return `${originalTopic} ${keywords.slice(0, 3).join(" ")} 심화 분석`;
}

private extractKeywords(text: string): string[] {
  // 간단한 키워드 추출 (실제로는 더 정교한 NLP 사용)
  const words = text.toLowerCase().match(/\b\w{4,}\b/g) || [];
  const frequency = words.reduce((acc, word) => {
    acc[word] = (acc[word] || 0) + 1;
    return acc;
  }, {} as Record<string, number>);
```

```
    return Object.entries(frequency)
      .sort(([,a], [,b]) => b - a)
      .slice(0, 10)
      .map(([word]) => word);
  }

  private synthesizeResearch(steps: ResearchStep[]): string {
    const findings = steps.map(s => s.findings).join("\n\n");
    const avgConfidence = steps.reduce((acc, s) => acc + s.confidence, 0) / steps.length;

    return `
연구 종합 (신뢰도: ${Math.round(avgConfidence)}%)

주요 발견사항:
${findings}

결론:
본 연구를 통해 ${steps[^0].query.split(" ")[^0]}에 대한 포괄적인 이해를 얻었습니다.
${steps.length}단계의 체계적인 조사를 통해 다양한 관점에서 분석한 결과,
평균 ${Math.round(avgConfidence)}%의 신뢰도를 보이는 종합적인 결론에 도달했습니다.
    `.trim();
  }

  private extractCitations(steps: ResearchStep[]): string[] {
    return steps.flatMap(step => step.sources).filter((url, index, arr) => arr.indexOf(ur
  }

  private formatResearchOutput(result: ResearchResult, format: string): string {
    switch (format) {
      case "summary":
        return `# ${result.topic} 연구 요약\n\n${result.synthesis}\n\n## 참고 자료\n${resul

      case "academic":
        return this.formatAcademicOutput(result);

      default: // detailed
        return JSON.stringify(result, null, 2);
    }
  }

  private formatAcademicOutput(result: ResearchResult): string {
    return `
# ${result.topic}에 대한 연구 보고서

## 초록
본 연구는 "${result.topic}"에 대한 포괄적인 분석을 수행하였다. ${result.steps.length}단계의 체계

## 1. 서론
${result.steps[^0]?.findings || "연구 배경 및 목적을 설명한다."}

## 2. 문헌 조사
${result.steps.map((step, i) => `### 2.${i+1} ${step.step}\n${step.findings}`).join("\n\n

## 3. 종합 분석
${result.synthesis}
```

```
## 4. 결론 및 제언
본 연구를 통해 ${result.topic}의 현황과 전망을 종합적으로 분석하였다. 향후 연구에서는 보다 구체적인

## 참고문헌
${result.citations.map((citation, i) => `[${i+1}] ${citation}`).join("\n")}

*연구 수행일: ${new Date(result.timestamp).toLocaleDateString("ko-KR")}*
    `.trim();
  }

  private async buildKnowledgeGraph(args: any): Promise<any> {
    const { research_data, relationship_types = ["related_to", "causes", "includes", "af

    try {
      const data = JSON.parse(research_data);
      const graph = this.constructKnowledgeGraph(data, relationship_types);

      return {
        content: [
          {
            type: "text",
            text: JSON.stringify(graph, null, 2),
          },
        ],
      };
    } catch (error) {
      throw new Error(`지식 그래프 구성 오류: ${error.message}`);
    }
  }

  private constructKnowledgeGraph(data: any, relationshipTypes: string[]) {
    const nodes: Array<{id: string, label: string, type: string}> = [];
    const edges: Array<{from: string, to: string, type: string, weight: number}> = [];

    // 엔터티 추출 및 노드 생성
    const entities = this.extractEntities(data);
    entities.forEach((entity, index) => {
      nodes.push({
        id: `entity_${index}`,
        label: entity,
        type: "concept"
      });
    });

    // 관계 추출 및 엣지 생성
    for (let i = 0; i < entities.length; i++) {
      for (let j = i + 1; j < entities.length; j++) {
        const relationship = this.detectRelationship(entities[i], entities[j], data);
        if (relationship) {
          edges.push({
            from: `entity_${i}`,
            to: `entity_${j}`,
            type: relationship.type,
            weight: relationship.confidence
          });
        }
      }
    }

    return {
      nodes,
      edges,
      metadata: {
        total_nodes: nodes.length,
        total_edges: edges.length,
        created_at: new Date().toISOString()
      }
    };
  }

  private extractEntities(data: any): string[] {
    // 간단한 엔터티 추출 (실제로는 NER 모델 사용 권장)
    const text = JSON.stringify(data).toLowerCase();
    const commonEntities = [
      "기술", "연구", "개발", "시스템", "방법", "분석", "결과", "데이터",
      "모델", "알고리즘", "성능", "효율", "최적화", "구현", "설계"
    ];

    return commonEntities.filter(entity => text.includes(entity));
  }

  private detectRelationship(entity1: string, entity2: string, data: any) {
    const text = JSON.stringify(data).toLowerCase();
    const combinedText = `${entity1} ${entity2}`;

    if (text.includes(`${entity1}는 ${entity2}`) || text.includes(`${entity1}은 ${entity2
      return { type: "related_to", confidence: 0.8 };
    }

    if (text.includes(`${entity1} 때문에 ${entity2}`) || text.includes(`${entity1}로 인해 $
      return { type: "causes", confidence: 0.9 };
    }

    // 기본 관련성 검사
    const distance = this.calculateSemanticDistance(entity1, entity2, text);
    if (distance > 0.5) {
      return { type: "related_to", confidence: distance };
    }

    return null;
  }

  private calculateSemanticDistance(entity1: string, entity2: string, text: string): numb
    // 간단한 의미적 거리 계산 (실제로는 임베딩 사용 권장)
    const entity1Count = (text.match(new RegExp(entity1, "g")) || []).length;
    const entity2Count = (text.match(new RegExp(entity2, "g")) || []).length;
    const coOccurrence = (text.match(new RegExp(`${entity1}.*${entity2}|${entity2}.*${ent

    if (entity1Count === 0 || entity2Count === 0) return 0;

    return coOccurrence / Math.min(entity1Count, entity2Count);
  }
  public async run(): Promise<void> {
    const transport = new StdioServerTransport();
    await this.server.connect(transport);
  }
}
```

2.3 Claude Desktop 통합

2.3.1 설정 파일 구성

Claude Desktop과 MCP 서버를 연동하기 위한 설정 파일을 작성합니다[10][11]:

```
// claude_desktop_config.json (Windows: %APPDATA%\Claude\claude_desktop_config.json)
// claude_desktop_config.json (macOS: ~/Library/Application Support/Claude/claude_desktop
{
  "mcpServers": {
    "web-search": {
      "command": "node",
      "args": [
        "/absolute/path/to/search-server/build/index.js"
      ],
      "env": {
        "BRAVE_API_KEY": "your-brave-api-key",
        "GOOGLE_API_KEY": "your-google-api-key",
        "GOOGLE_CSE_ID": "your-google-cse-id"
      }
    },
    "research-pipeline": {
      "command": "node",
      "args": [
        "/absolute/path/to/research-pipeline-server/build/index.js"
      ],
      "env": {
        "OPENAI_API_KEY": "your-openai-api-key"
      }
    },
    "filesystem": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-filesystem",
        "/path/to/research/projects",
        "/path/to/documents",
        "/path/to/data"
      ]
    }
  }
}
```

2.3.2 개발자 모드 활성화 및 디버깅

Claude Desktop에서 MCP 서버의 상태를 모니터링하고 디버깅하기 위해 개발자 모드를 활성화합니다[^11]:

1. Claude Desktop 실행
2. 상단 메뉴: **도움말 > 개발자 모드 활성화**
3. 애플리케이션 재시작
4. **설정 > 개발자** 탭에서 MCP 서버 상태 확인

3. Electron 보안 및 데이터 관리

3.1 Electron 보안 모범 사례

3.1.1 Context Isolation 구현

Electron의 보안을 위해 컨텍스트 격리를 활성화하고 안전한 IPC 통신을 구현합니다[12][13]:

```
// main/main.ts
import { app, BrowserWindow, ipcMain } from 'electron';
import * as path from 'path';

class ElectronApp {
  private mainWindow: BrowserWindow | null = null;

  constructor() {
    this.init();
  }

  private init(): void {
    app.whenReady().then(() => {
      this.createWindow();
      this.setupIpcHandlers();
    });

    app.on('window-all-closed', () => {
      if (process.platform !== 'darwin') {
        app.quit();
      }
    });

    app.on('activate', () => {
      if (BrowserWindow.getAllWindows().length === 0) {
        this.createWindow();
      }
    });
  }

  private createWindow(): void {
    this.mainWindow = new BrowserWindow({
      width: 1400,
      height: 900,
      webPreferences: {
        nodeIntegration: false, // 노드 통합 비활성화
        contextIsolation: true, // 컨텍스트 격리 활성화
        enableRemoteModule: false, // 원격 모듈 비활성화
        preload: path.join(__dirname, 'preload.js'), // 프리로드 스크립트 사용
        sandbox: true, // 샌드박스 모드 활성화
        webSecurity: true, // 웹 보안 활성화
      },
    });

    this.mainWindow.loadFile('renderer/index.html');

    // 개발 모드에서만 DevTools 열기
    if (process.env.NODE_ENV === 'development') {
      this.mainWindow.webContents.openDevTools();
    }
  }

  private setupIpcHandlers(): void {
    // 안전한 API 호출 핸들러
    ipcMain.handle('secure-api-call', async (event, { endpoint, method, data }) => {
      // API 엔드포인트 검증
      const allowedEndpoints = [
        '/api/search',
        '/api/research',
        '/api/projects',
        '/api/files'
      ];

      if (!allowedEndpoints.includes(endpoint)) {
        throw new Error('Unauthorized endpoint');
      }

      try {
        return await this.makeSecureApiCall(endpoint, method, data);
      } catch (error) {
        console.error('API call failed:', error);
        throw error;
      }
    });

    // 파일 시스템 접근 핸들러
    ipcMain.handle('fs-operation', async (event, { operation, path, data }) => {
      const allowedOperations = ['read', 'write', 'delete', 'list'];
      const allowedPaths = [
        app.getPath('documents'),
        app.getPath('userData'),
      ];

      if (!allowedOperations.includes(operation)) {
        throw new Error('Unauthorized operation');
      }

      if (!this.isPathAllowed(path, allowedPaths)) {
        throw new Error('Unauthorized path access');
      }

      return await this.performFileOperation(operation, path, data);
    });

    // 암호화된 저장소 핸들러
    ipcMain.handle('secure-storage', async (event, { action, key, value }) => {
      switch (action) {
        case 'get':
          return await this.getSecureData(key);
        case 'set':
          return await this.setSecureData(key, value);
        case 'delete':
          return await this.deleteSecureData(key);
        default:
          throw new Error('Invalid storage action');
      }
    });
  }

  private async makeSecureApiCall(endpoint: string, method: string, data: any) {
    // API 호출 로직 구현
    const axios = await import('axios');

    const config = {
      method,
      url: `${process.env.API_BASE_URL}${endpoint}`,
      data,
      headers: {
        'Authorization': `Bearer ${await this.getApiToken()}`,
        'Content-Type': 'application/json',
      },
      timeout: 30000,
    };

    const response = await axios.default(config);
    return response.data;
  }

  private isPathAllowed(targetPath: string, allowedPaths: string[]): boolean {
    const normalizedTarget = path.normalize(targetPath);
    return allowedPaths.some(allowedPath => {
      const normalizedAllowed = path.normalize(allowedPath);
      return normalizedTarget.startsWith(normalizedAllowed);
    });
  }

  private async performFileOperation(operation: string, filePath: string, data?: any) {
    const fs = await import('fs/promises');

    switch (operation) {
      case 'read':
        return await fs.readFile(filePath, 'utf-8');
      case 'write':
        return await fs.writeFile(filePath, data);
      case 'delete':
        return await fs.unlink(filePath);
      case 'list':
        return await fs.readdir(filePath);
      default:
        throw new Error('Invalid file operation');
    }
  }

  private async getApiToken(): string {
    // 안전한 토큰 저장소에서 토큰 조회
    const { safeStorage } = await import('electron');

    if (!safeStorage.isEncryptionAvailable()) {
      throw new Error('Encryption not available');
    }

    try {
      const encryptedToken = await this.getSecureData('api_token');
      if (!encryptedToken) {
        throw new Error('No API token found');
      }

      const decryptedToken = safeStorage.decryptString(Buffer.from(encryptedToken, 'base6
      return decryptedToken;
    } catch (error) {
      throw new Error('Failed to retrieve API token');
    }
  }

  private async getSecureData(key: string): Promise<string | null> {
    // 구현 예정
    return null;
  }

  private async setSecureData(key: string, value: string): Promise<void> {
    // 구현 예정
  }

  private async deleteSecureData(key: string): Promise<void> {
    // 구현 예정
  }
}

new ElectronApp();
```

3.1.2 Preload 스크립트 구현

보안 컨텍스트 브리지를 통한 안전한 API 노출[^13]:

