Electron 기반 웹 크롤러 봇 감지 회피 기술 문서

문서 정보

- **작성일**: 2025-12-05
- **대상 플랫폼**: Electron (Chromium 기반)
- **목적**: 웹 크롤링 시 봇 감지 시스템 우회
- **적용 사례**: 무신사(Musinsa) 등 상용 전자상거래 플랫폼

---

1. 개요

1.1 배경

현대 웹사이트들은 자동화된 봇 트래픽을 감지하고 차단하기 위해 다양한 기술을 사용합니다. Electron 기반 크롤러는 일반 브라우저와 달리 여러 자동화 흔적을 남기기 때문에 쉽게 감지될 수 있습니다.

1.2 봇 감지 주요 메커니즘

- User Agent 및 HTTP 헤더 분석
- JavaScript 환경 속성 검사 (navigator.webdriver 등)
- Canvas/WebGL Fingerprinting
- 마우스/키보드 이벤트 패턴 분석
- 네트워크 요청 빈도 및 타이밍 분석
- 세션 및 쿠키 검증

---

2. 기술적 구현 방법

2.1 User Agent 및 HTTP 헤더 정상화

목적: 실제 브라우저의 요청과 동일한 헤더를 전송

구현 코드:

---

10. 완전한 구현 예시

10.1 통합 크롤러 클래스

목적: 모든 기능을 통합한 프로덕션 레벨 크롤러

구현 코드:
```javascript
const { app, BrowserWindow, session } = require('electron');
const path = require('path');

class CompleteCrawler {
  constructor(config = {}) {
    this.config = {
      maxRequestsPerMinute: config.maxRequestsPerMinute || 8,
      minDelay: config.minDelay || 3000,
      maxDelay: config.maxDelay || 7000,
      maxRetries: config.maxRetries || 3,
      userAgent: config.userAgent || 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36',
      headless: config.headless !== false,
      ...config
    };

    this.rateLimiter = new CrawlerRateLimiter(this.config);
    this.errorHandler = new CrawlerErrorHandler(this.config.maxRetries);
    this.monitor = new CrawlerMonitor();
    this.logger = new CrawlerLogger('./logs');
    this.riskStrategy = new RiskMitigationStrategy();
    this.memoryManager = null;
    this.cache = new CrawlerCache({ maxAge: 3600000, maxSize: 500 });
    this.isRunning = false;
  }

  async initialize() {
    await app.whenReady();

    this.win = new BrowserWindow({
      width: 1920,
      height: 1080,
      show: !this.config.headless,
      webPreferences: {
        preload: path.join(__dirname, 'preload.js'),
        nodeIntegration: false,
```

```
      contextIsolation: true,
      webSecurity: true
    }
  });

  this.memoryManager = new MemoryManager(this.win);
  this.dashboard = new MonitoringDashboard(this.monitor);

  // 설정 적용
  await this.setupHeaders();
  await this.setupResourceBlocking();

  await this.logger.logSessionStart(this.config);
  console.log('✓ Crawler initialized successfully');
}

setupHeaders() {
  const ses = this.win.webContents.session;
  ses.setUserAgent(this.config.userAgent);

  ses.webRequest.onBeforeSendHeaders((details, callback) => {
    details.requestHeaders['Accept-Language'] = 'ko-KR,ko;q=0.9,en-US;q=0.8';
    details.requestHeaders['Accept-Encoding'] = 'gzip, deflate, br';
    details.requestHeaders['Sec-Fetch-Dest'] = 'document';
    details.requestHeaders['Sec-Fetch-Mode'] = 'navigate';
    details.requestHeaders['Sec-Fetch-Site'] = 'same-origin';
    details.requestHeaders['Sec-Ch-Ua'] = '"Not_A Brand";v="8", "Chromium";v="120"';
    details.requestHeaders['Sec-Ch-Ua-Mobile'] = '?0';
    details.requestHeaders['Sec-Ch-Ua-Platform'] = '"Windows"';

    callback({ requestHeaders: details.requestHeaders });
  });
}

setupResourceBlocking() {
  const ses = this.win.webContents.session;
  const blockList = [
    'google-analytics.com',
    'googletagmanager.com',
    'facebook.net',
    'doubleclick.net',
    'hotjar.com',
    'crazyegg.com'
  ];

  ses.webRequest.onBeforeRequest((details, callback) => {
    const shouldBlock = blockList.some(domain =>
```

```
      details.url.includes(domain)
    );
    callback({ cancel: shouldBlock });
  });
}

async crawl(url, useCache = true) {
  // 캐시 확인
  if (useCache && this.cache.has(url)) {
    await this.logger.info(`Cache hit: ${url}`);
    return this.cache.get(url);
  }

  return await this.errorHandler.executeWithRetry(async () => {
    await this.rateLimiter.throttle();

    const startTime = Date.now();

    try {
      await this.logger.info(`Crawling: ${url}`);

      // 페이지 로드
      await this.win.loadURL(url);
      await this.waitForPageLoad();
      await randomDelay(2000, 4000);

      // 데이터 추출
      const data = await this.extractData();

      // 캐시 저장
      if (useCache) {
        this.cache.set(url, data);
      }

      // 모니터링
      const duration = Date.now() - startTime;
      this.monitor.recordRequest(true, duration);
      await this.logger.logPageCrawl(url, true, duration);

      // 메모리 관리
      await this.memoryManager.checkAndClear();

      // 위험 평가
      const risk = await this.riskStrategy.evaluateRisk(this.monitor);
      await this.handleRisk(risk);

      return data;
```

```
    } catch (error) {
      const duration = Date.now() - startTime;
      this.monitor.recordRequest(false, duration, error);
      await this.logger.logPageCrawl(url, false, duration, error);
      throw error;
    }
  });
}

async handleRisk(risk) {
  if (risk.action === 'STOP') {
    await this.logger.error('Critical risk detected', risk);
    await this.riskStrategy.handleStop(this.logger);
  } else if (risk.action === 'SLOW_DOWN') {
    await this.logger.warn('Risk warning', risk);
    await this.riskStrategy.handleSlowDown(this.rateLimiter, this.logger);
  } else if (risk.action === 'MANUAL_INTERVENTION') {
    await this.riskStrategy.handleManualIntervention(this.logger);
  }
}

async waitForPageLoad() {
  await this.win.webContents.executeJavaScript(`
    new Promise((resolve) => {
      if (document.readyState === 'complete') {
        resolve();
      } else {
        window.addEventListener('load', resolve);
      }
    });
  `);
}

async extractData() {
  // 기본 데이터 추출 - 실제로는 사이트별로 커스터마이징 필요
  return await this.win.webContents.executeJavaScript(`
    ({
      title: document.title,
      url: window.location.href,
      timestamp: Date.now(),
      html: document.documentElement.outerHTML
    });
  `);
}

async crawlMultiple(urls, options = {}) {
```

```
  const {
    parallel = false,
    maxConcurrent = 3,
    useCache = true
  } = options;

  this.isRunning = true;
  const results = [];

  if (parallel) {
    const parallelCrawler = new ParallelCrawler({
      maxConcurrent,
      rateLimiter: this.rateLimiter
    });

    const promises = urls.map(url =>
      parallelCrawler.addTask(() => this.crawl(url, useCache))
    );

    const settled = await Promise.allSettled(promises);

    settled.forEach((result, index) => {
      if (result.status === 'fulfilled') {
        results.push({
          success: true,
          url: urls[index],
          data: result.value
        });
      } else {
        results.push({
          success: false,
          url: urls[index],
          error: result.reason.message
        });
      }
    });

  } else {
    // 순차 처리
    for (const url of urls) {
      if (!this.isRunning) {
        await this.logger.warn('Crawler stopped by user');
        break;
      }

      try {
        const data = await this.crawl(url, useCache);
```

```
        results.push({ success: true, url, data });
      } catch (error) {
        await this.logger.error(`Failed to crawl ${url}`, error);
        results.push({ success: false, url, error: error.message });
      }
    }
  }

  return results;
}

pause() {
  this.isRunning = false;
  this.logger.info('Crawler paused');
}

resume() {
  this.isRunning = true;
  this.logger.info('Crawler resumed');
}

getStats() {
  return {
    monitor: this.monitor.getReport(),
    cache: this.cache.getStats(),
    memory: this.memoryManager.getMemoryUsage()
  };
}

printStats() {
  this.dashboard.printDashboard();
}

startAutoMonitoring(intervalMs = 30000) {
  this.dashboard.startAutoDisplay(intervalMs);
}

stopAutoMonitoring() {
  this.dashboard.stopAutoDisplay();
}

async cleanup() {
  await this.logger.logSessionEnd(this.getStats());

  if (this.win) {
    this.win.close();
  }
```

```
    this.stopAutoMonitoring();

    console.log('✓ Crawler cleanup complete');
  }
}

// 사용 예시
async function main() {
  const crawler = new CompleteCrawler({
    maxRequestsPerMinute: 6,
    minDelay: 5000,
    maxDelay: 10000,
    headless: true
  });

  try {
    await crawler.initialize();

    // 자동 모니터링 시작
    crawler.startAutoMonitoring(30000);

    const urls = [
      'https://www.musinsa.com/categories/item/001',
      'https://www.musinsa.com/categories/item/002',
      'https://www.musinsa.com/categories/item/003'
    ];

    console.log(`Starting to crawl ${urls.length} URLs...`);

    const results = await crawler.crawlMultiple(urls, {
      parallel: false,
      useCache: true
    });

    // 결과 출력
    console.log('\n=== Crawling Results ===');
    const successCount = results.filter(r => r.success).length;
    console.log(`Success: ${successCount}/${results.length}`);

    // 통계 출력
    crawler.printStats();

    // 결과 저장
    const fs = require('fs').promises;
    await fs.writeFile(
      './results.json',
```

```
      JSON.stringify(results, null, 2),
      'utf8'
    );
    console.log('✓ Results saved to results.json');

  } catch (error) {
    console.error('Crawler error:', error);
  } finally {
    await crawler.cleanup();
    app.quit();
  }
}

// Electron 앱 시작
app.on('ready', main);

// 종료 처리
process.on('SIGINT', async () => {
  console.log('\nReceived SIGINT, shutting down...');
  app.quit();
});
```