```
// main/preload.ts
import { contextBridge, ipcRenderer } from 'electron';

// 검증된 채널 목록
const validChannels = {
  invoke: [
    'secure-api-call',
    'fs-operation',
    'secure-storage',
    'mcp-request',
    'research-query',
    'project-management'
  ],
  on: [
    'research-progress',
    'mcp-response',
    'file-change',
    'project-update'
  ]
};

// 안전한 API 인터페이스 정의
interface SecureElectronAPI {
  // 연구 관련 API
  research: {
    performSearch: (query: string, options?: SearchOptions) => Promise<SearchResult>;
    conductDeepResearch: (topic: string, options?: ResearchOptions) => Promise<ResearchRe
    buildKnowledgeGraph: (data: any) => Promise<KnowledgeGraph>;
  };

  // 프로젝트 관리 API
  projects: {
    create: (project: ProjectConfig) => Promise<string>;
    list: () => Promise<Project[]>;
    get: (id: string) => Promise<Project>;
    update: (id: string, updates: Partial<Project>) => Promise<void>;
    delete: (id: string) => Promise<void>;
  };

  // 파일 시스템 API
  files: {
    read: (path: string) => Promise<string>;
    write: (path: string, content: string) => Promise<void>;
    list: (directory: string) => Promise<FileInfo[]>;
    search: (query: string, directory?: string) => Promise<FileSearchResult[]>;
  };

  // 보안 저장소 API
  storage: {
    get: (key: string) => Promise<string | null>;
    set: (key: string, value: string) => Promise<void>;
    delete: (key: string) => Promise<void>;
  };

  // 이벤트 리스너
  events: {
    on: (channel: string, callback: (...args: any[]) => void) => void;
    off: (channel: string, callback: (...args: any[]) => void) => void;
  };
}

// 타입 정의
```

```
interface SearchOptions {
  source?: 'brave' | 'google' | 'both';
  count?: number;
  language?: string;
  freshness?: string;
}

interface SearchResult {
  query: string;
  results: Array<{
    title: string;
    url: string;
    snippet: string;
    publishedDate?: string;
  }>;
  totalResults: number;
  searchTime: number;
}

interface ResearchOptions {
  depth?: number;
  focusAreas?: string[];
  outputFormat?: 'summary' | 'detailed' | 'academic';
  includeCitations?: boolean;
}

interface ResearchResult {
  topic: string;
  steps: Array<{
    step: string;
    query: string;
    findings: string;
    sources: string[];
    confidence: number;
  }>;
  synthesis: string;
  citations: string[];
  timestamp: string;
}

interface KnowledgeGraph {
  nodes: Array<{id: string, label: string, type: string}>;
  edges: Array<{from: string, to: string, type: string, weight: number}>;
  metadata: {
    totalNodes: number;
    totalEdges: number;
    createdAt: string;
  };
}

interface ProjectConfig {
  name: string;
  description?: string;
  template?: string;
  workspace?: string;
  tags?: string[];
```

```
}

interface Project {
  id: string;
  name: string;
  description: string;
  createdAt: string;
  updatedAt: string;
  workspace: string;
  tags: string[];
  structure: ProjectStructure;
}

interface ProjectStructure {
  directories: string[];
  files: string[];
  metadata: Record<string, any>;
}

interface FileInfo {
  name: string;
  path: string;
  size: number;
  modifiedDate: string;
  type: 'file' | 'directory';
}

interface FileSearchResult extends FileInfo {
  matches: Array<{
    line: number;
    content: string;
    context: string;
  }>;
  relevanceScore: number;
}

// 안전한 API 구현
const secureElectronAPI: SecureElectronAPI = {
  research: {
    async performSearch(query: string, options: SearchOptions = {}): Promise<SearchResult
      return await ipcRenderer.invoke('secure-api-call', {
        endpoint: '/api/search',
        method: 'POST',
        data: { query, options }
      });
    },

    async conductDeepResearch(topic: string, options: ResearchOptions = {}): Promise<Rese
      return await ipcRenderer.invoke('mcp-request', {
        server: 'research-pipeline',
        tool: 'deep_research',
        arguments: { topic, ...options }
      });
    },

    async buildKnowledgeGraph(data: any): Promise<KnowledgeGraph> {
      return await ipcRenderer.invoke('mcp-request', {
        server: 'research-pipeline',
        tool: 'build_knowledge_graph',
        arguments: { research_data: JSON.stringify(data) }
      });
    }
  },

  projects: {
    async create(project: ProjectConfig): Promise<string> {
      return await ipcRenderer.invoke('secure-api-call', {
        endpoint: '/api/projects',
        method: 'POST',
        data: project
      });
    },

    async list(): Promise<Project[]> {
      return await ipcRenderer.invoke('secure-api-call', {
        endpoint: '/api/projects',
        method: 'GET',
        data: null
      });
    },

    async get(id: string): Promise<Project> {
      return await ipcRenderer.invoke('secure-api-call', {
        endpoint: `/api/projects/${id}`,
        method: 'GET',
        data: null
      });
    },

    async update(id: string, updates: Partial<Project>): Promise<void> {
      await ipcRenderer.invoke('secure-api-call', {
        endpoint: `/api/projects/${id}`,
        method: 'PATCH',
        data: updates
      });
    },

    async delete(id: string): Promise<void> {
      await ipcRenderer.invoke('secure-api-call', {
        endpoint: `/api/projects/${id}`,
        method: 'DELETE',
        data: null
      });
    }
  },

  files: {
    async read(path: string): Promise<string> {
      return await ipcRenderer.invoke('fs-operation', {
        operation: 'read',
        path
      });
    },

    async write(path: string, content: string): Promise<void> {
      await ipcRenderer.invoke('fs-operation', {
        operation: 'write',
        path,
        data: content
      });
    },

    async list(directory: string): Promise<FileInfo[]> {
      return await ipcRenderer.invoke('fs-operation', {
        operation: 'list',
        path: directory
      });
    },

    async search(query: string, directory?: string): Promise<FileSearchResult[]> {
      return await ipcRenderer.invoke('secure-api-call', {
        endpoint: '/api/files/search',
        method: 'POST',
        data: { query, directory }
      });
    }
  },

  storage: {
    async get(key: string): Promise<string | null> {
      return await ipcRenderer.invoke('secure-storage', {
        action: 'get',
        key
      });
    },

    async set(key: string, value: string): Promise<void> {
      await ipcRenderer.invoke('secure-storage', {
        action: 'set',
        key,
        value
      });
    },

    async delete(key: string): Promise<void> {
      await ipcRenderer.invoke('secure-storage', {
        action: 'delete',
        key
      });
    }
  },

  events: {
    on(channel: string, callback: (...args: any[]) => void): void {
      if (validChannels.on.includes(channel)) {
        ipcRenderer.on(channel, callback);
      } else {
        throw new Error(`Invalid channel: ${channel}`);
      }
    },

    off(channel: string, callback: (...args: any[]) => void): void {
      if (validChannels.on.includes(channel)) {
        ipcRenderer.off(channel, callback);
      }
    }
  }
};

// 컨텍스트 브리지를 통해 안전하게 API 노출
contextBridge.exposeInMainWorld('electronAPI', secureElectronAPI);

// TypeScript 타입 선언 (renderer에서 사용)
declare global {
  interface Window {
    electronAPI: SecureElectronAPI;
  }
}
```

3.2 데이터 암호화 및 보안 저장

3.2.1 AES-256 암호화 구현

민감한 데이터를 위한 AES-256 암호화 시스템을 구현합니다[4][14]:

```
// src/security/encryption.ts
import * as crypto from 'crypto';
import { safeStorage } from 'electron';

const ALGORITHM = 'aes-256-cbc';
const IV_LENGTH = 16; // AES 블록 크기
const KEY_LENGTH = 32; // AES-256 키 길이

export interface EncryptionConfig {
  algorithm?: string;
  keyDerivationRounds?: number;
  saltLength?: number;
}

export interface EncryptedData {
  iv: string;
  salt: string;
  data: string;
  algorithm: string;
  keyDerivationRounds: number;
}

export class EncryptionManager {
  private masterKey: Buffer | null = null;
  private config: Required<EncryptionConfig>;

  constructor(config: EncryptionConfig = {}) {
    this.config = {
      algorithm: config.algorithm || ALGORITHM,
      keyDerivationRounds: config.keyDerivationRounds || 100000,
      saltLength: config.saltLength || 32,
    };
  }

  /**
   * 마스터 키 초기화 또는 로드
   */
  public async initializeMasterKey(password?: string): Promise<void> {
    if (safeStorage.isEncryptionAvailable()) {
      // Electron safeStorage 사용
      try {
        const storedKey = await this.getStoredMasterKey();
        if (storedKey) {
          this.masterKey = safeStorage.decryptString(storedKey);
        } else {
          // 새로운 마스터 키 생성
          this.masterKey = crypto.randomBytes(KEY_LENGTH);
          await this.storeMasterKey(this.masterKey);
        }
      } catch (error) {
        console.error('SafeStorage 초기화 실패:', error);
        // 패스워드 기반 키 파생으로 대체
        if (password) {
          await this.initializePasswordBasedKey(password);
        } else {
          throw new Error('Encryption initialization failed');
        }
      }
    } else {
      // 패스워드 기반 키 파생
      if (password) {
        await this.initializePasswordBasedKey(password);
      } else {
        throw new Error('Password required for encryption');
      }
    }
  }

  private async initializePasswordBasedKey(password: string): Promise<void> {
    const salt = crypto.randomBytes(this.config.saltLength);
    this.masterKey = crypto.pbkdf2Sync(
      password,
      salt,
      this.config.keyDerivationRounds,
      KEY_LENGTH,
      'sha256'
    );
  }

  private async getStoredMasterKey(): Promise<Buffer | null> {
    // 플랫폼별 보안 저장소에서 키 조회
    try {
      const fs = await import('fs/promises');
      const os = await import('os');
      const path = await import('path');

      const keyPath = path.join(os.homedir(), '.ai-research-assistant', 'master.key');
      const encryptedKey = await fs.readFile(keyPath);
      return encryptedKey;
    } catch (error) {
      return null;
    }
  }

  private async storeMasterKey(key: Buffer): Promise<void> {
    if (!safeStorage.isEncryptionAvailable()) {
      throw new Error('SafeStorage not available');
    }

    const fs = await import('fs/promises');
    const os = await import('os');
    const path = await import('path');

    const configDir = path.join(os.homedir(), '.ai-research-assistant');
    await fs.mkdir(configDir, { recursive: true });

    const encryptedKey = safeStorage.encryptString(key.toString('base64'));
    const keyPath = path.join(configDir, 'master.key');
    await fs.writeFile(keyPath, encryptedKey);
  }

  /**
   * 데이터 암호화
   */
  public encrypt(plaintext: string, key?: Buffer): EncryptedData {
    if (!this.masterKey && !key) {
      throw new Error('Encryption key not initialized');
    }

    const encryptionKey = key || this.masterKey!;
    const salt = crypto.randomBytes(this.config.saltLength);
    const iv = crypto.randomBytes(IV_LENGTH);

    // 키 파생
    const derivedKey = crypto.pbkdf2Sync(
      encryptionKey,
      salt,
      this.config.keyDerivationRounds,
      KEY_LENGTH,
      'sha256'
    );

    // 데이터 암호화
    const cipher = crypto.createCipheriv(this.config.algorithm, derivedKey, iv);
    let encrypted = cipher.update(plaintext, 'utf8', 'hex');
    encrypted += cipher.final('hex');

    return {
      iv: iv.toString('hex'),
      salt: salt.toString('hex'),
      data: encrypted,
      algorithm: this.config.algorithm,
      keyDerivationRounds: this.config.keyDerivationRounds,
    };
  }

  /**
   * 데이터 복호화
   */
  public decrypt(encryptedData: EncryptedData, key?: Buffer): string {
    if (!this.masterKey && !key) {
      throw new Error('Decryption key not initialized');
    }

    const decryptionKey = key || this.masterKey!;
    const salt = Buffer.from(encryptedData.salt, 'hex');
    const iv = Buffer.from(encryptedData.iv, 'hex');
    const encrypted = encryptedData.data;

    // 키 파생
    const derivedKey = crypto.pbkdf2Sync(
      decryptionKey,
      salt,
      encryptedData.keyDerivationRounds,
      KEY_LENGTH,
      'sha256'
    );

    // 데이터 복호화
    const decipher = crypto.createDecipheriv(encryptedData.algorithm, derivedKey, iv);
    let decrypted = decipher.update(encrypted, 'hex', 'utf8');
    decrypted += decipher.final('utf8');

    return decrypted;
  }

  /**
   * 파일 암호화
   */
  public async encryptFile(inputPath: string, outputPath?: string): Promise<string> {
    const fs = await import('fs/promises');
    const path = await import('path');

    const content = await fs.readFile(inputPath, 'utf8');
    const encrypted = this.encrypt(content);

    const finalOutputPath = outputPath || inputPath + '.encrypted';
    await fs.writeFile(finalOutputPath, JSON.stringify(encrypted));

    return finalOutputPath;
  }

  /**
   * 파일 복호화
   */
  public async decryptFile(inputPath: string, outputPath?: string): Promise<string> {
    const fs = await import('fs/promises');
    const path = await import('path');

    const encryptedContent = await fs.readFile(inputPath, 'utf8');
    const encryptedData: EncryptedData = JSON.parse(encryptedContent);
    const decrypted = this.decrypt(encryptedData);

    const finalOutputPath = outputPath || inputPath.replace('.encrypted', '');
    await fs.writeFile(finalOutputPath, decrypted);

    return finalOutputPath;
  }

  /**
   * 대량 데이터 스트리밍 암호화
   */
  public createEncryptionStream(key?: Buffer) {
    if (!this.masterKey && !key) {
      throw new Error('Encryption key not initialized');
    }

    const encryptionKey = key || this.masterKey!;
    const salt = crypto.randomBytes(this.config.saltLength);
    const iv = crypto.randomBytes(IV_LENGTH);

    const derivedKey = crypto.pbkdf2Sync(
      encryptionKey,
      salt,
      this.config.keyDerivationRounds,
      KEY_LENGTH,
      'sha256'
    );

    const cipher = crypto.createCipheriv(this.config.algorithm, derivedKey, iv);

    return {
      cipher,
      metadata: {
        iv: iv.toString('hex'),
        salt: salt.toString('hex'),
        algorithm: this.config.algorithm,
        keyDerivationRounds: this.config.keyDerivationRounds,
      }
    };
  }

  /**
   * 대량 데이터 스트리밍 복호화
   */
  public createDecryptionStream(metadata: Omit<EncryptedData, 'data'>, key?: Buffer) {
    if (!this.masterKey && !key) {
      throw new Error('Decryption key not initialized');
    }

    const decryptionKey = key || this.masterKey!;
    const salt = Buffer.from(metadata.salt, 'hex');
    const iv = Buffer.from(metadata.iv, 'hex');

    const derivedKey = crypto.pbkdf2Sync(
      decryptionKey,
      salt,
      metadata.keyDerivationRounds,
      KEY_LENGTH,
      'sha256'
    );

    return crypto.createDecipheriv(metadata.algorithm, derivedKey, iv);
  }

  /**
   * 키 회전 (보안 향상을 위한 정기적인 키 변경)
   */
  public async rotateKey(newPassword?: string): Promise<void> {
    if (!this.masterKey) {
      throw new Error('Master key not initialized');
    }

    const oldKey = this.masterKey;

    if (newPassword) {
      await this.initializePasswordBasedKey(newPassword);
    } else {
      // 새로운 랜덤 키 생성
      this.masterKey = crypto.randomBytes(KEY_LENGTH);
      await this.storeMasterKey(this.masterKey);
    }

    // 이벤트 발생 (기존 데이터 재암호화 필요)
    this.emit('keyRotated', { oldKey, newKey: this.masterKey });
  }

  private emit(event: string, data: any): void {
    // 이벤트 처리 로직 (필요시 EventEmitter 상속)
    console.log(`Encryption event: ${event}`, data);
  }

  /**
   * 메모리 정리 (보안상 중요)
   */
  public dispose(): void {
    if (this.masterKey) {
      this.masterKey.fill(0); // 메모리에서 키 데이터 삭제
      this.masterKey = null;
    }
  }
}

// 전역 암호화 관리자 인스턴스
export const globalEncryptionManager = new EncryptionManager();
```

3.2.2 보안 저장소 구현

암호화된 데이터베이스와 파일 시스템을 통합한 보안 저장소를 구현합니다:

```
// src/storage/secure-storage.ts
import { EncryptionManager, EncryptedData } from '../security/encryption';
import { Database } from 'sqlite3';
import * as path from 'path';
import * as fs from 'fs/promises';

export interface SecureStorageConfig {
  databasePath: string;
  encryptionManager: EncryptionManager;
  autoBackup?: boolean;
  backupInterval?: number; // 분 단위
}

export interface StorageItem {
  key: string;
  value: string;
  metadata: {
    createdAt: string;
    updatedAt: string;
    tags?: string[];
    category?: string;
    expiresAt?: string;
  };
}

export interface BackupConfig {
  location: string;
  maxBackups: number;
  compression: boolean;
}

export class SecureStorage {
  private db: Database;
  private encryption: EncryptionManager;
  private config: SecureStorageConfig;
  private backupTimer?: NodeJS.Timeout;

  constructor(config: SecureStorageConfig) {
    this.config = config;
    this.encryption = config.encryptionManager;
    this.initializeDatabase();

    if (config.autoBackup) {
      this.startAutoBackup();
    }
  }

  private async initializeDatabase(): Promise<void> {
    // 데이터베이스 디렉토리 생성
    const dbDir = path.dirname(this.config.databasePath);
    await fs.mkdir(dbDir, { recursive: true });
    this.db = new Database(this.config.databasePath);

    // 테이블 생성
    await this.createTables();

    // WAL 모드 활성화 (성능 향상)
    await this.execute("PRAGMA journal_mode=WAL");
    await this.execute("PRAGMA synchronous=NORMAL");
    await this.execute("PRAGMA cache_size=10000");
    await this.execute("PRAGMA temp_store=memory");
  }

  private async createTables(): Promise<void> {
    const queries = [
      `CREATE TABLE IF NOT EXISTS secure_storage (
        key TEXT PRIMARY KEY,
        encrypted_value TEXT NOT NULL,
        iv TEXT NOT NULL,
        salt TEXT NOT NULL,
        algorithm TEXT NOT NULL,
        key_derivation_rounds INTEGER NOT NULL,
        created_at TEXT NOT NULL,
        updated_at TEXT NOT NULL,
        tags TEXT,
        category TEXT,
        expires_at TEXT
      )`,

      `CREATE TABLE IF NOT EXISTS file_storage (
        id TEXT PRIMARY KEY,
        original_path TEXT NOT NULL,
        encrypted_path TEXT NOT NULL,
        file_size INTEGER NOT NULL,
        mime_type TEXT,
        checksum TEXT NOT NULL,
        created_at TEXT NOT NULL,
        updated_at TEXT NOT NULL,
        metadata TEXT
      )`,

      `CREATE TABLE IF NOT EXISTS backup_log (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        backup_path TEXT NOT NULL,
        backup_size INTEGER NOT NULL,
        created_at TEXT NOT NULL,
        restored_at TEXT,
        checksum TEXT NOT NULL
      )`,

      // 인덱스 생성
      `CREATE INDEX IF NOT EXISTS idx_storage_category ON secure_storage(category)`,
      `CREATE INDEX IF NOT EXISTS idx_storage_tags ON secure_storage(tags)`,
      `CREATE INDEX IF NOT EXISTS idx_storage_expires ON secure_storage(expires_at)`,
      `CREATE INDEX IF NOT EXISTS idx_file_path ON file_storage(original_path)`,
      `CREATE INDEX IF NOT EXISTS idx_backup_date ON backup_log(created_at)`,
    ];
    for (const query of queries) {
      await this.execute(query);
    }
  }

  private execute(query: string, params: any[] = []): Promise<any> {
    return new Promise((resolve, reject) => {
      this.db.run(query, params, function(err) {
        if (err) reject(err);
        else resolve(this);
      });
    });
  }

  private get(query: string, params: any[] = []): Promise<any> {
    return new Promise((resolve, reject) => {
      this.db.get(query, params, (err, row) => {
        if (err) reject(err);
        else resolve(row);
      });
    });
  }

  private all(query: string, params: any[] = []): Promise<any[]> {
    return new Promise((resolve, reject) => {
      this.db.all(query, params, (err, rows) => {
        if (err) reject(err);
        else resolve(rows || []);
      });
    });
  }

  /**
   * 보안 데이터 저장
   */
  public async setItem(key: string, value: string, metadata: Partial<StorageItem['metadat
    const encrypted = this.encryption.encrypt(value);
    const now = new Date().toISOString();

    const itemMetadata = {
      createdAt: now,
      updatedAt: now,
      ...metadata,
    };

    await this.execute(
      `INSERT OR REPLACE INTO secure_storage
       (key, encrypted_value, iv, salt, algorithm, key_derivation_rounds,
        created_at, updated_at, tags, category, expires_at)
       VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)`,
      [
        key,
        encrypted.data,
        encrypted.iv,
        encrypted.salt,
        encrypted.algorithm,
        encrypted.keyDerivationRounds,
        itemMetadata.createdAt,
        itemMetadata.updatedAt,
        itemMetadata.tags ? JSON.stringify(itemMetadata.tags) : null,
        itemMetadata.category || null,
        itemMetadata.expiresAt || null,
      ]
    );
  }

  /**
   * 보안 데이터 조회
   */
  public async getItem(key: string): Promise<StorageItem | null> {
    const row = await this.get(
      `SELECT * FROM secure_storage WHERE key = ? AND (expires_at IS NULL OR expires_at >
      [key, new Date().toISOString()]
    );

    if (!row) return null;

    const encryptedData: EncryptedData = {
      data: row.encrypted_value,
      iv: row.iv,
      salt: row.salt,
      algorithm: row.algorithm,
      keyDerivationRounds: row.key_derivation_rounds,
    };

    const decryptedValue = this.encryption.decrypt(encryptedData);

    return {
      key: row.key,
      value: decryptedValue,
      metadata: {
        createdAt: row.created_at,
        updatedAt: row.updated_at,
        tags: row.tags ? JSON.parse(row.tags) : undefined,
        category: row.category,
        expiresAt: row.expires_at,
      },
    };
  }

  /**
   * 데이터 삭제
   */
  public async removeItem(key: string): Promise<boolean> {
    const result = await this.execute(
      `DELETE FROM secure_storage WHERE key = ?`,
      [key]
    );
    return result.changes > 0;
  }
```

```
/**
 * 카테고리별 데이터 조회
 */
public async getItemsByCategory(category: string): Promise<StorageItem[]> {
  const rows = await this.all(
    `SELECT * FROM secure_storage WHERE category = ? AND (expires_at IS NULL OR expires
    [category, new Date().toISOString()]
  );

  const items: StorageItem[] = [];

  for (const row of rows) {
    const encryptedData: EncryptedData = {
      data: row.encrypted_value,
      iv: row.iv,
      salt: row.salt,
      algorithm: row.algorithm,
      keyDerivationRounds: row.key_derivation_rounds,
    };

    const decryptedValue = this.encryption.decrypt(encryptedData);

    items.push({
      key: row.key,
      value: decryptedValue,
      metadata: {
        createdAt: row.created_at,
        updatedAt: row.updated_at,
        tags: row.tags ? JSON.parse(row.tags) : undefined,
        category: row.category,
        expiresAt: row.expires_at,
      },
    });
  }

  return items;
}

/**
 * 태그 검색
 */
public async searchByTags(tags: string[]): Promise<StorageItem[]> {
  const tagConditions = tags.map(() => `tags LIKE ?`).join(' AND ');
  const tagParams = tags.map(tag => `%"${tag}"%`);

  const rows = await this.all(
    `SELECT * FROM secure_storage WHERE ${tagConditions} AND (expires_at IS NULL OR exp
    [...tagParams, new Date().toISOString()]
  );

  const items: StorageItem[] = [];

  for (const row of rows) {
    const encryptedData: EncryptedData = {
      data: row.encrypted_value,
      iv: row.iv,
      salt: row.salt,
      algorithm: row.algorithm,
      keyDerivationRounds: row.key_derivation_rounds,
    };

    const decryptedValue = this.encryption.decrypt(encryptedData);

    items.push({
      key: row.key,
      value: decryptedValue,
      metadata: {
        createdAt: row.created_at,
        updatedAt: row.updated_at,
        tags: row.tags ? JSON.parse(row.tags) : undefined,
        category: row.category,
        expiresAt: row.expires_at,
      },
    });
  }

  return items;
}

/**
 * 만료된 데이터 정리
 */
public async cleanupExpiredData(): Promise<number> {
  const result = await this.execute(
    `DELETE FROM secure_storage WHERE expires_at IS NOT NULL AND expires_at <= ?`,
    [new Date().toISOString()]
  );
  return result.changes;
}