10.2 무신사 특화 크롤러

목적: 무신사 사이트에 최적화된 크롤러 구현

구현 코드:
```javascript
class MusinsaSpecificCrawler extends CompleteCrawler {
  constructor(config = {}) {
    super({
      ...config,
      maxRequestsPerMinute: config.maxRequestsPerMinute || 6,
      minDelay: 5000,
      maxDelay: 10000
    });

    this.baseUrl = 'https://www.musinsa.com';
  }

  async initialize() {
    await super.initialize();

    // 무신사 특화 헤더 설정
    this.setupMusinsaHeaders();
```

```
  // 세션 워밍업
  await this.warmup();
}

setupMusinsaHeaders() {
  const ses = this.win.webContents.session;

  ses.webRequest.onBeforeSendHeaders((details, callback) => {
    const url = new URL(details.url);

    if (url.hostname.includes('musinsa.com')) {
      details.requestHeaders['Referer'] = this.baseUrl;
      details.requestHeaders['Origin'] = this.baseUrl;

      if (url.pathname.includes('/api/') || url.pathname.includes('/graphql')) {
        details.requestHeaders['X-Requested-With'] = 'XMLHttpRequest';
      }
    }

    callback({ requestHeaders: details.requestHeaders });
  });
}

async warmup() {
  await this.logger.info('Starting Musinsa session warmup...');

  // 메인 페이지
  await this.win.loadURL(this.baseUrl);
  await randomDelay(3000, 5000);

  // 카테고리 페이지 방문
  await this.win.loadURL(`${this.baseUrl}/categories/item/001`);
  await randomDelay(2000, 4000);

  // 자연스러운 스크롤
  await naturalScroll(this.win, 1000);
  await randomDelay(1000, 2000);

  await this.logger.info('Warmup complete');
}

async crawlProductList(categoryUrl, maxPages = 5) {
  const allProducts = [];

  for (let page = 1; page <= maxPages; page++) {
    const pageUrl = `${categoryUrl}?page=${page}`;
```

```
    try {
      await this.logger.info(`Crawling category page ${page}/${maxPages}`);

      await this.rateLimiter.throttle();
      await this.win.loadURL(pageUrl);
      await this.waitForPageLoad();
      await randomDelay(2000, 4000);

      // 스크롤하여 이미지 로드
      await naturalScroll(this.win, 2000);
      await randomDelay(1000, 2000);

      // 제품 데이터 추출
      const products = await this.win.webContents.executeJavaScript(`
        Array.from(document.querySelectorAll('.li_box')).map(item => {
          const link = item.querySelector('a');
          const brand = item.querySelector('.item_title');
          const name = item.querySelector('.item_name');
          const price = item.querySelector('.price');
          const img = item.querySelector('img');

          return {
            url: link ? link.href : null,
            brand: brand ? brand.textContent.trim() : null,
            name: name ? name.textContent.trim() : null,
            price: price ? price.textContent.trim() : null,
            imageUrl: img ? img.src : null
          };
        }).filter(item => item.url);
      `);

      allProducts.push(...products);
      await this.logger.info(`Found ${products.length} products on page ${page}`);

    } catch (error) {
      await this.logger.error(`Failed to crawl page ${page}`, error);
    }
  }

  return allProducts;
}

async crawlProductDetail(productUrl) {
  try {
    await this.rateLimiter.throttle();
    await this.win.loadURL(productUrl);
    await this.waitForPageLoad();
```

```
    await randomDelay(3000, 5000);

    // 상세 페이지 스크롤
    await naturalScroll(this.win, 1500);
    await randomDelay(1000, 2000);

    // 상세 정보 추출
    const detail = await this.win.webContents.executeJavaScript(`
      (() => {
        const data = {};

        // 기본 정보
        data.brand = document.querySelector('.product_article .product_title')?.textContent.trim();
        data.name = document.querySelector('.product_article .product_subtitle')?.textContent.trim();
        data.price = document.querySelector('.product_article .product_price')?.textContent.trim();

        // 옵션 정보
        data.sizes = Array.from(document.querySelectorAll('.option_select option'))
          .map(opt => opt.textContent.trim())
          .filter(text => text && text !== '선택');

        // 상세 설명
        data.description = document.querySelector('.product_information')?.textContent.trim();

        // 이미지
        data.images = Array.from(document.querySelectorAll('.product_img img'))
          .map(img => img.src)
          .filter(src => src);

        // 리뷰 수
        data.reviewCount = document.querySelector('.review_count')?.textContent.trim();

        // 평점
        data.rating = document.querySelector('.rating')?.textContent.trim();

        return data;
      })();
    `);

    return detail;

  } catch (error) {
    await this.logger.error(`Failed to crawl product detail: ${productUrl}`, error);
    throw error;
  }
}
```

```
  async crawlCategory(categoryUrl, options = {}) {
    const {
      maxPages = 5,
      includeDetails = false,
      maxDetails = 10
    } = options;

    // 제품 리스트 수집
    const products = await this.crawlProductList(categoryUrl, maxPages);

    // 상세 정보 수집 (옵션)
    if (includeDetails && products.length > 0) {
      const detailProducts = products.slice(0, maxDetails);

      for (let i = 0; i < detailProducts.length; i++) {
        try {
          const detail = await this.crawlProductDetail(detailProducts[i].url);
          products[i] = { ...products[i], ...detail };

          await this.logger.info(`Collected detail ${i + 1}/${detailProducts.length}`);

        } catch (error) {
          await this.logger.error(`Failed to collect detail for product ${i + 1}`, error);
        }
      }
    }

    return products;
  }
}

// 사용 예시
async function crawlMusinsa() {
  const crawler = new MusinsaSpecificCrawler({
    maxRequestsPerMinute: 6,
    headless: true
  });

  try {
    await crawler.initialize();
    crawler.startAutoMonitoring(60000);

    // 카테고리 크롤링
    const products = await crawler.crawlCategory(
      'https://www.musinsa.com/categories/item/001',
      {
        maxPages: 3,
```

```
        includeDetails: true,
        maxDetails: 5
      }
    );

    console.log(`\nCollected ${products.length} products`);

    // 결과 저장
    const fs = require('fs').promises;
    await fs.writeFile(
      './musinsa_products.json',
      JSON.stringify(products, null, 2),
      'utf8'
    );

    crawler.printStats();

  } catch (error) {
    console.error('Crawling error:', error);
  } finally {
    await crawler.cleanup();
  }
}
`````

---

11. 결론 및 권장사항

11.1 핵심 요약

필수 구현 사항 (반드시 포함):
1. User Agent 및 HTTP 헤더 정상화
2. WebDriver 속성 완전 제거
3. Rate Limiting (분당 6-10회)
4. 랜덤 지연 (2-7초)
5. 에러 핸들링 및 재시도 로직

권장 구현 사항 (품질 향상):
1. 자연스러운 행동 패턴 (스크롤, 클릭)
2. 세션 관리 및 워밍업
3. 모니터링 및 로깅 시스템
4. 메모리 관리
5. 캐싱 전략

선택 구현 사항 (고급 기능):
1. 프록시 로테이션

2. 병렬 처리
3. Canvas Fingerprinting 대응
4. 실시간 대시보드

11.2 성공을 위한 핵심 원칙

1. 보수적 접근 (Conservative Approach)
- 처음에는 매우 느린 속도로 시작 (분당 3-5회)
- 성공률을 모니터링하며 점진적으로 속도 증가
- 이상 징후 발견 시 즉시 속도 감소

2. 실시간 모니터링 (Real-time Monitoring)
- 성공률, 응답 시간, 에러 패턴 추적
- 임계값 설정 및 자동 대응
- 상세 로그 기록으로 문제 추적

3. 적응성 (Adaptability)
- 차단 징후 발견 시 즉시 전략 변경
- 동적 Rate Limiting 조정
- 에러 패턴에 따른 대응

4. 지속성 (Persistence)
- 장기적인 세션 유지로 신뢰도 구축
- 정상적인 사용자 패턴 모방
- 급격한 변화 피하기

5. 투명성 (Transparency)
- 법적, 윤리적 가이드라인 준수
- robots.txt 및 이용약관 확인
- 필요 시 사전 허가 요청

11.3 단계별 도입 계획

1단계: 기본 구현 (1-2일)
- User Agent, WebDriver 제거
- Rate Limiting 기본 구현
- 단순 에러 핸들링
- 목표: 10개 페이지 안정적 크롤링

2단계: 안정화 (3-5일)
- 자연스러운 행동 패턴 추가
- 세션 관리 및 워밍업
- 상세 로깅 시스템
- 목표: 100개 페이지 90% 성공률

3단계: 최적화 (1주)
- 캐싱 전략 구현

- 메모리 관리 최적화
- 모니터링 대시보드
- 목표: 1000개 페이지 안정적 처리

4단계: 고급 기능 (선택)
- 병렬 처리 구현
- 프록시 로테이션
- Canvas Fingerprinting 대응
- 목표: 대규모 크롤링 가능

11.4 문제 해결 가이드

문제: 403 Forbidden 에러 발생
- 원인: IP 차단 또는 봇 감지
- 해결: 30분 대기 → IP 변경 → 속도 감소

문제: 429 Rate Limit 에러
- 원인: 요청 빈도 초과
- 해결: Retry-After 준수 → 속도 50% 감소

문제: CAPTCHA 발동
- 원인: 강력한 봇 감지
- 해결: 수동 해결 → 세션 재설정 → 워밍업 재수행

문제: 낮은 성공률 (<70%)
- 원인: 차단 또는 불안정한 네트워크
- 해결: 속도 감소 → 에러 패턴 분석 → 전략 조정

문제: 메모리 누수
- 원인: 캐시 미정리 또는 리소스 미해제
- 해결: 주기적 cleanup → 캐시 크기 제한 → GC 강제 실행

11.5 최종 체크리스트

배포 전 반드시 확인:
- [ ] 모든 자동화 흔적 제거 완료
- [ ] Rate Limiting 적절히 설정
- [ ] 에러 핸들링 완전 구현
- [ ] 메모리 누수 테스트 완료
- [ ] 로깅 시스템 정상 작동
- [ ] 법적 검토 완료 (robots.txt, 이용약관)
- [ ] 모니터링 대시보드 준비
- [ ] 긴급 중단 메커니즘 구현
- [ ] 백업 및 복구 계획 수립
- [ ] 팀원 교육 완료

11.6 추가 학습 자료

공식 문서:
- Electron 공식 문서: https://www.electronjs.org/docs
- Chromium DevTools Protocol 문서
- HTTP/2 및 헤더 표준 문서

관련 기술:
- Canvas Fingerprinting 원리
- TLS Fingerprinting
- Behavioral Analysis 기법
- Rate Limiting 알고리즘

커뮤니티:
- Electron GitHub Issues
- Stack Overflow (electron, web-scraping 태그)
- Reddit r/webscraping

부록 A: preload.js 전체 코드
```javascript
// preload.js - 봇 감지 우회를 위한 사전 로드 스크립트

window.addEventListener('DOMContentLoaded', () => {
  console.log('[Anti-Detection] Applying measures...');

  // ========================================
  // 1. WebDriver 속성 제거
  // ========================================
  Object.defineProperty(navigator, 'webdriver', {
    get: () => undefined,
    configurable: true
  });

  delete navigator.__proto__.webdriver;

  // ========================================
  // 2. Chrome 객체 추가
  // ========================================
  if (!window.chrome) {
    window.chrome = {
      runtime: {},
      loadTimes: function() {
        return {
          commitLoadTime: Date.now() / 1000 - Math.random() * 10,
          connectionInfo: 'http/1.1',
          finishDocumentLoadTime: Date.now() / 1000 - Math.random() * 5,
```

```
        finishLoadTime: Date.now() / 1000 - Math.random() * 3,
        firstPaintAfterLoadTime: Date.now() / 1000 - Math.random() * 2,
        firstPaintTime: Date.now() / 1000 - Math.random() * 8,
        navigationType: 'Other',
        npnNegotiatedProtocol: 'h2',
        requestTime: Date.now() / 1000 - Math.random() * 15,
        startLoadTime: Date.now() / 1000 - Math.random() * 12,
        wasAlternateProtocolAvailable: false,
        wasFetchedViaSpdy: true,
        wasNpnNegotiated: true
      };
    },
    csi: function() {
      return {
        onloadT: Date.now(),
        pageT: Date.now() - Math.random() * 1000,
        startE: Date.now() - Math.random() * 2000,
        tran: 15
      };
    },
    app: {
      InstallState: { DISABLED: 'disabled', INSTALLED: 'installed', NOT_INSTALLED: 'not_installed' },
      RunningState: { CANNOT_RUN: 'cannot_run', READY_TO_RUN: 'ready_to_run', RUNNING: 'running' },
      getDetails: function() { return { id: '' }; },
      getIsInstalled: function() { return false; },
      installState: function() { return 'not_installed'; },
      runningState: function() { return 'cannot_run'; }
    }
  };
}

// =====================================
// 3. Plugins 정상화
// =====================================
Object.defineProperty(navigator, 'plugins', {
  get: () => {
    return [
      {
        0: { type: 'application/pdf', suffixes: 'pdf', description: 'Portable Document Format' },
        name: 'Chrome PDF Plugin',
        filename: 'internal-pdf-viewer',
        description: 'Portable Document Format',
        length: 1
      },
      {
        0: { type: 'application/x-google-chrome-pdf', suffixes: 'pdf', description: '' },
        name: 'Chrome PDF Viewer',
```

```
        filename: 'mhjfbmdgcfjbbpaeojofohoefgiehjai',
        description: '',
        length: 1
      },
      {
        0: { type: 'application/x-nacl', suffixes: '', description: 'Native Client Executable' },
        1: { type: 'application/x-pnacl', suffixes: '', description: 'Portable Native Client Executable' },
        name: 'Native Client',
        filename: 'internal-nacl-plugin',
        description: '',
        length: 2
      }
    ];
  },
  configurable: true
});

// =====================================
// 4. Languages 정상화
// =====================================
Object.defineProperty(navigator, 'languages', {
  get: () => ['ko-KR', 'ko', 'en-US', 'en'],
  configurable: true
});

// =====================================
// 5. Permission API 정상화
// =====================================
const originalQuery = window.navigator.permissions.query;
window.navigator.permissions.query =javascript
const { session } = require('electron');

// 세션 레벨에서 헤더 수정
session.defaultSession.webRequest.onBeforeSendHeaders((details, callback) => {
  details.requestHeaders['User-Agent'] = 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36';
  details.requestHeaders['Accept'] =
'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8';
  details.requestHeaders['Accept-Language'] = 'ko-KR,ko;q=0.9,en-US;q=0.8';
  details.requestHeaders['Accept-Encoding'] = 'gzip, deflate, br';
  details.requestHeaders['Sec-Fetch-Dest'] = 'document';
  details.requestHeaders['Sec-Fetch-Mode'] = 'navigate';
  details.requestHeaders['Sec-Fetch-Site'] = 'same-origin';
  details.requestHeaders['Sec-Ch-Ua'] = '"Not_A Brand";v="8", "Chromium";v="120"';
  details.requestHeaders['Sec-Ch-Ua-Mobile'] = '?0';
  details.requestHeaders['Sec-Ch-Ua-Platform'] = '"Windows"';
```

```
  callback({ requestHeaders: details.requestHeaders });
});
```

핵심 포인트:
- Sec-Fetch-* 헤더는 최신 브라우저의 보안 정책을 반영
- User Agent는 최신 Chrome 버전과 일치시킴
- Accept-Language는 타겟 사이트의 주 언어로 설정

2.2 WebDriver 속성 제거

목적: 자동화 도구 사용 흔적 제거

구현 코드:
```javascript
// preload.js
Object.defineProperty(navigator, 'webdriver', {
  get: () => undefined,
  configurable: true
});

// 프로토타입 체인에서도 제거
delete navigator.__proto__.webdriver;

// Chrome 객체 추가 (일반 브라우저 환경)
window.chrome = {
  runtime: {},
  loadTimes: function() {},
  csi: function() {}
};

// Electron 특화 객체 제거
delete window.process;
delete window.require;
delete window.module;
delete window.exports;
```

핵심 포인트:
- `navigator.webdriver`는 가장 기본적인 봇 감지 메커니즘
- Chrome 객체를 추가하여 Chromium 기반 브라우저로 위장
- Electron 특유의 Node.js 통합 흔적 제거

2.3 JavaScript 환경 정상화

목적: 일반 브라우저 환경과 동일한 JavaScript API 제공

구현 코드:
```javascript
// preload.js 또는 executeJavaScript로 주입
window.addEventListener('DOMContentLoaded', () => {
  // Plugins 정상화
  Object.defineProperty(navigator, 'plugins', {
    get: () => {
      return [
        {
          name: 'Chrome PDF Plugin',
          filename: 'internal-pdf-viewer',
          description: 'Portable Document Format'
        },
        {
          name: 'Chrome PDF Viewer',
          filename: 'mhjfbmdgcfjbbpaeojofohoefgiehjai',
          description: ''
        },
        {
          name: 'Native Client',
          filename: 'internal-nacl-plugin',
          description: ''
        }
      ];
    }
  });

  // Languages 정상화
  Object.defineProperty(navigator, 'languages', {
    get: () => ['ko-KR', 'ko', 'en-US', 'en']
  });

  // Permission API 정상화
  const originalQuery = window.navigator.permissions.query;
  window.navigator.permissions.query = (parameters) => {
    return parameters.name === 'notifications'
      ? Promise.resolve({ state: Notification.permission })
      : originalQuery(parameters);
  };

  // Viewport 크기 정상화
  Object.defineProperty(window, 'outerWidth', {
    get: () => window.innerWidth
```

```
  });

  Object.defineProperty(window, 'outerHeight', {
    get: () => window.innerHeight
  });
});
```

핵심 포인트:
- Plugins 배열은 실제 Chrome과 동일하게 구성
- Permission API는 봇 감지의 새로운 벡터로 사용됨
- Viewport 크기 불일치는 헤드리스 브라우저의 특징