/**
 * 파일 암호화 및 저장
 */
public async storeFile(filePath: string, metadata: Record<string, any> = {}): Promise<s
  const stats = await fs.stat(filePath);
  const fileContent = await fs.readFile(filePath);

  // 체크섬 계산
  const crypto = await import('crypto');
  const checksum = crypto.createHash('sha256').update(fileContent).digest('hex');

  // 파일 암호화
  const fileId = crypto.randomUUID();
  const encryptedPath = path.join(path.dirname(this.config.databasePath), 'files', `${f

  await fs.mkdir(path.dirname(encryptedPath), { recursive: true });
  await this.encryption.encryptFile(filePath, encryptedPath);

  // 메타데이터 저장
  const now = new Date().toISOString();
  await this.execute(
    `INSERT INTO file_storage
```

```
        (id, original_path, encrypted_path, file_size, mime_type, checksum, created_at, up
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)`,
      [
        fileId,
        filePath,
        encryptedPath,
        stats.size,
        this.getMimeType(filePath),
        checksum,
        now,
        now,
        JSON.stringify(metadata),
      ]
    );

    return fileId;
  }

  /**
   * 파일 복호화 및 복원
   */
  public async retrieveFile(fileId: string, outputPath?: string): Promise<string> {
    const row = await this.get(
      `SELECT * FROM file_storage WHERE id = ?`,
      [fileId]
    );

    if (!row) {
      throw new Error(`File not found: ${fileId}`);
    }

    const finalOutputPath = outputPath || row.original_path;
    await this.encryption.decryptFile(row.encrypted_path, finalOutputPath);

    // 체크섬 검증
    const restoredContent = await fs.readFile(finalOutputPath);
    const crypto = await import('crypto');
    const newChecksum = crypto.createHash('sha256').update(restoredContent).digest('hex')

    if (newChecksum !== row.checksum) {
      throw new Error('File integrity check failed');
    }

    return finalOutputPath;
  }

  /**
   * 백업 생성
   */
  public async createBackup(backupConfig: BackupConfig): Promise<string> {
    const timestamp = new Date().toISOString().replace(/[:.]/g, '-');
    const backupFileName = `secure-storage-backup-${timestamp}.db`;
    const backupPath = path.join(backupConfig.location, backupFileName);

    // 백업 디렉토리 생성
    await fs.mkdir(backupConfig.location, { recursive: true });
    // 데이터베이스 백업
    await fs.copyFile(this.config.databasePath, backupPath);

    // 압축 (선택사항)
    let finalBackupPath = backupPath;
    if (backupConfig.compression) {
      const zlib = await import('zlib');
      const gzip = zlib.createGzip();
      const input = await fs.readFile(backupPath);
      const compressed = await new Promise<Buffer>((resolve, reject) => {
        gzip.end(input, (err) => {
          if (err) reject(err);
        });

        const chunks: Buffer[] = [];
        gzip.on('data', chunk => chunks.push(chunk));
        gzip.on('end', () => resolve(Buffer.concat(chunks)));
        gzip.on('error', reject);
      });

      finalBackupPath = backupPath + '.gz';
      await fs.writeFile(finalBackupPath, compressed);
      await fs.unlink(backupPath); // 원본 삭제
    }

    // 백업 로그 기록
    const stats = await fs.stat(finalBackupPath);
    const crypto = await import('crypto');
    const content = await fs.readFile(finalBackupPath);
    const checksum = crypto.createHash('sha256').update(content).digest('hex');

    await this.execute(
      `INSERT INTO backup_log (backup_path, backup_size, created_at, checksum)
       VALUES (?, ?, ?, ?)`,
      [finalBackupPath, stats.size, new Date().toISOString(), checksum]
    );

    // 오래된 백업 정리
    await this.cleanupOldBackups(backupConfig);

    return finalBackupPath;
  }

  /**
   * 백업 복원
   */
  public async restoreBackup(backupPath: string): Promise<void> {
    // 백업 파일 존재 확인
    const stats = await fs.stat(backupPath);
    if (!stats.isFile()) {
      throw new Error('Backup file not found');
    }

    // 현재 데이터베이스 백업 (안전을 위해)
    const currentBackupPath = this.config.databasePath + '.before-restore';
    await fs.copyFile(this.config.databasePath, currentBackupPath);

    try {
      // 압축 해제 (필요시)
      let sourceFile = backupPath;
      if (backupPath.endsWith('.gz')) {
        const zlib = await import('zlib');
        const compressed = await fs.readFile(backupPath);
        const decompressed = await new Promise<Buffer>((resolve, reject) => {
          zlib.gunzip(compressed, (err, result) => {
            if (err) reject(err);
            else resolve(result);
          });
        });

        sourceFile = backupPath.replace('.gz', '');
        await fs.writeFile(sourceFile, decompressed);
      }

      // 데이터베이스 종료
      await this.close();

      // 백업으로 덮어쓰기
      await fs.copyFile(sourceFile, this.config.databasePath);

      // 데이터베이스 재연결
      await this.initializeDatabase();

      // 복원 로그 기록
      await this.execute(
        `UPDATE backup_log SET restored_at = ? WHERE backup_path = ?`,
        [new Date().toISOString(), backupPath]
      );

      // 임시 파일 정리
      if (sourceFile !== backupPath) {
        await fs.unlink(sourceFile);
      }

    } catch (error) {
      // 복원 실패시 원본 복구
      await fs.copyFile(currentBackupPath, this.config.databasePath);
      await this.initializeDatabase();
      throw error;
    } finally {
      // 임시 백업 파일 삭제
      try {
        await fs.unlink(currentBackupPath);
      } catch {
        // 무시
      }
    }
  }

  private async cleanupOldBackups(config: BackupConfig): Promise<void> {
    const backups = await this.all(
      `SELECT * FROM backup_log ORDER BY created_at DESC LIMIT -1 OFFSET ?`,
      [config.maxBackups]
    );

    for (const backup of backups) {
      try {
        await fs.unlink(backup.backup_path);
        await this.execute(
          `DELETE FROM backup_log WHERE id = ?`,
          [backup.id]
        );
      } catch (error) {
        console.warn(`Failed to delete old backup: ${backup.backup_path}`, error);
      }
    }
  }

  private startAutoBackup(): void {
    if (this.backupTimer) {
      clearInterval(this.backupTimer);
    }

    const interval = (this.config.backupInterval || 60) * 60 * 1000; // 분을 밀리초로 변환

    this.backupTimer = setInterval(async () => {
      try {
        const backupPath = path.join(path.dirname(this.config.databasePath), 'backups');
        await this.createBackup({
          location: backupPath,
          maxBackups: 10,
          compression: true,
        });
        console.log('Auto backup completed');
      } catch (error) {
        console.error('Auto backup failed:', error);
      }
    }, interval);
  }

  private getMimeType(filePath: string): string {
    const ext = path.extname(filePath).toLowerCase();
    const mimeTypes: Record<string, string> = {
      '.txt': 'text/plain',
      '.md': 'text/markdown',
      '.json': 'application/json',
      '.pdf': 'application/pdf',
      '.doc': 'application/msword',
      '.docx': 'application/vnd.openxmlformats-officedocument.wordprocessingml.document',
      '.xls': 'application/vnd.ms-excel',
      '.xlsx': 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet',
      '.jpg': 'image/jpeg',
      '.jpeg': 'image/jpeg',
      '.png': 'image/png',
      '.gif': 'image/gif',
      '.mp4': 'video/mp4',
      '.mp3': 'audio/mpeg',
    };

    return mimeTypes[ext] || 'application/octet-stream';
  }

  /**
   * 데이터베이스 연결 종료
   */
  public async close(): Promise<void> {
    if (this.backupTimer) {
      clearInterval(this.backupTimer);
    }

    return new Promise((resolve, reject) => {
      this.db.close((err) => {
        if (err) reject(err);
        else resolve();
      });
    });
  }

  /**
   * 메모리 및 리소스 정리
   */
  public dispose(): void {
    this.close().catch(console.error);
    this.encryption.dispose();
  }
}
```

4. SQLite FTS5 전문 검색 구현

4.1 FTS5 데이터베이스 설계

SQLite의 FTS5 확장을 활용한 고성능 전문 검색 시스템을 구현합니다[15][16]:

```
// src/database/fts-manager.ts
import { Database } from 'sqlite3';
import * as path from 'path';
import { EncryptionManager } from '../security/encryption';

export interface SearchableDocument {
  id: string;
  title: string;
  content: string;
  category: string;
  tags: string[];
  metadata: Record<string, any>;
  createdAt: string;
  updatedAt: string;
}

export interface SearchOptions {
  query: string;
  categories?: string[];
  tags?: string[];
  dateRange?: {
    start: string;
    end: string;
  };
  limit?: number;
  offset?: number;
  highlight?: boolean;
  rankingBoost?: Record<string, number>;
}

export interface SearchResult {
  document: SearchableDocument;
  rank: number;
  snippet: string;
  highlights: string[];
  matchCount: number;
}

export interface SearchResponse {
  results: SearchResult[];
  totalCount: number;
  queryTime: number;
  suggestions?: string[];
}

export class FTSManager {
  private db: Database;
  private encryption: EncryptionManager;
  private dbPath: string;

  constructor(dbPath: string, encryption: EncryptionManager) {
    this.dbPath = dbPath;
    this.encryption = encryption;
    this.initializeDatabase();
  }

  private async initializeDatabase(): Promise<void> {
    this.db = new Database(this.dbPath);

    // WAL 모드 및 성능 최적화 설정
    await this.execute("PRAGMA journal_mode=WAL");
    await this.execute("PRAGMA synchronous=NORMAL");
    await this.execute("PRAGMA cache_size=20000");
    await this.execute("PRAGMA temp_store=memory");
    await this.execute("PRAGMA mmap_size=134217728"); // 128MB

    // 테이블 생성
    await this.createTables();
    await this.createIndexes();
  }

  private async createTables(): Promise<void> {
    // 문서 메타데이터 테이블
    await this.execute(`
      CREATE TABLE IF NOT EXISTS documents (
        id TEXT PRIMARY KEY,
        title TEXT NOT NULL,
        category TEXT NOT NULL,
        tags TEXT, -- JSON 배열
        file_path TEXT,
        file_size INTEGER,
        file_hash TEXT,
        created_at TEXT NOT NULL,
        updated_at TEXT NOT NULL,
        metadata TEXT -- JSON 객체
      )
    `);

    // FTS5 가상 테이블 생성
    await this.execute(`
      CREATE VIRTUAL TABLE IF NOT EXISTS documents_fts USING fts5(
        id UNINDEXED,
        title,
        content,
        category UNINDEXED,
        tags,
        content='documents',
        content_rowid='rowid',
        tokenize='porter unicode61 remove_diacritics 1'
      )
    `);

    // FTS5 트리거 설정 (문서 변경시 자동 인덱스 업데이트)
    await this.execute(`
      CREATE TRIGGER IF NOT EXISTS documents_ai AFTER INSERT ON documents BEGIN
        INSERT INTO documents_fts(rowid, id, title, content, category, tags)
        VALUES (new.rowid, new.id, new.title, '', new.category, new.tags);
      END
    `);

    await this.execute(`
      CREATE TRIGGER IF NOT EXISTS documents_ad AFTER DELETE ON documents BEGIN
        INSERT INTO documents_fts(documents_fts, rowid, id, title, content, category, tag
        VALUES ('delete', old.rowid, old.id, old.title, '', old.category, old.tags);
      END
    `);

    await this.execute(`
      CREATE TRIGGER IF NOT EXISTS documents_au AFTER UPDATE ON documents BEGIN
        INSERT INTO documents_fts(documents_fts, rowid, id, title, content, category, tag
        VALUES ('delete', old.rowid, old.id, old.title, '', old.category, old.tags);
        INSERT INTO documents_fts(rowid, id, title, content, category, tags)
        VALUES (new.rowid, new.id, new.title, '', new.category, new.tags);
      END
    `);

    // 검색 기록 테이블
    await this.execute(`
      CREATE TABLE IF NOT EXISTS search_history (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        query TEXT NOT NULL,
        filters TEXT, -- JSON 객체
        result_count INTEGER NOT NULL,
        query_time REAL NOT NULL,
        created_at TEXT NOT NULL
      )
    `);

    // 자주 검색되는 용어 테이블
    await this.execute(`
      CREATE TABLE IF NOT EXISTS search_analytics (
        term TEXT PRIMARY KEY,
        search_count INTEGER NOT NULL DEFAULT 1,
        last_searched TEXT NOT NULL
      )
    `);
  }

  private async createIndexes(): Promise<void> {
    const indexes = [
      "CREATE INDEX IF NOT EXISTS idx_documents_category ON documents(category)",
      "CREATE INDEX IF NOT EXISTS idx_documents_created ON documents(created_at)",
      "CREATE INDEX IF NOT EXISTS idx_documents_updated ON documents(updated_at)",
      "CREATE INDEX IF NOT EXISTS idx_search_history_query ON search_history(query)",
      "CREATE INDEX IF NOT EXISTS idx_search_history_created ON search_history(created_at
      "CREATE INDEX IF NOT EXISTS idx_analytics_count ON search_analytics(search_count DE
    ];

    for (const index of indexes) {
      await this.execute(index);
    }
  }

  private execute(query: string, params: any[] = []): Promise<any> {
    return new Promise((resolve, reject) => {
      this.db.run(query, params, function(err) {
        if (err) reject(err);
        else resolve(this);
      });
    });
  }

  private get(query: string, params: any[] = []): Promise<any> {
    return new Promise((resolve, reject) => {
      this.db.get(query, params, (err, row) => {
        if (err) reject(err);
        else resolve(row);
      });
    });
  }

  private all(query: string, params: any[] = []): Promise<any[]> {
    return new Promise((resolve, reject) => {
      this.db.all(query, params, (err, rows) => {
        if (err) reject(err);
        else resolve(rows || []);
      });
    });
  }

  /**
   * 문서 추가
   */
  public async addDocument(document: Omit<SearchableDocument, 'createdAt' | 'updatedAt'>)
    const now = new Date().toISOString();
    const documentData = {
      ...document,
      createdAt: now,
      updatedAt: now,
    };

    // 메타데이터 테이블에 문서 정보 저장
    await this.execute(`
      INSERT OR REPLACE INTO documents
      (id, title, category, tags, file_path, created_at, updated_at, metadata)
      VALUES (?, ?, ?, ?, ?, ?, ?, ?)
    `, [
      documentData.id,
      documentData.title,
      documentData.category,
      JSON.stringify(documentData.tags),
      null, // file_path는 필요시 추가
      documentData.createdAt,
      documentData.updatedAt,
      JSON.stringify(documentData.metadata),
    ]);

    // FTS 테이블에 검색 가능한 콘텐츠 저장
    await this.execute(`
      INSERT OR REPLACE INTO documents_fts (id, title, content, category, tags)
      VALUES (?, ?, ?, ?, ?)
    `, [
      documentData.id,
      documentData.title,
      documentData.content,
      documentData.category,
      documentData.tags.join(' '),
    ]);
  }

  /**
   * 문서 업데이트
   */
  public async updateDocument(id: string, updates: Partial<SearchableDocument>): Promise<
    const now = new Date().toISOString();

    // 기존 문서 조회
    const existing = await this.get(`SELECT * FROM documents WHERE id = ?`, [id]);
    if (!existing) {
      throw new Error(`Document not found: ${id}`);
    }
```

```
    const updatedData = {
      ...existing,
      ...updates,
      tags: updates.tags ? JSON.stringify(updates.tags) : existing.tags,
      metadata: updates.metadata ? JSON.stringify(updates.metadata) : existing.metadata,
      updatedAt: now,
    };

    // 메타데이터 업데이트
    await this.execute(`
      UPDATE documents SET
        title = ?, category = ?, tags = ?, updated_at = ?, metadata = ?
      WHERE id = ?
    `, [
      updatedData.title,
      updatedData.category,
      updatedData.tags,
      updatedData.updatedAt,
      updatedData.metadata,
      id,
    ]);

    // FTS 인덱스 업데이트 (트리거에 의해 자동 처리됨)
    if (updates.title || updates.content || updates.tags) {
      await this.execute(`
        UPDATE documents_fts SET
          title = ?, content = ?, tags = ?
        WHERE id = ?
      `, [
        updates.title || existing.title,
        updates.content || '',
        updates.tags ? updates.tags.join(' ') : JSON.parse(existing.tags).join(' '),
        id,
      ]);
    }
  }

  /**
   * 문서 삭제
   */
  public async removeDocument(id: string): Promise<boolean> {
    const result = await this.execute(`DELETE FROM documents WHERE id = ?`, [id]);
    return result.changes > 0;
  }

  /**
   * 고급 검색 수행
   */
  public async search(options: SearchOptions): Promise<SearchResponse> {
    const startTime = Date.now();

    // 검색 쿼리 구성
    const { querySQL, params } = this.buildSearchQuery(options);

    // 검색 실행
    const results = await this.all(querySQL, params);
    // 총 개수 조회 (페이징을 위해)
    const countQuery = this.buildCountQuery(options);
    const totalCount = (await this.get(countQuery.querySQL, countQuery.params))?.total ||

    // 결과 변환
    const searchResults: SearchResult[] = await Promise.all(
      results.map(async (row) => await this.convertToSearchResult(row, options))
    );

    const queryTime = Date.now() - startTime;

    // 검색 기록 저장
    await this.recordSearch(options, totalCount, queryTime);

    // 검색 제안 생성
    const suggestions = await this.generateSuggestions(options.query);

    return {
      results: searchResults,
      totalCount,
      queryTime,
      suggestions,
    };
  }

  private buildSearchQuery(options: SearchOptions): { querySQL: string; params: any[] } {
    const {
      query,
      categories,
      tags,
      dateRange,
      limit = 50,
      offset = 0,
      rankingBoost = {},
    } = options;

    let whereConditions: string[] = [];
    let params: any[] = [];
    let joins: string[] = [];

    // FTS 검색 조건
    if (query.trim()) {
      // 쿼리 전처리 (불린 연산자 지원)
      const processedQuery = this.preprocessQuery(query);
      whereConditions.push("documents_fts MATCH ?");
      params.push(processedQuery);
    }

    // 카테고리 필터
    if (categories && categories.length > 0) {
      const placeholders = categories.map(() => '?').join(',');
      whereConditions.push(`d.category IN (${placeholders})`);
      params.push(...categories);
    }
    // 태그 필터
    if (tags && tags.length > 0) {
      const tagConditions = tags.map(() => `json_extract(d.tags, '$') LIKE ?`).join(' AND
      whereConditions.push(`(${tagConditions})`);
      params.push(...tags.map(tag => `%"${tag}"%`));
    }

    // 날짜 범위 필터
    if (dateRange) {
      if (dateRange.start) {
        whereConditions.push("d.created_at >= ?");
        params.push(dateRange.start);
      }
      if (dateRange.end) {
        whereConditions.push("d.created_at <= ?");
        params.push(dateRange.end);
      }
    }

    // 랭킹 부스트 계산
    let rankExpression = "bm25(documents_fts)";
    if (Object.keys(rankingBoost).length > 0) {
      const boostConditions = Object.entries(rankingBoost)
        .map(([field, boost]) => `CASE WHEN ${field} IS NOT NULL THEN ${boost} ELSE 1 END
        .join(' * ');
      rankExpression = `bm25(documents_fts) * (${boostConditions})`;
    }

    const whereClause = whereConditions.length > 0 ? `WHERE ${whereConditions.join(' AND

    const querySQL = `
      SELECT
        d.*,
        documents_fts.title as fts_title,
        documents_fts.content as fts_content,
        ${rankExpression} as rank,
        highlight(documents_fts, 1, '<mark>', '</mark>') as title_highlight,
        snippet(documents_fts, 2, '<mark>', '</mark>', '...', 32) as content_snippet
      FROM documents_fts
      JOIN documents d ON d.rowid = documents_fts.rowid
      ${joins.join(' ')}
      ${whereClause}
      ORDER BY rank
      LIMIT ? OFFSET ?
    `;

    params.push(limit, offset);

    return { querySQL, params };
  }

  private buildCountQuery(options: SearchOptions): { querySQL: string; params: any[] } {
    const { query, categories, tags, dateRange } = options;

    let whereConditions: string[] = [];
    let params: any[] = [];
    if (query.trim()) {
      const processedQuery = this.preprocessQuery(query);
      whereConditions.push("documents_fts MATCH ?");
      params.push(processedQuery);
    }

    if (categories && categories.length > 0) {
      const placeholders = categories.map(() => '?').join(',');
      whereConditions.push(`d.category IN (${placeholders})`);
      params.push(...categories);
    }

    if (tags && tags.length > 0) {
      const tagConditions = tags.map(() => `json_extract(d.tags, '$') LIKE ?`).join(' AND
      whereConditions.push(`(${tagConditions})`);
      params.push(...tags.map(tag => `%"${tag}"%`));
    }

    if (dateRange) {
      if (dateRange.start) {
        whereConditions.push("d.created_at >= ?");
        params.push(dateRange.start);
      }
      if (dateRange.end) {
        whereConditions.push("d.created_at <= ?");
        params.push(dateRange.end);
      }
    }

    const whereClause = whereConditions.length > 0 ? `WHERE ${whereConditions.join(' AND

    const querySQL = `
      SELECT COUNT(*) as total
      FROM documents_fts
      JOIN documents d ON d.rowid = documents_fts.rowid
      ${whereClause}
    `;

    return { querySQL, params };
  }

  private preprocessQuery(query: string): string {
    // 한국어 검색어 처리
    let processedQuery = query.trim();

    // AND, OR, NOT 연산자 지원
    processedQuery = processedQuery
      .replace(/\s+(AND|and)\s+/g, ' AND ')
      .replace(/\s+(OR|or)\s+/g, ' OR ')
      .replace(/\s+(NOT|not)\s+/g, ' NOT ');

    // 구문 검색 지원 (따옴표로 묶인 부분)
    processedQuery = processedQuery.replace(/"([^"]+)"/g, '"$1"');

    // 와일드카드 검색 지원
    if (!processedQuery.includes('"') && !processedQuery.includes('*')) {
      // 일반 검색어의 경우 각 단어에 대해 접두어 매칭 활성화
      const words = processedQuery.split(/\s+/).filter(word => word.length > 0);
      if (words.length === 1) {
        processedQuery = `${words[^0]}*`;
      } else {
        processedQuery = words.map(word => `${word}*`).join(' AND ');
      }
    }

    return processedQuery;
  }

  private async convertToSearchResult(row: any, options: SearchOptions): Promise<SearchRe
    // 메타데이터 파싱
    const tags = JSON.parse(row.tags || '[]');
    const metadata = JSON.parse(row.metadata || '{}');

    // 하이라이트 추출
    const highlights: string[] = [];
    if (options.highlight) {
      if (row.title_highlight && row.title_highlight !== row.title) {
        highlights.push(row.title_highlight);
      }
      if (row.content_snippet) {
        highlights.push(row.content_snippet);
      }
    }

    // 매치 카운트 계산 (간단한 구현)
    const matchCount = this.calculateMatchCount(options.query, row.fts_title + ' ' + row.

    return {
      document: {
        id: row.id,
        title: row.title,
        content: row.fts_content,
        category: row.category,
        tags,
        metadata,
        createdAt: row.created_at,
        updatedAt: row.updated_at,
      },
      rank: row.rank || 0,
      snippet: row.content_snippet || '',
      highlights,
      matchCount,
    };
  }

  private calculateMatchCount(query: string, content: string): number {
    const queryWords = query.toLowerCase().split(/\s+/).filter(word => word.length > 1);
    const contentLower = content.toLowerCase();

    let matchCount = 0;
    for (const word of queryWords) {
      const cleanWord = word.replace(/[*"]/g, '');
      const regex = new RegExp(`\\b${cleanWord}`, 'g');
      const matches = contentLower.match(regex);
      if (matches) {
        matchCount += matches.length;
      }
    }

    return matchCount;
  }

  private async recordSearch(options: SearchOptions, resultCount: number, queryTime: numb
    // 검색 기록 저장
    await this.execute(`
      INSERT INTO search_history (query, filters, result_count, query_time, created_at)
      VALUES (?, ?, ?, ?, ?)
    `, [
      options.query,
      JSON.stringify({
        categories: options.categories,
        tags: options.tags,
        dateRange: options.dateRange,
      }),
      resultCount,
      queryTime,
      new Date().toISOString(),
    ]);