2.4 Canvas Fingerprinting 대응

목적: Canvas 기반 디바이스 지문 추적 방해

구현 코드:
```javascript
// Canvas에 미세한 노이즈 추가
const originalGetImageData = CanvasRenderingContext2D.prototype.getImageData;
CanvasRenderingContext2D.prototype.getImageData = function(...args) {
  const imageData = originalGetImageData.apply(this, args);

  // 픽셀 데이터에 미세한 변화 추가 (감지 불가능한 수준)
  for (let i = 0; i < imageData.data.length; i += 4) {
    const noise = Math.floor(Math.random() * 3) - 1; // -1, 0, 1
    imageData.data[i] = Math.max(0, Math.min(255, imageData.data[i] + noise));
  }

  return imageData;
};

// toDataURL도 동일하게 처리
const originalToDataURL = HTMLCanvasElement.prototype.toDataURL;
HTMLCanvasElement.prototype.toDataURL = function(...args) {
  const context = this.getContext('2d');
  if (context) {
    const imageData = context.getImageData(0, 0, this.width, this.height);
    // 노이즈가 이미 추가된 데이터 사용
  }
  return originalToDataURL.apply(this, args);
};
```

핵심 포인트:
- Canvas Fingerprinting은 각 디바이스의 고유한 렌더링 특성을 추적
- 완전히 차단하면 의심을 살 수 있으므로 미세한 노이즈 추가
- 매 요청마다 다른 지문을 생성하여 추적 방지

2.5 인간적인 행동 패턴 구현

목적: 실제 사용자의 마우스/키보드 입력 패턴 모방

구현 코드:
```javascript
// 랜덤 지연 함수
function randomDelay(min = 1000, max = 3000) {
  return new Promise(resolve =>
    setTimeout(resolve, min + Math.random() * (max - min))
  );
}

// 자연스러운 스크롤
async function naturalScroll(win, targetY) {
  const steps = 5 + Math.floor(Math.random() * 5); // 5-10 단계
  const stepSize = targetY / steps;

  for (let i = 0; i < steps; i++) {
    await win.webContents.executeJavaScript(`
      window.scrollBy({
        top: ${stepSize + (Math.random() - 0.5) * 50},
        behavior: 'smooth'
      });
    `);
    await randomDelay(100, 300);
  }
}

// 사람처럼 클릭
async function humanClick(win, selector) {
  // 요소까지 스크롤
  await win.webContents.executeJavaScript(`
    const element = document.querySelector('${selector}');
    if (element) {
      element.scrollIntoView({
        behavior: 'smooth',
        block: 'center'
      });
    }
```

```
  `);

  await randomDelay(500, 1500);

  // 마우스 이동 시뮬레이션 (옵션)
  await win.webContents.executeJavaScript(`
    const element = document.querySelector('${selector}');
    if (element) {
      const rect = element.getBoundingClientRect();
      const mouseEvent = new MouseEvent('mouseover', {
        bubbles: true,
        cancelable: true,
        view: window,
        clientX: rect.left + rect.width / 2,
        clientY: rect.top + rect.height / 2
      });
      element.dispatchEvent(mouseEvent);
    }
  `);

  await randomDelay(100, 300);

  // 클릭 실행
  await win.webContents.executeJavaScript(`
    document.querySelector('${selector}')?.click();
  `);
}

// 사람처럼 타이핑
async function humanType(win, selector, text) {
  await win.webContents.executeJavaScript(`
    new Promise(async (resolve) => {
      const input = document.querySelector('${selector}');
      input.focus();

      for (let char of '${text}') {
        input.value += char;

        // input 이벤트 발생
        input.dispatchEvent(new Event('input', { bubbles: true }));

        // 타이핑 속도 변화 (50-200ms)
        await new Promise(r => setTimeout(r, 50 + Math.random() * 150));
      }

      resolve();
    });
```

```
  `);
}
`````

핵심 포인트:
- 모든 동작에 랜덤 지연 추가
- 스크롤은 여러 단계로 분할하여 자연스럽게
- 클릭 전 요소에 마우스 hover 이벤트 발생
- 타이핑 속도는 사람처럼 불규칙하게

2.6 Rate Limiting 및 요청 관리

목적: 비정상적인 트래픽 패턴 방지

구현 코드:
`````javascript
class CrawlerRateLimiter {
  constructor(options = {}) {
    this.maxRequestsPerMinute = options.maxRequestsPerMinute || 10;
    this.minDelay = options.minDelay || 2000;
    this.maxDelay = options.maxDelay || 5000;
    this.requestCount = 0;
    this.windowStart = Date.now();
    this.requestHistory = [];
  }

  async throttle() {
    const now = Date.now();

    // 1분 윈도우 초기화
    if (now - this.windowStart >= 60000) {
      this.requestCount = 0;
      this.windowStart = now;
      this.requestHistory = [];
    }

    // 요청 제한 체크
    if (this.requestCount >= this.maxRequestsPerMinute) {
      const waitTime = 60000 - (now - this.windowStart);
      console.log(`Rate limit reached. Waiting ${waitTime}ms`);
      await new Promise(resolve => setTimeout(resolve, waitTime));
      this.requestCount = 0;
      this.windowStart = Date.now();
    }
```

```
    // 랜덤 지연
    const delay = this.minDelay + Math.random() * (this.maxDelay - this.minDelay);
    await new Promise(resolve => setTimeout(resolve, delay));

    this.requestCount++;
    this.requestHistory.push(now);
  }

  // 통계 정보
  getStats() {
    return {
      requestCount: this.requestCount,
      timeRemaining: 60000 - (Date.now() - this.windowStart),
      averageDelay: this.calculateAverageDelay()
    };
  }

  calculateAverageDelay() {
    if (this.requestHistory.length < 2) return 0;

    let totalDelay = 0;
    for (let i = 1; i < this.requestHistory.length; i++) {
      totalDelay += this.requestHistory[i] - this.requestHistory[i - 1];
    }

    return totalDelay / (this.requestHistory.length - 1);
  }
}

// 사용 예시
const rateLimiter = new CrawlerRateLimiter({
  maxRequestsPerMinute: 8,
  minDelay: 3000,
  maxDelay: 7000
});

async function crawlPage(url) {
  await rateLimiter.throttle();
  // 크롤링 로직...
}
`````

핵심 포인트:
- 분당 요청 수를 보수적으로 설정 (8-10회 권장)
- 각 요청 간 2-7초의 랜덤 지연
- 요청 이력을 기록하여 패턴 분석 가능
- 급격한 트래픽 증가 방지
```
```

2.7 세션 및 쿠키 관리

목적: 정상적인 브라우저 세션 유지

구현 코드:
```javascript
const { session } = require('electron');
const path = require('path');

// Persistent 세션 생성
const persistentSession = session.fromPartition('persist:crawler', {
  cache: true
});

// 쿠키 저장 및 로드
class SessionManager {
  constructor(sessionName) {
    this.session = session.fromPartition(`persist:${sessionName}`);
    this.cookieStore = [];
  }

  // 쿠키 저장
  async saveCookies() {
    const cookies = await this.session.cookies.get({});
    this.cookieStore = cookies;
    return cookies;
  }

  // 쿠키 복원
  async restoreCookies() {
    for (const cookie of this.cookieStore) {
      const url = `http${cookie.secure ? 's' : ''}://${cookie.domain}${cookie.path}`;
      await this.session.cookies.set({
        url,
        name: cookie.name,
        value: cookie.value,
        domain: cookie.domain,
        path: cookie.path,
        secure: cookie.secure,
        httpOnly: cookie.httpOnly,
        expirationDate: cookie.expirationDate
      });
    }
  }
```

```
  // 특정 도메인 쿠키 가져오기
  async getCookiesForDomain(domain) {
    return await this.session.cookies.get({ domain });
  }

  // 세션 초기화
  async clearSession() {
    await this.session.clearStorageData();
    this.cookieStore = [];
  }
}

// 초기 세션 워밍업
async function warmupSession(win, baseUrl) {
  // 메인 페이지 방문
  await win.loadURL(baseUrl);
  await randomDelay(3000, 5000);

  // 몇 가지 일반적인 페이지 방문
  const commonPages = ['/about', '/categories', '/new'];
  for (const page of commonPages) {
    await win.loadURL(`${baseUrl}${page}`);
    await randomDelay(2000, 4000);
  }
}
`````

핵심 포인트:
- Persistent partition 사용으로 세션 유지
- 크롤링 전 "워밍업" 세션으로 정상 사용자 흉내
- 쿠키를 저장하여 재사용
- LocalStorage, SessionStorage도 함께 관리

---

3. 무신사 특화 구현

3.1 무신사 플랫폼 특성
- React 기반 SPA (Single Page Application)
- GraphQL API 사용
- 강력한 봇 감지 시스템 (Imperva/Incapsula 추정)
- 로그인 필요한 기능 존재

3.2 무신사 특화 헤더 설정

구현 코드:

`````javascript
async function setupMusinsaHeaders(session) {
  session.webRequest.onBeforeSendHeaders((details, callback) => {
    const url = new URL(details.url);

    // 무신사 도메인 요청에만 적용
    if (url.hostname.includes('musinsa.com')) {
      details.requestHeaders['Referer'] = 'https://www.musinsa.com/';
      details.requestHeaders['Origin'] = 'https://www.musinsa.com';

      // API 요청인 경우
      if (url.pathname.includes('/api/') || url.pathname.includes('/graphql')) {
        details.requestHeaders['X-Requested-With'] = 'XMLHttpRequest';
        details.requestHeaders['Content-Type'] = 'application/json';
      }
    }

    callback({ requestHeaders: details.requestHeaders });
  });
}
`````

3.3 무신사 제품 크롤링 워크플로우

구현 코드:
`````javascript
class MusinsaCrawler {
  constructor() {
    this.rateLimiter = new CrawlerRateLimiter({
      maxRequestsPerMinute: 6,
      minDelay: 5000,
      maxDelay: 10000
    });
    this.sessionManager = new SessionManager('musinsa');
  }

  async initialize(win) {
    this.win = win;

    // 세션 워밍업
    await this.warmup();
  }

  async warmup() {
    console.log('Starting session warmup...');

    // 메인 페이지
`````

```
  await this.win.loadURL('https://www.musinsa.com');
  await randomDelay(3000, 5000);

  // 카테고리 페이지
  await this.win.loadURL('https://www.musinsa.com/categories/item/001');
  await randomDelay(2000, 4000);

  // 스크롤 행동
  await naturalScroll(this.win, 1000);
  await randomDelay(1000, 2000);

  console.log('Warmup complete');
}

async crawlProductList(categoryUrl, maxPages = 5) {
  const products = [];

  for (let page = 1; page <= maxPages; page++) {
    await this.rateLimiter.throttle();

    const url = `${categoryUrl}?page=${page}`;
    await this.win.loadURL(url);

    // 페이지 로드 대기
    await this.win.webContents.executeJavaScript(`
      new Promise((resolve) => {
        if (document.readyState === 'complete') {
          resolve();
        } else {
          window.addEventListener('load', resolve);
        }
      });
    `);

    await randomDelay(2000, 4000);

    // 스크롤하면서 이미지 로드
    await naturalScroll(this.win, 2000);
    await randomDelay(1000, 2000);

    // 제품 데이터 추출
    const pageProducts = await this.win.webContents.executeJavaScript(`
      Array.from(document.querySelectorAll('.li_box')).map(item => {
        const link = item.querySelector('a');
        const brandEl = item.querySelector('.item_title');
        const nameEl = item.querySelector('.item_name');
        const priceEl = item.querySelector('.price');
```

```
        return {
          url: link ? link.href : null,
          brand: brandEl ? brandEl.textContent.trim() : null,
          name: nameEl ? nameEl.textContent.trim() : null,
          price: priceEl ? priceEl.textContent.trim() : null,
          imageUrl: item.querySelector('img') ? item.querySelector('img').src : null
        };
      }).filter(item => item.url);
    `);

    products.push(...pageProducts);

    console.log(`Page ${page}: Found ${pageProducts.length} products`);
  }

  return products;
}

async crawlProductDetail(productUrl) {
  await this.rateLimiter.throttle();

  await this.win.loadURL(productUrl);
  await randomDelay(3000, 5000);

  // 페이지 스크롤 (리뷰 등 로드)
  await naturalScroll(this.win, 1500);
  await randomDelay(1000, 2000);

  // 상세 정보 추출
  const productDetail = await this.win.webContents.executeJavaScript(`
    (() => {
      const detail = {};

      // 기본 정보
      detail.brand = document.querySelector('.product_article .product_title')?.textContent.trim();
      detail.name = document.querySelector('.product_article .product_subtitle')?.textContent.trim();
      detail.price = document.querySelector('.product_article .product_price')?.textContent.trim();

      // 옵션 정보
      detail.sizes = Array.from(document.querySelectorAll('.option_select option'))
        .map(opt => opt.textContent.trim())
        .filter(text => text && text !== '선택');

      // 상세 설명
      detail.description = document.querySelector('.product_article
.product_information')?.textContent.trim();
```

```
        // 이미지
        detail.images = Array.from(document.querySelectorAll('.product_img img'))
          .map(img => img.src);

        return detail;
      })();
    `);

    return productDetail;
  }
}

// 사용 예시
async function main() {
  const { BrowserWindow } = require('electron');

  const win = new BrowserWindow({
    width: 1920,
    height: 1080,
    webPreferences: {
      preload: path.join(__dirname, 'preload.js'),
      nodeIntegration: false,
      contextIsolation: true
    }
  });

  const crawler = new MusinsaCrawler();
  await crawler.initialize(win);

  // 카테고리별 제품 수집
  const products = await crawler.crawlProductList(
    'https://www.musinsa.com/categories/item/001',
    3
  );

  // 상세 정보 수집 (일부만)
  for (const product of products.slice(0, 5)) {
    const detail = await crawler.crawlProductDetail(product.url);
    console.log(detail);
  }
}
`````

핵심 포인트:
- 세션 워밍업으로 정상 사용자 흉내
- 페이지당 5-10초 간격 유지

- 스크롤 동작으로 Lazy Loading 트리거
- 선택자(selector)는 무신사 HTML 구조에 맞게 조정 필요

---

4. 추가 보안 고려사항

4.1 프록시 로테이션

목적: IP 기반 차단 우회

구현 코드:
```javascript
class ProxyRotator {
  constructor(proxyList) {
    this.proxies = proxyList;
    this.currentIndex = 0;
  }

  getNextProxy() {
    const proxy = this.proxies[this.currentIndex];
    this.currentIndex = (this.currentIndex + 1) % this.proxies.length;
    return proxy;
  }

  applyProxy(win) {
    const proxy = this.getNextProxy();

    win.webContents.session.setProxy({
      proxyRules: `${proxy.protocol}://${proxy.host}:${proxy.port}`,
      proxyBypassRules: 'localhost'
    });
  }
}

// 사용 예시
const proxyList = [
  { protocol: 'http', host: '1.2.3.4', port: 8080 },
  { protocol: 'http', host: '5.6.7.8', port: 8080 }
];

const proxyRotator = new ProxyRotator(proxyList);
```

4.2 불필요한 리소스 차단

목적: 로딩 속도 향상 및 봇티 감소

구현 코드:
```javascript
function setupResourceBlocking(session) {
  const blockList = [
    'google-analytics.com',
    'googletagmanager.com',
    'facebook.net',
    'doubleclick.net',
    'hotjar.com',
    'crazyegg.com'
  ];

  session.webRequest.onBeforeRequest((details, callback) => {
    const shouldBlock = blockList.some(domain =>
      details.url.includes(domain)
    );

    callback({ cancel: shouldBlock });
  });
}
```

4.3 에러 처리 및 복구

구현 코드:
```javascript
class CrawlerErrorHandler {
  constructor(maxRetries = 3) {
    this.maxRetries = maxRetries;
  }

  async executeWithRetry(fn, context = null) {
    let lastError;

    for (let attempt = 1; attempt <= this.maxRetries; attempt++) {
      try {
        return await fn.call(context);
      } catch (error) {
        lastError = error;
        console.error(`Attempt ${attempt} failed:`, error.message);

        // 차단 감지 시 대기 시간 증가
        if (this.isBlockedError(error)) {
          const waitTime = Math.pow(2, attempt) * 60000; // 지수 백오프
          console.log(`Detected blocking. Waiting ${waitTime}ms`);
          await new Promise(resolve => setTimeout(resolve, waitTime));
```

```
        } else {
          await randomDelay(1000, 3000);
        }
      }
    }

    throw new Error(`Failed after ${this.maxRetries} attempts: ${lastError.message}`);
  }

  isBlockedError(error) {
    const blockIndicators = [
      '403',
      'captcha',
      'blocked',
      'rate limit'
    ];

    return blockIndicators.some(indicator =>
      error.message.toLowerCase().includes(indicator)
    );
  }
}
`````

---

5. 모니터링 및 로깅

5.1 크롤링 상태 모니터링

목적: 크롤링 성능 추적 및 이상 징후 조기 발견

구현 코드:
`````javascript
class CrawlerMonitor {
  constructor() {
    this.stats = {
      requestsSent: 0,
      requestsSucceeded: 0,
      requestsFailed: 0,
      totalDelay: 0,
      startTime: Date.now(),
      errors: []
    };
  }

  recordRequest(success, delay, error = null) {
```

```
    this.stats.requestsSent++;
    if (success) {
      this.stats.requestsSucceeded++;
    } else {
      this.stats.requestsFailed++;
      if (error) {
        this.stats.errors.push({
          timestamp: Date.now(),
          message: error.message,
          type: this.categorizeError(error)
        });
      }
    }
    this.stats.totalDelay += delay;
  }

  categorizeError(error) {
    const message = error.message.toLowerCase();
    if (message.includes('403') || message.includes('blocked')) return 'BLOCKED';
    if (message.includes('429') || message.includes('rate limit')) return 'RATE_LIMITED';
    if (message.includes('timeout')) return 'TIMEOUT';
    if (message.includes('captcha')) return 'CAPTCHA';
    return 'UNKNOWN';
  }

  getReport() {
    const elapsed = Date.now() - this.stats.startTime;
    const successRate = this.stats.requestsSent > 0
      ? (this.stats.requestsSucceeded / this.stats.requestsSent * 100).toFixed(2)
      : 0;
    const avgDelay = this.stats.requestsSent > 0
      ? (this.stats.totalDelay / this.stats.requestsSent).toFixed(0)
      : 0;
    const reqPerMin = elapsed > 0
      ? (this.stats.requestsSent / (elapsed / 60000)).toFixed(2)
      : 0;

    return {
      requestsSent: this.stats.requestsSent,
      requestsSucceeded: this.stats.requestsSucceeded,
      requestsFailed: this.stats.requestsFailed,
      elapsedTime: `${(elapsed / 1000).toFixed(0)}s`,
      successRate: `${successRate}%`,
      averageDelay: `${avgDelay}ms`,
      requestsPerMinute: reqPerMin,
      errorSummary: this.getErrorSummary()
    };
```

```
  }

  getErrorSummary() {
    const summary = {};
    this.stats.errors.forEach(error => {
      summary[error.type] = (summary[error.type] || 0) + 1;
    });
    return summary;
  }

  printReport() {
    console.log('\n========== Crawler Statistics ==========');
    const report = this.getReport();
    console.log(`Total Requests: ${report.requestsSent}`);
    console.log(`Succeeded: ${report.requestsSucceeded}`);
    console.log(`Failed: ${report.requestsFailed}`);
    console.log(`Success Rate: ${report.successRate}`);
    console.log(`Average Delay: ${report.averageDelay}`);
    console.log(`Requests/Minute: ${report.requestsPerMinute}`);
    console.log(`Elapsed Time: ${report.elapsedTime}`);

    if (Object.keys(report.errorSummary).length > 0) {
      console.log('\nError Summary:');
      Object.entries(report.errorSummary).forEach(([type, count]) => {
        console.log(`  ${type}: ${count}`);
      });
    }
    console.log('========================================\n');
  }

  // 실시간 모니터링 (주기적 출력)
  startPeriodicReport(intervalMs = 60000) {
    this.reportInterval = setInterval(() => {
      this.printReport();
    }, intervalMs);
  }

  stopPeriodicReport() {
    if (this.reportInterval) {
      clearInterval(this.reportInterval);
    }
  }
}
```