    // 검색 분석 업데이트
    const words = options.query.toLowerCase().split(/\s+/).filter(word => word.length > 1
    for (const word of words) {
      await this.execute(`
        INSERT OR REPLACE INTO search_analytics (term, search_count, last_searched)
        VALUES (?, COALESCE((SELECT search_count FROM search_analytics WHERE term = ?) +
      `, [word, word, new Date().toISOString()]);
    }
  }

  private async generateSuggestions(query: string): Promise<string[]> {
    if (!query || query.trim().length < 2) {
      return [];
    }

    // 인기 검색어 기반 제안
    const popularTerms = await this.all(`
      SELECT term FROM search_analytics
      WHERE term LIKE ?
      ORDER BY search_count DESC
      LIMIT 5
    `, [`${query.toLowerCase()}%`]);

    // FTS 인덱스 기반 제안
    const ftsResults = await this.all(`
      SELECT DISTINCT title FROM documents_fts
      WHERE title MATCH ?
      LIMIT 5
    `, [`${query}*`]);

    const suggestions = [
      ...popularTerms.map((row: any) => row.term),
      ...ftsResults.map((row: any) => row.title),
    ];

    // 중복 제거 및 정렬
    return [...new Set(suggestions)].slice(0, 8);
  }

  /**
   * 문서 유사도 검색
   */
  public async findSimilarDocuments(documentId: string, limit: number = 10): Promise<Sear
    // 기준 문서 조회
    const baseDoc = await this.get(`
      SELECT * FROM documents_fts WHERE id = ?
    `, [documentId]);

    if (!baseDoc) {
      throw new Error(`Document not found: ${documentId}`);
    }

    // 기준 문서의 주요 키워드 추출
    const keywords = this.extractKeywords(baseDoc.title + ' ' + baseDoc.content);
    const query = keywords.slice(0, 10).join(' OR ');

    // 유사 문서 검색 (기준 문서 제외)
    const results = await this.search({
      query,
      limit,
      highlight: false,
    });

    return results.results.filter(result => result.document.id !== documentId);
  }

  private extractKeywords(text: string): string[] {
    // 간단한 키워드 추출 (실제로는 TF-IDF 또는 더 정교한 방법 사용)
    const words = text.toLowerCase()
      .replace(/[^\w\s]/g, ' ')
      .split(/\s+/)
      .filter(word => word.length > 2);

    const frequency: Record<string, number> = {};
    words.forEach(word => {
      frequency[word] = (frequency[word] || 0) + 1;
    });

    return Object.entries(frequency)
      .sort(([,a], [,b]) => b - a)
      .slice(0, 20)
      .map(([word]) => word);
  }
  /**
   * 검색 통계 조회
   */
  public async getSearchAnalytics(limit: number = 50): Promise<any[]> {
    return await this.all(`
      SELECT
        term,
        search_count,
        last_searched,
        datetime(last_searched) as last_searched_formatted
      FROM search_analytics
      ORDER BY search_count DESC
      LIMIT ?
    `, [limit]);
  }

  /**
   * 인덱스 최적화
   */
  public async optimizeIndex(): Promise<void> {
    await this.execute("INSERT INTO documents_fts(documents_fts) VALUES('optimize')");
    await this.execute("VACUUM");
    await this.execute("ANALYZE");
  }

  /**
   * 데이터베이스 정리
   */
  public async cleanup(): Promise<void> {
    // 오래된 검색 기록 삭제 (30일 이상)
    const thirtyDaysAgo = new Date(Date.now() - 30 * 24 * 60 * 60 * 1000).toISOString();
    await this.execute(`
      DELETE FROM search_history WHERE created_at < ?
    `, [thirtyDaysAgo]);

    // 사용되지 않는 검색 용어 정리 (검색 횟수가 1회이고 30일 이상 검색되지 않은 용어)
    await this.execute(`
      DELETE FROM search_analytics
      WHERE search_count = 1 AND last_searched < ?
    `, [thirtyDaysAgo]);

    // 인덱스 최적화
    await this.optimizeIndex();
  }

  /**
   * 데이터베이스 연결 종료
   */
  public async close(): Promise<void> {
    return new Promise((resolve, reject) => {
      this.db.close((err) => {
        if (err) reject(err);
        else resolve();
      });
    });
  }
}
```

4.2 실시간 검색 인터페이스 구현

React 기반의 실시간 검색 UI를 구현합니다:

```
// src/components/SearchInterface.tsx
import React, { useState, useEffect, useCallback, useMemo } from 'react';
import { debounce } from 'lodash';
import { SearchOptions, SearchResponse, SearchResult } from '../database/fts-manager';

interface SearchInterfaceProps {
  onResultSelect?: (result: SearchResult) => void;
  initialQuery?: string;
  enableFilters?: boolean;
  enableSuggestions?: boolean;
  maxResults?: number;
}

interface SearchFilters {
  categories: string[];
  tags: string[];
  dateRange: {
    start: string;
    end: string;
  } | null;
}

export const SearchInterface: React.FC<SearchInterfaceProps> = ({
  onResultSelect,
  initialQuery = '',
  enableFilters = true,
  enableSuggestions = true,
  maxResults = 50,
}) => {
  const [query, setQuery] = useState(initialQuery);
  const [results, setResults] = useState<SearchResponse | null>(null);
  const [loading, setLoading] = useState(false);
  const [suggestions, setSuggestions] = useState<string[]>([]);
  const [showSuggestions, setShowSuggestions] = useState(false);
  const [filters, setFilters] = useState<SearchFilters>({
    categories: [],
    tags: [],
    dateRange: null,
  });
  const [availableCategories, setAvailableCategories] = useState<string[]>([]);
  const [availableTags, setAvailableTags] = useState<string[]>([]);
  const [currentPage, setCurrentPage] = useState(0);
  const [selectedResultIndex, setSelectedResultIndex] = useState(-1);

  // 디바운스된 검색 함수
  const debouncedSearch = useCallback(
    debounce(async (searchQuery: string, searchFilters: SearchFilters, page: number = 0)
      if (!searchQuery.trim() && searchFilters.categories.length === 0 && searchFilters.t
        setResults(null);
        return;
      }

      setLoading(true);
      try {
        const searchOptions: SearchOptions = {
          query: searchQuery,
          categories: searchFilters.categories.length > 0 ? searchFilters.categories : ur
          tags: searchFilters.tags.length > 0 ? searchFilters.tags : undefined,
          dateRange: searchFilters.dateRange || undefined,
          limit: maxResults,
          offset: page * maxResults,
          highlight: true,
          rankingBoost: {
            title: 2.0, // 제목 매치에 더 높은 점수
            category: 1.5, // 카테고리 매치에 보너스
          },
        };

        const response = await window.electronAPI.research.performAdvancedSearch(searchOp
        setResults(response);
      } catch (error) {
        console.error('검색 오류:', error);
        setResults({ results: [], totalCount: 0, queryTime: 0 });
      } finally {
        setLoading(false);
      }
    }, 300),
    [maxResults]
  );

  // 자동완성 제안 가져오기
  const debouncedGetSuggestions = useCallback(
    debounce(async (searchQuery: string) => {
      if (!enableSuggestions || searchQuery.trim().length < 2) {
        setSuggestions([]);
        return;
      }

      try {
        const suggestionResults = await window.electronAPI.research.getSearchSuggestions(
        setSuggestions(suggestionResults);
      } catch (error) {
        console.error('제안 가져오기 오류:', error);
        setSuggestions([]);
      }
    }, 200),
    [enableSuggestions]
  );

  // 사용 가능한 카테고리 및 태그 로드
  useEffect(() => {
    const loadFilterOptions = async () => {
      try {
        const [categories, tags] = await Promise.all([
          window.electronAPI.research.getAvailableCategories(),
          window.electronAPI.research.getAvailableTags(),
        ]);
        setAvailableCategories(categories);
        setAvailableTags(tags);
      } catch (error) {
        console.error('필터 옵션 로드 오류:', error);
      }
    };

    loadFilterOptions();
  }, []);

  // 검색 실행
  useEffect(() => {
    debouncedSearch(query, filters, currentPage);
  }, [query, filters, currentPage, debouncedSearch]);

  // 자동완성 제안 업데이트
  useEffect(() => {
    if (query !== initialQuery) {
      debouncedGetSuggestions(query);
    }
  }, [query, initialQuery, debouncedGetSuggestions]);

  // 키보드 이벤트 처리
  const handleKeyDown = useCallback((e: React.KeyboardEvent) => {
    if (!results || results.results.length === 0) return;

    switch (e.key) {
      case 'ArrowDown':
        e.preventDefault();
        setSelectedResultIndex(prev =>
          prev < results.results.length - 1 ? prev + 1 : prev
        );
        break;
      case 'ArrowUp':
        e.preventDefault();
        setSelectedResultIndex(prev => prev > 0 ? prev - 1 : -1);
        break;
      case 'Enter':
        e.preventDefault();
        if (selectedResultIndex >= 0 && selectedResultIndex < results.results.length) {
          const selectedResult = results.results[selectedResultIndex];
          onResultSelect?.(selectedResult);
        } else if (showSuggestions && suggestions.length > 0) {
          setQuery(suggestions[^0]);
          setShowSuggestions(false);
        }
        break;
      case 'Escape':
        setShowSuggestions(false);
        setSelectedResultIndex(-1);
        break;
    }
  }, [results, selectedResultIndex, showSuggestions, suggestions, onResultSelect]);
```

```
// 필터 변경 핸들러
const handleFilterChange = useCallback((filterType: keyof SearchFilters, value: any) =>
  setFilters(prev => ({
    ...prev,
    [filterType]: value,
  }));
  setCurrentPage(0); // 필터 변경시 첫 페이지로 이동
}, []);

// 페이지 변경 핸들러
const handlePageChange = useCallback((newPage: number) => {
  setCurrentPage(newPage);
  setSelectedResultIndex(-1);
}, []);

// 검색 결과 하이라이트 렌더링
const renderHighlightedText = useCallback((text: string) => {
  return <span dangerouslySetInnerHTML={{ __html: text }} />;
}, []);

// 검색 통계 표시
const searchStats = useMemo(() => {
  if (!results) return null;

  const totalPages = Math.ceil(results.totalCount / maxResults);
  const currentStart = currentPage * maxResults + 1;
  const currentEnd = Math.min((currentPage + 1) * maxResults, results.totalCount);

  return {
    totalPages,
    currentStart,
    currentEnd,
    totalCount: results.totalCount,
    queryTime: results.queryTime,
  };
}, [results, currentPage, maxResults]);

return (
  <div className="search-interface">
    {/* 검색 입력 영역 */}
    <div className="search-input-container">
      <div className="search-input-wrapper">
        <input
          type="text"
          value={query}
          onChange={(e) => {
            setQuery(e.target.value);
            setShowSuggestions(true);
            setCurrentPage(0);
          }}
          onKeyDown={handleKeyDown}
          onFocus={() => setShowSuggestions(true)}
          onBlur={() => setTimeout(() => setShowSuggestions(false), 200)}
          placeholder="검색어를 입력하세요..."
          className="search-input"
        />
        {loading && <div className="search-loading">검색 중...</div>}
      </div>

      {/* 자동완성 제안 */}
      {enableSuggestions && showSuggestions && suggestions.length > 0 && (
        <div className="search-suggestions">
          {suggestions.map((suggestion, index) => (
            <div
              key={index}
              className="suggestion-item"
              onClick={() => {
                setQuery(suggestion);
                setShowSuggestions(false);
              }}
            >
              {suggestion}
            </div>
          ))}
        </div>
      )}
    </div>

    {/* 필터 영역 */}
    {enableFilters && (
      <div className="search-filters">
        {/* 카테고리 필터 */}
        <div className="filter-group">
          <label>카테고리:</label>
          <select
            multiple
            value={filters.categories}
            onChange={(e) => {
              const selected = Array.from(e.target.selectedOptions, option => option.va
              handleFilterChange('categories', selected);
            }}
            className="filter-select"
          >
            {availableCategories.map(category => (
              <option key={category} value={category}>
                {category}
              </option>
            ))}
          </select>
        </div>

        {/* 태그 필터 */}
        <div className="filter-group">
          <label>태그:</label>
          <div className="tag-filter">
            {availableTags.map(tag => (
              <label key={tag} className="tag-checkbox">
                <input
                  type="checkbox"
                  checked={filters.tags.includes(tag)}
                  onChange={(e) => {
                    const newTags = e.target.checked
                      ? [...filters.tags, tag]
                      : filters.tags.filter(t => t !== tag);
                    handleFilterChange('tags', newTags);
                  }}
                />
                {tag}
              </label>
            ))}
          </div>
        </div>