### 5.2 상세 로깅 시스템

**목적**: 디버깅 및 문제 추적을 위한 상세 로그

**구현 코드**:
```javascript
const fs = require('fs').promises;
const path = require('path');

class CrawlerLogger {
  constructor(logDir = './logs') {
    this.logDir = logDir;
    this.logFile = path.join(logDir, `crawler-${this.getTimestamp()}.log`);
    this.ensureLogDir();
  }

  async ensureLogDir() {
    try {
      await fs.mkdir(this.logDir, { recursive: true });
    } catch (error) {
      console.error('Failed to create log directory:', error);
    }
  }

  getTimestamp() {
    return new Date().toISOString().replace(/[:.]/g, '-').split('.')[0];
  }

  async log(level, message, data = null) {
    const timestamp = new Date().toISOString();
    const logEntry = {
      timestamp,
      level,
      message,
      data
    };

    // 콘솔 출력
    const consoleMessage = `[${timestamp}] [${level}] ${message}`;
    if (level === 'ERROR') {
      console.error(consoleMessage, data || '');
    } else if (level === 'WARN') {
      console.warn(consoleMessage, data || '');
    } else {
      console.log(consoleMessage, data || '');
    }

    // 파일 저장
    try {
```

```
      const logLine = JSON.stringify(logEntry) + '\n';
      await fs.appendFile(this.logFile, logLine, 'utf8');
    } catch (error) {
      console.error('Failed to write log:', error);
    }
  }

  async info(message, data) {
    await this.log('INFO', message, data);
  }

  async warn(message, data) {
    await this.log('WARN', message, data);
  }

  async error(message, data) {
    await this.log('ERROR', message, data);
  }

  async debug(message, data) {
    await this.log('DEBUG', message, data);
  }

  // 크롤링 세션 시작 로그
  async logSessionStart(config) {
    await this.info('Crawler session started', {
      config,
      nodeVersion: process.version,
      platform: process.platform
    });
  }

  // 크롤링 세션 종료 로그
  async logSessionEnd(stats) {
    await this.info('Crawler session ended', stats);
  }

  // 페이지 크롤링 로그
  async logPageCrawl(url, success, duration, error = null) {
    const level = success ? 'INFO' : 'ERROR';
    await this.log(level, `Crawled page: ${url}`, {
      success,
      duration: `${duration}ms`,
      error: error ? error.message : null
    });
  }
}
```

`````

핵심 포인트:
- 로그는 JSON 형식으로 저장하여 분석 용이
- 로그 레벨(INFO, WARN, ERROR, DEBUG)로 분류
- 타임스탬프와 함께 모든 이벤트 기록
- 파일과 콘솔 동시 출력

---

6. 실전 체크리스트

6.1 크롤링 시작 전 점검사항

필수 구현 항목:
- [ ] User Agent 및 HTTP 헤더 정상화 완료
- [ ] navigator.webdriver 속성 제거 확인
- [ ] Chrome 객체 추가 확인
- [ ] Electron 특화 객체 제거 (process, require 등)
- [ ] Canvas Fingerprinting 대응 구현
- [ ] Rate Limiter 설정 완료 (분당 6-10회)
- [ ] 랜덤 지연 함수 구현 (2-7초)
- [ ] 세션 관리자 구현
- [ ] 에러 핸들러 및 재시도 로직 구현

권장 구현 항목:
- [ ] 프록시 로테이션 시스템
- [ ] 리소스 차단 (광고, 분석 스크립트)
- [ ] 모니터링 및 로깅 시스템
- [ ] 스크롤/클릭 자연스러운 동작 구현
- [ ] 세션 워밍업 로직
- [ ] 메모리 관리 시스템

테스트 체크리스트:
- [ ] 단일 페이지 크롤링 성공 확인
- [ ] 연속 10개 페이지 크롤링 안정성 확인
- [ ] Rate Limiting 정상 작동 확인
- [ ] 에러 발생 시 재시도 로직 확인
- [ ] 메모리 누수 테스트 완료
- [ ] 로그 정상 기록 확인

6.2 운영 중 모니터링 지표

주요 모니터링 항목:

1. **성공률 (Success Rate)**
   - 목표: 90% 이상 유지
`````

   - 70% 이하 시: 경고 (크롤링 속도 감소)
   - 50% 이하 시: 위험 (크롤링 중단)

2. **평균 응답 시간 (Average Response Time)**
   - 정상: 2-5초
   - 주의: 10초 이상 (서버 부하 또는 차단 징후)
   - 위험: 30초 이상 (즉시 조사 필요)

3. **에러 패턴 (Error Patterns)**
   - 403 Forbidden: IP 또는 봇 차단
   - 429 Too Many Requests: Rate Limit 초과
   - 503 Service Unavailable: 서버 과부하
   - Timeout: 네트워크 또는 차단 문제

4. **요청 속도 (Request Rate)**
   - 목표: 분당 6-10회
   - 최대: 분당 15회 (단기간만)
   - 최소: 분당 3회 (안전 모드)

5. **세션 수명 (Session Lifetime)**
   - 이상적: 1시간 이상
   - 경고: 10분 이하 (세션 무효화 의심)

모니터링 대시보드 구현:
```javascript
class MonitoringDashboard {
  constructor(monitor) {
    this.monitor = monitor;
    this.alerts = [];
  }

  checkHealth() {
    const report = this.monitor.getReport();
    const health = {
      status: 'HEALTHY',
      alerts: []
    };

    // 성공률 체크
    const successRate = parseFloat(report.successRate);
    if (successRate < 50) {
      health.status = 'CRITICAL';
      health.alerts.push({
        type: 'CRITICAL',
        message: `Success rate critically low: ${report.successRate}`
      });
    } else if (successRate < 70) {
```

```
    health.status = 'WARNING';
    health.alerts.push({
      type: 'WARNING',
      message: `Success rate declining: ${report.successRate}`
    });
  }

  // 요청 속도 체크
  const reqPerMin = parseFloat(report.requestsPerMinute);
  if (reqPerMin > 15) {
    health.alerts.push({
      type: 'WARNING',
      message: `Request rate too high: ${reqPerMin}/min`
    });
  }

  // 에러 패턴 체크
  const errorSummary = report.errorSummary || {};
  if (errorSummary.BLOCKED > 3) {
    health.status = 'CRITICAL';
    health.alerts.push({
      type: 'CRITICAL',
      message: `Multiple blocking detected: ${errorSummary.BLOCKED} times`
    });
  }

  if (errorSummary.RATE_LIMITED > 2) {
    health.status = 'WARNING';
    health.alerts.push({
      type: 'WARNING',
      message: `Rate limiting detected: ${errorSummary.RATE_LIMITED} times`
    });
  }

  return health;
}

printDashboard() {
  const report = this.monitor.getReport();
  const health = this.checkHealth();

  console.log('\n' + '='.repeat(60));
  console.log('           CRAWLER MONITORING DASHBOARD');
  console.log('='.repeat(60));
  console.log(`Status: ${health.status}`);
  console.log('-'.repeat(60));
  console.log(`Requests: ${report.requestsSent} (✓ ${report.requestsSucceeded} / ✗
```

```
${report.requestsFailed})`);
    console.log(`Success Rate: ${report.successRate}`);
    console.log(`Speed: ${report.requestsPerMinute} req/min`);
    console.log(`Avg Delay: ${report.averageDelay}`);
    console.log(`Uptime: ${report.elapsedTime}`);

    if (health.alerts.length > 0) {
      console.log('-'.repeat(60));
      console.log('ALERTS:');
      health.alerts.forEach(alert => {
        console.log(`  [${alert.type}] ${alert.message}`);
      });
    }

    console.log('='.repeat(60) + '\n');
  }

  startAutoDisplay(intervalMs = 30000) {
    this.displayInterval = setInterval(() => {
      this.printDashboard();
    }, intervalMs);
  }

  stopAutoDisplay() {
    if (this.displayInterval) {
      clearInterval(this.displayInterval);
    }
  }
}
`````

---

7. 위험 요소 및 대응 방안

7.1 주요 위험 요소

높은 위험 (즉시 대응 필요):

1. **IP 차단**
   - 증상: 403 에러, 연속적인 타임아웃
   - 원인: 동일 IP에서 과도한 요청
   - 영향: 해당 IP로 접근 불가

2. **CAPTCHA 발동**
   - 증상: CAPTCHA 페이지로 리다이렉트
   - 원인: 봇 행동 패턴 감지
```
```

   - 영향: 수동 해결 전까지 접근 차단

3. **계정 정지**
   - 증상: 로그인 불가, 계정 잠김
   - 원인: 로그인 상태에서 비정상 행동
   - 영향: 계정 복구 필요

중간 위험 (모니터링 필요):

4. **Session 무효화**
   - 증상: 반복적인 로그아웃, 쿠키 삭제
   - 원인: 의심스러운 세션 활동
   - 영향: 재로그인 필요