        {/* 날짜 범위 필터 */}
        <div className="filter-group">
          <label>날짜 범위:</label>
          <input
            type="date"
            value={filters.dateRange?.start || ''}
            onChange={(e) => {
              const newDateRange = {
                start: e.target.value,
                end: filters.dateRange?.end || '',
              };
              handleFilterChange('dateRange', newDateRange);
            }}
            className="date-input"
          />
          <span>~</span>
          <input
            type="date"
            value={filters.dateRange?.end || ''}
            onChange={(e) => {
              const newDateRange = {
                start: filters.dateRange?.start || '',
                end: e.target.value,
              };
              handleFilterChange('dateRange', newDateRange);
            }}
            className="date-input"
          />
        </div>
      </div>
    )}

    {/* 검색 통계 */}
    {searchStats && (
      <div className="search-stats">
        총 {searchStats.totalCount.toLocaleString()}개 결과 중 {searchStats.currentStart
        ({searchStats.queryTime}ms)
      </div>
    )}

    {/* 검색 결과 */}
    <div className="search-results">
      {results?.results.map((result, index) => (
        <div
          key={result.document.id}
          className={`search-result-item ${index === selectedResultIndex ? 'selected' :
          onClick={() => onResultSelect?.(result)}
        >
          <div className="result-header">
            <h3 className="result-title">
              {renderHighlightedText(result.highlights[^0] || result.document.title)}
            </h3>
            <div className="result-meta">
              <span className="result-category">{result.document.category}</span>
              <span className="result-date">
                {new Date(result.document.createdAt).toLocaleDateString('ko-KR')}
              </span>
              <span className="result-rank">점수: {result.rank.toFixed(2)}</span>
            </div>
          </div>

          <div className="result-content">
            <p className="result-snippet">
              {renderHighlightedText(result.snippet)}
            </p>
          </div>

          <div className="result-footer">
            <div className="result-tags">
              {result.document.tags.slice(0, 5).map(tag => (
                <span key={tag} className="result-tag">
                  {tag}
                </span>
              ))}
            </div>
            <div className="result-matches">
              {result.matchCount}개 일치
            </div>
          </div>
        </div>
      ))}
    </div>

    {/* 페이지네이션 */}
    {searchStats && searchStats.totalPages > 1 && (
      <div className="pagination">
        <button
          disabled={currentPage === 0}
          onClick={() => handlePageChange(currentPage - 1)}
          className="pagination-button"
        >
          이전
        </button>

        <div className="pagination-info">
          {currentPage + 1} / {searchStats.totalPages}
        </div>

        <button
          disabled={currentPage >= searchStats.totalPages - 1}
```

```
            onClick={() => handlePageChange(currentPage + 1)}
            className="pagination-button"
          >
            다음
          </button>
        </div>
      )}

      {/* 검색 결과 없음 */}
      {results && results.results.length === 0 && !loading && (
        <div className="no-results">
          <h3>검색 결과가 없습니다</h3>
          <p>다른 검색어를 시도해보세요.</p>
          {results.suggestions && results.suggestions.length > 0 && (
            <div className="search-suggestions-inline">
              <p>추천 검색어:</p>
              {results.suggestions.map(suggestion => (
                <button
                  key={suggestion}
                  onClick={() => setQuery(suggestion)}
                  className="suggestion-button"
                >
                  {suggestion}
                </button>
              ))}
            </div>
          )}
        </div>
      )}
    </div>
  );
};

export default SearchInterface;
```

5. 클라우드 스토리지 연동

5.1 OneDrive 연동 구현

Microsoft Graph API를 활용한 OneDrive 연동을 구현합니다[17][18]:

```
// src/cloud/onedrive-integration.ts
import axios, { AxiosInstance } from 'axios';
import { EncryptionManager } from '../security/encryption';

export interface OneDriveConfig {
  clientId: string;
  clientSecret: string;
  redirectUri: string;
  scopes: string[];
}

export interface OneDriveAuth {
  accessToken: string;
  refreshToken: string;
  expiresAt: number;
}

export interface OneDriveFile {
  id: string;
  name: string;
  size: number;
  createdDateTime: string;
  lastModifiedDateTime: string;
  webUrl: string;
  downloadUrl?: string;
  isFolder: boolean;
  parentPath: string;
  metadata?: Record<string, any>;
}

export interface OneDriveUploadOptions {
  file: Buffer | string;
  fileName: string;
  parentPath?: string;
  metadata?: Record<string, any>;
  encrypt?: boolean;
  conflictBehavior?: 'fail' | 'replace' | 'rename';
}

export interface OneDriveSync {
  localPath: string;
  remotePath: string;
  lastSync: string;
  syncDirection: 'upload' | 'download' | 'bidirectional';
  autoSync: boolean;
}

export class OneDriveIntegration {
  private config: OneDriveConfig;
  private auth: OneDriveAuth | null = null;
  private httpClient: AxiosInstance;
  private encryption: EncryptionManager;
  private syncConfigs: Map<string, OneDriveSync> = new Map();

  constructor(config: OneDriveConfig, encryption: EncryptionManager) {
    this.config = config;
    this.encryption = encryption;

    this.httpClient = axios.create({
      baseURL: 'https://graph.microsoft.com/v1.0',
      timeout: 30000,
    });

    this.setupInterceptors();
  }

  private setupInterceptors(): void {
    // 요청 인터셉터: 인증 토큰 자동 추가
    this.httpClient.interceptors.request.use(
      async (config) => {
        await this.ensureValidToken();
        if (this.auth?.accessToken) {
          config.headers['Authorization'] = `Bearer ${this.auth.accessToken}`;
        }
        return config;
      },
      (error) => Promise.reject(error)
    );

    // 응답 인터셉터: 토큰 만료시 자동 갱신
    this.httpClient.interceptors.response.use(
      (response) => response,
      async (error) => {
        if (error.response?.status === 401 && this.auth?.refreshToken) {
          try {
            await this.refreshAccessToken();
            // 원래 요청 재시도
            const originalRequest = error.config;
            originalRequest.headers['Authorization'] = `Bearer ${this.auth.accessToken}`;
            return this.httpClient(originalRequest);
          } catch (refreshError) {
            console.error('토큰 갱신 실패:', refreshError);
            await this.clearAuth();
            throw refreshError;
          }
        }
        return Promise.reject(error);
      }
    );
  }

  /**
   * OAuth 인증 URL 생성
   */
  public generateAuthUrl(): string {
    const params = new URLSearchParams({
      client_id: this.config.clientId,
      response_type: 'code',
      redirect_uri: this.config.redirectUri,
      scope: this.config.scopes.join(' '),
      state: this.generateRandomState(),
    });

    return `https://login.microsoftonline.com/common/oauth2/v2.0/authorize?${params.toStr
  }

  /**
   * 인증 코드로 토큰 교환
   */
  public async exchangeCodeForTokens(code: string, state: string): Promise<void> {
    const tokenData = {
      client_id: this.config.clientId,
      client_secret: this.config.clientSecret,
      code,
      redirect_uri: this.config.redirectUri,
      grant_type: 'authorization_code',
    };

    try {
      const response = await axios.post(
        'https://login.microsoftonline.com/common/oauth2/v2.0/token',
        new URLSearchParams(tokenData).toString(),
        {
          headers: {
            'Content-Type': 'application/x-www-form-urlencoded',
          },
        }
      );

      const { access_token, refresh_token, expires_in } = response.data;

      this.auth = {
        accessToken: access_token,
        refreshToken: refresh_token,
        expiresAt: Date.now() + (expires_in * 1000),
      };

      // 토큰을 암호화하여 저장
      await this.saveAuth();
    } catch (error) {
      console.error('토큰 교환 실패:', error);
      throw new Error('OneDrive 인증 실패');
    }
  }

  /**
   * 저장된 인증 정보 로드
   */
  public async loadAuth(): Promise<boolean> {
    try {
      const authData = await window.electronAPI.storage.get('onedrive_auth');
      if (authData) {
        this.auth = JSON.parse(authData);
        return true;
      }
    } catch (error) {
      console.error('인증 정보 로드 실패:', error);
    }
    return false;
  }

  /**
   * 인증 정보 저장
   */
  private async saveAuth(): Promise<void> {
    if (this.auth) {
      const encryptedAuth = this.encryption.encrypt(JSON.stringify(this.auth));
      await window.electronAPI.storage.set('onedrive_auth', JSON.stringify(encryptedAuth)
    }
  }
  /**
   * 인증 정보 삭제
   */
  private async clearAuth(): Promise<void> {
    this.auth = null;
    await window.electronAPI.storage.delete('onedrive_auth');
  }

  /**
   * 액세스 토큰 갱신
   */
  private async refreshAccessToken(): Promise<void> {
    if (!this.auth?.refreshToken) {
      throw new Error('리프레시 토큰이 없습니다');
    }

    const tokenData = {
      client_id: this.config.clientId,
      client_secret: this.config.clientSecret,
      refresh_token: this.auth.refreshToken,
      grant_type: 'refresh_token',
    };

    try {
      const response = await axios.post(
        'https://login.microsoftonline.com/common/oauth2/v2.0/token',
        new URLSearchParams(tokenData).toString(),
        {
          headers: {
            'Content-Type': 'application/x-www-form-urlencoded',
          },
        }
      );

      const { access_token, refresh_token, expires_in } = response.data;

      this.auth = {
        accessToken: access_token,
        refreshToken: refresh_token || this.auth.refreshToken,
        expiresAt: Date.now() + (expires_in * 1000),
      };

      await this.saveAuth();
    } catch (error) {
      console.error('토큰 갱신 실패:', error);
      throw error;
    }
  }

  /**
   * 토큰 유효성 확인 및 갱신
   */
  private async ensureValidToken(): Promise<void> {
    if (!this.auth) {
      const loaded = await this.loadAuth();
      if (!loaded) {
        throw new Error('OneDrive 인증이 필요합니다');
      }
    }

    // 토큰 만료 5분 전에 갱신
    if (this.auth && this.auth.expiresAt - Date.now() < 5 * 60 * 1000) {
      await this.refreshAccessToken();
    }
  }

  /**
   * 파일 목록 조회
   */
  public async listFiles(path: string = ''): Promise<OneDriveFile[]> {
    try {
      const encodedPath = encodeURIComponent(path);
      const endpoint = path
        ? `/me/drive/root:/${encodedPath}:/children`
        : '/me/drive/root/children';

      const response = await this.httpClient.get(endpoint);
      const items = response.data.value || [];

      return items.map((item: any) => ({
        id: item.id,
        name: item.name,
        size: item.size || 0,
        createdDateTime: item.createdDateTime,
        lastModifiedDateTime: item.lastModifiedDateTime,
        webUrl: item.webUrl,
        downloadUrl: item['@microsoft.graph.downloadUrl'],
        isFolder: !!item.folder,
        parentPath: path,
        metadata: {
          mimeType: item.file?.mimeType,
          sha1Hash: item.file?.hashes?.sha1Hash,
        },
      }));
    } catch (error) {
      console.error('파일 목록 조회 실패:', error);
      throw error;
    }
  }

  /**
   * 파일 업로드
   */
  public async uploadFile(options: OneDriveUploadOptions): Promise<OneDriveFile> {
    const { file, fileName, parentPath = '', metadata = {}, encrypt = false, conflictBeha

    try {
      let fileContent: Buffer;
      let finalFileName = fileName;

      // 파일 내용 처리
      if (typeof file === 'string') {
        fileContent = Buffer.from(file, 'utf8');
      } else {
        fileContent = file;
      }

      // 암호화 처리
      if (encrypt) {
        const encrypted = this.encryption.encrypt(fileContent.toString('base64'));
        fileContent = Buffer.from(JSON.stringify(encrypted));
        finalFileName = fileName + '.encrypted';
      }

      // 대용량 파일 (4MB 이상)은 세션을 통한 업로드 사용
      if (fileContent.length > 4 * 1024 * 1024) {
        return await this.uploadLargeFile(fileContent, finalFileName, parentPath, conflic
      }

      // 일반 파일 업로드
      const encodedPath = parentPath
        ? encodeURIComponent(`${parentPath}/${finalFileName}`)
        : encodeURIComponent(finalFileName);

      const endpoint = `/me/drive/root:/${encodedPath}:/content`;
      const params = conflictBehavior !== 'replace'
        ? { '@microsoft.graph.conflictBehavior': conflictBehavior }
        : {};

      const response = await this.httpClient.put(endpoint, fileContent, {
        headers: {
          'Content-Type': 'application/octet-stream',
        },
        params,
      });

      const uploadedFile = response.data;