5. **Rate Limiting**
   - 증상: 429 에러, "Too Many Requests"
   - 원인: 요청 빈도 초과
   - 영향: 일시적 접근 제한 (수분~수십분)

낮은 위험 (정상 범위):

6. **일시적 서버 에러**
   - 증상: 500, 503 에러
   - 원인: 서버 측 문제
   - 영향: 재시도로 해결 가능

7.2 대응 전략 구현

위험 감지 및 자동 대응 시스템:
```javascript
class RiskMitigationStrategy {
  constructor(options = {}) {
    this.warningThreshold = options.warningThreshold || 0.7;
    this.criticalThreshold = options.criticalThreshold || 0.5;
    this.blockDetectionCount = 0;
    this.lastBlockTime = null;
  }

  async evaluateRisk(monitor) {
    const report = monitor.getReport();
    const successRate = parseFloat(report.successRate) / 100;
    const errorSummary = report.errorSummary || {};

    // 차단 감지
    if (errorSummary.BLOCKED > 0) {
      this.blockDetectionCount++;
      this.lastBlockTime = Date.now();
```

```
  }

  // 위험도 평가
  if (this.blockDetectionCount >= 3 || successRate < this.criticalThreshold) {
    return {
      level: 'CRITICAL',
      action: 'STOP',
      message: 'Multiple blocks detected or success rate critically low'
    };
  }

  if (errorSummary.RATE_LIMITED > 2 || successRate < this.warningThreshold) {
    return {
      level: 'WARNING',
      action: 'SLOW_DOWN',
      message: 'Rate limiting detected or success rate declining'
    };
  }

  if (errorSummary.CAPTCHA > 0) {
    return {
      level: 'CRITICAL',
      action: 'MANUAL_INTERVENTION',
      message: 'CAPTCHA detected - manual intervention required'
    };
  }

  return {
    level: 'NORMAL',
    action: 'CONTINUE',
    message: 'All systems normal'
  };
}

async handleSlowDown(rateLimiter, logger) {
  logger.warn('Slowing down crawler due to risk detection');

  // 지연 시간 2배 증가
  const oldMinDelay = rateLimiter.minDelay;
  const oldMaxDelay = rateLimiter.maxDelay;
  rateLimiter.minDelay *= 2;
  rateLimiter.maxDelay *= 2;

  // 요청 빈도 절반 감소
  const oldMaxReq = rateLimiter.maxRequestsPerMinute;
  rateLimiter.maxRequestsPerMinute = Math.max(
    3,
```

```
      Math.floor(rateLimiter.maxRequestsPerMinute * 0.5)
    );

    await logger.info('Rate limits adjusted', {
      before: { minDelay: oldMinDelay, maxDelay: oldMaxDelay, maxReq: oldMaxReq },
      after: {
        minDelay: rateLimiter.minDelay,
        maxDelay: rateLimiter.maxDelay,
        maxReq: rateLimiter.maxRequestsPerMinute
      }
    });

    // 쿨다운 대기 (5분)
    await logger.info('Cooling down for 5 minutes...');
    await new Promise(resolve => setTimeout(resolve, 300000));
  }

  async handleStop(logger) {
    await logger.error('Crawler stopped due to critical risk level', {
      blockCount: this.blockDetectionCount,
      lastBlockTime: this.lastBlockTime
    });

    throw new Error('CRITICAL_RISK: Crawler stopped for safety');
  }

  async handleManualIntervention(logger) {
    await logger.error('Manual intervention required - CAPTCHA detected');

    // 여기서 알림을 보내거나 크롤러를 일시 중지
    console.log('\n' + '!'.repeat(60));
    console.log('MANUAL INTERVENTION REQUIRED');
    console.log('CAPTCHA detected - please solve manually');
    console.log('!'.repeat(60) + '\n');

    // 30분 대기
    await new Promise(resolve => setTimeout(resolve, 1800000));
  }
}
```

구체적 대응 시나리오:

시나리오 1: 403 Forbidden 에러 발생
```javascript
async function handle403Error(crawler, logger) {
  await logger.error('403 Forbidden detected - possible IP block');
```

```
  // 1. 즉시 크롤링 중단
  crawler.pause();

  // 2. 현재 상태 저장
  await crawler.saveState();

  // 3. 대기 시간 설정 (30분)
  const waitTime = 30 * 60 * 1000;
  await logger.info(`Waiting ${waitTime / 60000} minutes before retry`);
  await new Promise(resolve => setTimeout(resolve, waitTime));

  // 4. IP 변경 (프록시 사용 시)
  if (crawler.proxyRotator) {
    crawler.proxyRotator.getNextProxy();
    await logger.info('Switched to new proxy');
  }

  // 5. User Agent 변경
  crawler.changeUserAgent();

  // 6. 더 보수적인 설정으로 재시작
  crawler.rateLimiter.maxRequestsPerMinute = 5;
  crawler.rateLimiter.minDelay = 10000;
  crawler.rateLimiter.maxDelay = 20000;

  await logger.info('Resuming with conservative settings');
  crawler.resume();
}
`````

시나리오 2: Rate Limiting (429) 발생
`````javascript
async function handle429Error(crawler, logger, retryAfter = null) {
  await logger.warn('429 Rate Limit detected');

  // Retry-After 헤더가 있으면 그 시간만큼 대기
  const waitTime = retryAfter
    ? parseInt(retryAfter) * 1000
    : 5 * 60 * 1000; // 기본 5분

  await logger.info(`Waiting ${waitTime / 1000} seconds as requested by server`);
  await new Promise(resolve => setTimeout(resolve, waitTime));

  // 요청 속도 감소
  crawler.rateLimiter.maxRequestsPerMinute = Math.max(
    3,
```

```
    Math.floor(crawler.rateLimiter.maxRequestsPerMinute * 0.7)
  );

  await logger.info('Reduced request rate', {
    newRate: crawler.rateLimiter.maxRequestsPerMinute
  });
}
```

**시나리오 3: CAPTCHA 감지**
```javascript
async function handleCaptcha(crawler, logger) {
  await logger.error('CAPTCHA detected - automated solving not implemented');

  // 알림 발송 (이메일, 슬랙 등)
  await sendAlert({
    type: 'CAPTCHA_DETECTED',
    message: 'Manual intervention required',
    timestamp: new Date().toISOString()
  });

  // 크롤러 일시 중지
  crawler.pause();

  // 수동 해결 대기 (또는 CAPTCHA 서비스 사용)
  console.log('Please solve CAPTCHA manually and press Enter to continue...');

  // 여기서는 자동으로 재개하지 않음
  throw new Error('CAPTCHA_INTERVENTION_REQUIRED');
}
```

---

## 8. 법적 및 윤리적 고려사항

### 8.1 법적 검토사항

**반드시 확인해야 할 항목**:

1. **robots.txt 확인**
```javascript
  async function checkRobotsTxt(baseUrl) {
    try {
      const robotsUrl = new URL('/robots.txt', baseUrl).href;
      const response = await fetch(robotsUrl);
      const content = await response.text();
```

```
      console.log('robots.txt content:');
      console.log(content);

      // User-agent: * 또는 특정 봇에 대한 Disallow 확인
      const lines = content.split('\n');
      const disallowedPaths = lines
        .filter(line => line.toLowerCase().startsWith('disallow:'))
        .map(line => line.split(':')[1].trim());

      return {
        exists: true,
        disallowedPaths
      };
    } catch (error) {
      return {
        exists: false,
        disallowedPaths: []
      };
    }
  }
`````

2. **이용약관 (Terms of Service) 검토**
   - 자동화 도구 사용 제한 여부
   - 데이터 수집 및 사용 제한
   - API 사용 권장 여부

3. **개인정보 보호법 준수**
   - 수집 데이터에 개인정보 포함 여부
   - 개인정보 처리 방침
   - GDPR, CCPA 등 관련 법규

4. **저작권법**
   - 수집한 콘텐츠의 사용 목적
   - 상업적 사용 여부
   - 저작권자의 권리 침해 여부

위험 최소화 전략:
- 공개된 데이터만 수집
- 개인정보는 수집하지 않거나 즉시 익명화
- 수집 목적을 명확히 하고 그 범위 내에서만 사용
- 필요한 최소한의 데이터만 수집
- 과도한 서버 부하를 주지 않는 속도 유지

8.2 윤리적 가이드라인

책임 있는 크롤링 원칙:

1. **서비스 존중**
   - 서비스 제공자의 리소스를 과도하게 소비하지 않기
   - 피크 시간대 피하기
   - 에러 발생 시 즉시 중단

2. **투명성**
   - User Agent에 실제 목적 명시 (가능한 경우)
   - 문의 시 크롤링 목적 명확히 설명
   - 필요 시 사전 허가 요청

3. **최소 수집**
   - 목적 달성에 필요한 최소한의 데이터만
   - 불필요한 이미지나 리소스는 다운로드하지 않기
   - 중복 요청 최소화

권장하지 않는 사용 사례:
- 경쟁사 방해 목적
- 서비스 복제 또는 대체
- 가격 조작 정보 수집
- 개인정보 대량 수집
- DDoS 공격과 유사한 과도한 요청