      // 메타데이터 업데이트
      if (Object.keys(metadata).length > 0) {
        await this.updateFileMetadata(uploadedFile.id, metadata);
      }

      return {
        id: uploadedFile.id,
        name: uploadedFile.name,
        size: uploadedFile.size,
        createdDateTime: uploadedFile.createdDateTime,
        lastModifiedDateTime: uploadedFile.lastModifiedDateTime,
        webUrl: uploadedFile.webUrl,
        isFolder: false,
        parentPath,
        metadata: {
          ...metadata,
          encrypted: encrypt,
        },
      };
    } catch (error) {
```

```
    console.error('파일 업로드 실패:', error);
    throw error;
  }
}

/**
 * 대용량 파일 업로드 (분할 업로드)
 */
private async uploadLargeFile(
  content: Buffer,
  fileName: string,
  parentPath: string,
  conflictBehavior: string
): Promise<OneDriveFile> {
  const chunkSize = 10 * 1024 * 1024; // 10MB 청크
  const encodedPath = parentPath
    ? encodeURIComponent(`${parentPath}/${fileName}`)
    : encodeURIComponent(fileName);

  // 업로드 세션 생성
  const sessionResponse = await this.httpClient.post(
    `/me/drive/root:/${encodedPath}:/createUploadSession`,
    {
      item: {
        '@microsoft.graph.conflictBehavior': conflictBehavior,
      },
    }
  );

  const uploadUrl = sessionResponse.data.uploadUrl;
  let uploadedBytes = 0;

  // 청크별 업로드
  while (uploadedBytes < content.length) {
    const start = uploadedBytes;
    const end = Math.min(uploadedBytes + chunkSize, content.length);
    const chunk = content.subarray(start, end);

    const response = await axios.put(uploadUrl, chunk, {
      headers: {
        'Content-Range': `bytes ${start}-${end - 1}/${content.length}`,
        'Content-Length': chunk.length.toString(),
      },
    });

    if (response.status === 201 || response.status === 200) {
      // 업로드 완료
      const uploadedFile = response.data;
      return {
        id: uploadedFile.id,
        name: uploadedFile.name,
        size: uploadedFile.size,
        createdDateTime: uploadedFile.createdDateTime,
        lastModifiedDateTime: uploadedFile.lastModifiedDateTime,
        webUrl: uploadedFile.webUrl,
        isFolder: false,
        parentPath,
      };
    }

    uploadedBytes = end;
  }

  throw new Error('대용량 파일 업로드 실패');
}

/**
 * 파일 다운로드
 */
public async downloadFile(fileId: string, decrypt?: boolean): Promise<Buffer> {
  try {
    const response = await this.httpClient.get(`/me/drive/items/${fileId}/content`, {
      responseType: 'arraybuffer',
    });

    let content = Buffer.from(response.data);

    // 복호화 처리
    if (decrypt) {
      try {
        const encryptedData = JSON.parse(content.toString());
        const decrypted = this.encryption.decrypt(encryptedData);
        content = Buffer.from(decrypted, 'base64');
      } catch (error) {
        console.warn('복호화 실패, 원본 파일 반환:', error);
      }
    }

    return content;
  } catch (error) {
    console.error('파일 다운로드 실패:', error);
    throw error;
  }
}

/**
 * 파일 삭제
 */
public async deleteFile(fileId: string): Promise<void> {
  try {
    await this.httpClient.delete(`/me/drive/items/${fileId}`);
  } catch (error) {
    console.error('파일 삭제 실패:', error);
    throw error;
  }
}

/**
 * 폴더 생성
 */
public async createFolder(name: string, parentPath: string = ''): Promise<OneDriveFile>
  try {
      const endpoint = parentPath
        ? `/me/drive/root:/${encodeURIComponent(parentPath)}:/children`
        : '/me/drive/root/children';

      const response = await this.httpClient.post(endpoint, {
        name,
        folder: {},
        '@microsoft.graph.conflictBehavior': 'rename',
      });

      const folder = response.data;
      return {
        id: folder.id,
        name: folder.name,
        size: 0,
        createdDateTime: folder.createdDateTime,
        lastModifiedDateTime: folder.lastModifiedDateTime,
        webUrl: folder.webUrl,
        isFolder: true,
        parentPath,
      };
    } catch (error) {
      console.error('폴더 생성 실패:', error);
      throw error;
    }
  }

  /**
   * 파일 메타데이터 업데이트
   */
  private async updateFileMetadata(fileId: string, metadata: Record<string, any>): Promis
    try {
      await this.httpClient.patch(`/me/drive/items/${fileId}`, {
        description: JSON.stringify(metadata),
      });
    } catch (error) {
      console.warn('메타데이터 업데이트 실패:', error);
    }
  }

  /**
   * 파일

<div style="text-align: center">⁂</div>

[^1]: https://github.com/KennyVaneetvelde/atomic-research-mcp
[^2]: https://github.com/mixelpixx/Google-Research-MCP
[^3]: https://electronjs.org/docs/latest/api/safe-storage
[^4]: https://mojoauth.com/encryption-decryption/aes-256-encryption--nodejs/
[^5]: https://mitcommlab.mit.edu/broad/commkit/file-structure/
[^6]: https://www.iteratorshq.com/blog/a-comprehensive-guide-on-project-folder-organizati
[^7]: https://theonetechnologies.com/blog/post/features-to-integrate-in-a-project-collabo
[^8]: https://dealhub.io/glossary/collaboration-features/
[^9]: https://modelcontextprotocol.io/quickstart/server
[^10]: https://www.weavely.ai/blog/claude-mcp
[^11]: https://illysamsa.tistory.com/entry/Claude-Desktop%EA%B3%BC-MCP-%EC%84%9C%EB%B2%84
```

```
[^12]: https://electronjs.org/docs/latest/tutorial/context-isolation
[^13]: https://nenara.tistory.com/263
[^14]: https://gist.github.com/vlucas/2bd40f62d20c1d49237a109d491974eb?permalink_comment_
[^15]: https://www.sqlite.org/fts5.html
[^16]: https://www.sqlitetutorial.net/sqlite-full-text-search/
[^17]: https://www.webdevstory.com/onedrive-integration-react/
[^18]: https://www.filestack.com/docs/tutorials/onedrive-for-business/
[^19]: https://en.wikipedia.org/wiki/Model_Context_Protocol
[^20]: https://github.blog/changelog/2025-04-04-github-mcp-server-public-preview/
[^21]: https://github.com/mnhlt/WebSearch-MCP
[^22]: https://www.anthropic.com/news/model-context-protocol
[^23]: https://github.com/orgs/anthropics/repositories
[^24]: https://platform.openai.com/docs/mcp
[^25]: https://openai.github.io/openai-agents-python/mcp/
[^26]: https://github.com/madhukarkumar/anthropic-mcp-servers
[^27]: https://www.claudemcp.com/servers/web-search
[^28]: https://www.redhat.com/en/blog/model-context-protocol-mcp-understanding-security-
[^29]: https://github.com/modelcontextprotocol/servers
[^30]: https://brightdata.com/blog/ai/web-scraping-with-mcp
[^31]: https://github.com/modelcontextprotocol
[^32]: https://github.com/anthropics
[^33]: https://mcp.so
[^34]: https://auth0.com/blog/mcp-specs-update-all-about-auth/
[^35]: https://www.anthropic.com/engineering/desktop-extensions
[^36]: https://lobechat.com/discover/mcp/taka499-mcp-web-search?hl=ko-KR
[^37]: https://github.com/modelcontextprotocol/modelcontextprotocol
[^38]: https://docs.anthropic.com/en/docs/claude-code/mcp
[^39]: https://github.com/kwp-lab/mcp-brave-search
[^40]: https://github.com/mixelpixx/Google-Search-MCP-Server
[^41]: https://lobehub.com/mcp/mark3labs-mcp-filesystem-server
[^42]: https://apidog.com/blog/brave-search-api-mcp-server/
[^43]: https://github.com/limklister/mcp-google-custom-search-server
[^44]: https://www.kdnuggets.com/10-awesome-mcp-servers
[^45]: https://brave.com/search/api/guides/use-with-claude-desktop-with-mcp/
[^46]: https://apidog.com/blog/google-search-console-mcp-server/
[^47]: https://github.com/mark3labs/mcp-filesystem-server
[^48]: https://blog.choonzang.com/it/ai/2626/
[^49]: https://mcpmarket.com/server/google-search-3
[^50]: https://mcpmarket.com/server/filesystem
[^51]: https://www.claudemcp.com/servers/brave-search
[^52]: https://mcpservers.org/servers/ahonn/mcp-server-gsc
[^53]: https://modelcontextprotocol.io/quickstart/user
[^54]: https://www.youtube.com/watch?v=l3vwwkmZN9M
[^55]: https://apidog.com/kr/blog/google-search-console-mcp-server-kr/
[^56]: https://www.reddit.com/r/ClaudeAI/comments/1h4yvep/mcp_filesystem_is_magic/
[^57]: https://modelcontextprotocol.io/examples
[^58]: https://www.flowhunt.io/mcp-servers/google-custom-search/
[^59]: https://openai.com/index/introducing-canvas/
[^60]: https://www.youtube.com/watch?v=KmBEN7b5Tbg
[^61]: https://www.timeundertension.ai/imagine/gpt-4o-canvas-a-new-way-to-collaborate-wit
[^62]: https://www.youtube.com/watch?v=tkkWLDcoLrQ
[^63]: https://www.youtube.com/watch?v=_n2P5UHGRBg
[^64]: https://www.deeplearning.ai/short-courses/collaborative-writing-and-coding-with-op
[^65]: https://n8n.io/integrations/canvas/and/claude/
[^66]: https://gemini.google/overview/canvas/
[^67]: https://ai.plainenglish.io/how-i-use-chatgpts-canvas-for-writing-and-editing-d70e7
```

```
[^68]: https://www.reddit.com/r/ClaudeAI/comments/1fvydtc/chatgpt_canvas_vs_claude_artifa
[^69]: https://www.geeky-gadgets.com/google-gemini-canvas-ai-coding-platform/
[^70]: https://www.youtube.com/watch?v=G1zFkbaodaI
[^71]: https://support.anthropic.com/en/articles/11649427-use-artifacts-to-visualize-and-
[^72]: https://www.theverge.com/google/631726/google-gemini-canvas-audio-overview-noteboo
[^73]: https://www.canva.com/ai-assistant/
[^74]: https://algocademy.com/blog/chatgpt-canvas-vs-claude-artifacts-for-programming-a-c
[^75]: https://support.google.com/gemini/answer/16047321?hl=ko
[^76]: https://github.com/langchain-ai/open-canvas
[^77]: https://altar.io/next-gen-of-human-ai-collaboration/
[^78]: https://blog.google/intl/ko-kr/products/gemini-collaboration-features/
[^79]: https://www.tiny.cloud/blog/how-ai-text-editors-improve-content-creation/
[^80]: https://canvascallback.vercel.app/guide
[^81]: https://hackernoon.com/lets-build-a-real-time-collaborative-document-editor-using-
[^82]: https://tiptap.dev/product/content-ai
[^83]: https://dev.to/sachinchaurasiya/how-to-build-a-collaborative-editor-with-nextjs-an
[^84]: https://faun.pub/windsurf-new-ai-text-editor-the-new-sota-dfa731b1a88f
[^85]: https://learn.thedesignsystem.guide/p/ai-ux-patterns-for-design-systems-661
[^86]: https://github.com/collaborativejs/collaborative-js
[^87]: https://lex.page
[^88]: https://uxdesign.cc/where-should-ai-sit-in-your-ui-1710a258390e
[^89]: https://javascript.plainenglish.io/how-i-built-a-real-time-collaborative-text-edit
[^90]: https://hyperwriteai.com
[^91]: https://departmentofproduct.substack.com/p/deep-the-ux-of-ai-assistants
[^92]: https://www.reddit.com/r/nextjs/comments/1gos5jy/suggestions_for_libraries_to_buil
[^93]: http://aieditor.dev
[^94]: https://uxplanet.org/7-key-design-patterns-for-ai-interfaces-893ab96988f6
[^95]: https://github.com/yjs/yjs
[^96]: https://paperpal.com
[^97]: https://www.koruux.com/ai-patterns-for-ui-design/
[^98]: https://elixirforum.com/t/reason-for-switching-from-monaco-to-codemirror/66999
[^99]: https://jkrsp.com/slate-js-vs-draft-js/
[^100]: https://saltaformaggio.ece.gatech.edu/publications/yang2025coindef.pdf
[^101]: https://stackoverflow.com/questions/49923334/how-to-make-monaco-editor-auto-fit-c
[^102]: https://npm-compare.com/draft-js,remirror,slate
[^103]: https://electronjs.org/docs/latest/tutorial/security
[^104]: https://npm-compare.com/react-ace,react-codemirror,react-monaco-editor
```