권장하는 사용 사례:
- 가격 비교 서비스
- 학술 연구 목적
- 시장 분석 및 통계
- 자사 콘텐츠 모니터링
- 공개 데이터 아카이빙

---

9. 성능 최적화

9.1 메모리 관리

목적: 장시간 크롤링 시 메모리 누수 방지

구현 코드:
```javascript
class MemoryManager {
  constructor(win, options = {}) {
    this.win = win;
    this.clearInterval = options.clearInterval || 50; // 50개 요청마다
    this.requestCount = 0;
    this.memoryThreshold = options.memoryThreshold || 500 * 1024 * 1024; // 500MB
```

```
}

async checkAndClear() {
  this.requestCount++;

  if (this.requestCount >= this.clearInterval) {
    const memUsage = process.memoryUsage();
    console.log('Memory usage:', {
      rss: `${(memUsage.rss / 1024 / 1024).toFixed(2)} MB`,
      heapUsed: `${(memUsage.heapUsed / 1024 / 1024).toFixed(2)} MB`,
      heapTotal: `${(memUsage.heapTotal / 1024 / 1024).toFixed(2)} MB`
    });

    // 임계값 초과 시 강제 정리
    if (memUsage.heapUsed > this.memoryThreshold) {
      console.log('Memory threshold exceeded. Forcing cleanup...');
      await this.forceCleanup();
    } else {
      await this.normalCleanup();
    }

    this.requestCount = 0;
  }
}

async normalCleanup() {
  console.log('Performing normal cleanup...');

  // 브라우저 캐시 정리
  await this.win.webContents.session.clearCache();

  // Storage 정리
  await this.win.webContents.session.clearStorageData({
    storages: ['cache', 'cachestorage']
  });
}

async forceCleanup() {
  console.log('Performing force cleanup...');

  // 모든 스토리지 정리
  await this.win.webContents.session.clearStorageData();

  // 페이지 리로드로 메모리 해제
  await this.win.loadURL('about:blank');

  // Node.js 가비지 컬렉션 (--expose-gc 플래그 필요)
```

```
    if (global.gc) {
      global.gc();
    }
  }

  getMemoryUsage() {
    const usage = process.memoryUsage();
    return {
      rss: usage.rss,
      heapUsed: usage.heapUsed,
      heapTotal: usage.heapTotal,
      external: usage.external,
      arrayBuffers: usage.arrayBuffers
    };
  }

  startMonitoring(intervalMs = 60000) {
    this.monitorInterval = setInterval(() => {
      const usage = this.getMemoryUsage();
      console.log('Current memory:', {
        heap: `${(usage.heapUsed / 1024 / 1024).toFixed(2)} MB`,
        total: `${(usage.rss / 1024 / 1024).toFixed(2)} MB`
      });
    }, intervalMs);
  }

  stopMonitoring() {
    if (this.monitorInterval) {
      clearInterval(this.monitorInterval);
    }
  }
}

// 사용 예시
const memManager = new MemoryManager(win, {
  clearInterval: 30,
  memoryThreshold: 400 * 1024 * 1024 // 400MB
});

// 크롤링 중
await memManager.checkAndClear();
```

9.2 병렬 처리 최적화

목적: 여러 페이지를 동시에 크롤링하여 효율성 향상

구현 코드:
`````javascript
class ParallelCrawler {
  constructor(options = {}) {
    this.maxConcurrent = options.maxConcurrent || 3;
    this.activeWorkers = 0;
    this.queue = [];
    this.results = [];
    this.rateLimiter = options.rateLimiter;
  }

  async addTask(task, priority = 0) {
    return new Promise((resolve, reject) => {
      this.queue.push({
        task,
        priority,
        resolve,
        reject,
        addedAt: Date.now()
      });

      // 우선순위로 정렬
      this.queue.sort((a, b) => b.priority - a.priority);

      this.processQueue();
    });
  }

  async processQueue() {
    while (this.activeWorkers < this.maxConcurrent && this.queue.length > 0) {
      const { task, resolve, reject } = this.queue.shift();
      this.activeWorkers++;

      // Rate limiting 적용
      if (this.rateLimiter) {
        await this.rateLimiter.throttle();
      }

      task()
        .then(result => {
          this.results.push(result);
          resolve(result);
        })
        .catch(error => {
          console.error('Task failed:', error.message);
          reject(error);
        })
`````

```
        .finally(() => {
          this.activeWorkers--;
          this.processQueue();
        });
    }
  }

  async crawlMultiple(urls, crawlFunction) {
    const tasks = urls.map(url =>
      () => crawlFunction(url)
    );

    const promises = tasks.map(task => this.addTask(task));
    return await Promise.allSettled(promises);
  }

  getStats() {
    return {
      activeWorkers: this.activeWorkers,
      queueLength: this.queue.length,
      completedTasks: this.results.length
    };
  }
}

// 사용 예시
const parallelCrawler = new ParallelCrawler({
  maxConcurrent: 3,
  rateLimiter: rateLimiter
});

const urls = [
  'https://example.com/page1',
  'https://example.com/page2',
  'https://example.com/page3'
];

const results = await parallelCrawler.crawlMultiple(urls, async (url) => {
  return await crawlPage(url);
});
`````

주의사항:
- 병렬 처리 시에도 전체 Rate Limit 준수
- 각 Worker에 별도의 BrowserWindow 사용 권장
- 메모리 사용량 모니터링 필수
- 에러 전파 방지를 위한 적절한 에러 핸들링

9.3 캐싱 전략

목적: 중복 요청 방지 및 성능 향상

구현 코드:
```javascript
class CrawlerCache {
  constructor(options = {}) {
    this.cache = new Map();
    this.maxAge = options.maxAge || 3600000; // 1시간
    this.maxSize = options.maxSize || 1000;
  }

  set(key, value) {
    // 캐시 크기 제한
    if (this.cache.size >= this.maxSize) {
      const firstKey = this.cache.keys().next().value;
      this.cache.delete(firstKey);
    }

    this.cache.set(key, {
      value,
      timestamp: Date.now()
    });
  }

  get(key) {
    const item = this.cache.get(key);

    if (!item) return null;

    // 만료 확인
    if (Date.now() - item.timestamp > this.maxAge) {
      this.cache.delete(key);
      return null;
    }

    return item.value;
  }

  has(key) {
    return this.get(key) !== null;
  }

  clear() {
    this.cache.clear();
```

```
  }

  getStats() {
    return {
      size: this.cache.size,
      maxSize: this.maxSize,
      maxAge: this.maxAge
    };
  }
}

// 사용 예시
const cache = new CrawlerCache({ maxAge: 3600000, maxSize: 500 });

async function crawlWithCache(url) {
  // 캐시 확인
  if (cache.has(url)) {
    console.log(`Cache hit: ${url}`);
    return cache.get(url);
  }

  // 캐시 미스 - 실제 크롤링
  console.log(`Cache miss: ${url}`);
  const data = await crawlPage(url);

  // 캐시 저장
  cache.set(url, data);

  return data;
}
````표
2. **평균 응답 시간**: 급격한 증가 시 차단 의심
3. **에러 패턴**: 403, 429 에러 발생 빈도
4. **요청 속도**: 분당 6-10회 유지
5. **세션 수명**: 장기간 유지될수록 신뢰도 증가

---

## 7. 위험 요소 및 대응 방안

### 7.1 주요 위험 요소

**높은 위험**:
- IP 차단: 동일 IP에서 과도한 요청
- CAPTCHA 발동: 봇 행동 패턴 감지
- 계정 정지: 로그인 상태에서 비정상 행동
```
```

중간 위험:
- Session 무효화: 쿠키 삭제 또는 만료
- Rate Limiting: 일시적 접근 제한
- User-Agent 블랙리스트: 특정 UA 차단

낮은 위험:
- JavaScript Challenge: Cloudflare 등의 JS 검증
- Cookie 검증: 필수 쿠키 누락
- Referer 검증: 잘못된 Referer 헤더

7.2 대응 전략

예방적 조치:
```javascript
class RiskMitigationStrategy {
  constructor() {
    this.warningThreshold = 0.7; // 성공률 70% 이하 시 경고
    this.criticalThreshold = 0.5; // 성공률 50% 이하 시 중단
  }

  async evaluateRisk(monitor) {
    const report = monitor.getReport();
    const successRate = parseFloat(report.successRate) / 100;

    if (successRate < this.criticalThreshold) {
      console.error('CRITICAL: Success rate too low. Stopping crawler.');
      return 'STOP';
    }

    if (successRate < this.warningThreshold) {
      console.warn('WARNING: Success rate declining. Increasing delays.');
      return 'SLOW_DOWN';
    }

    return 'CONTINUE';
  }

  async handleSlowDown(rateLimiter) {
    // 지연 시간 2배 증가
    rateLimiter.minDelay *= 2;
    rateLimiter.maxDelay *= 2;

    // 요청 빈도 감소
    rateLimiter.maxRequestsPerMinute = Math.max(
      3,
      Math.floor(rateLimiter.maxRequestsPerMinute * 0.5)
    );
```

```
    console.log('Adjusted rate limits:', {
      minDelay: rateLimiter.minDelay,
      maxDelay: rateLimiter.maxDelay,
      maxRequestsPerMinute: rateLimiter.maxRequestsPerMinute
    });

    // 추가 대기
    await new Promise(resolve => setTimeout(resolve, 300000)); // 5분
  }
}
```

반응적 조치:
1. **403 에러 발생 시**:
   - 즉시 요청 중단
   - 10-30분 대기
   - IP 변경 (프록시 사용 시)
   - User Agent 변경

2. **CAPTCHA 발동 시**:
   - 수동 해결 또는 CAPTCHA 서비스 활용
   - 세션 재설정
   - 더 보수적인 크롤링 파라미터 적용

3. **연속 실패 시**:
   - 지수 백오프 적용
   - 크롤링 패턴 재검토
   - 로그 분석으로 원인 파악

---

8. 법적 및 윤리적 고려사항

8.1 법적 검토사항

준수해야 할 사항:
1. **robots.txt 확인**: 사이트의 크롤링 정책 존중
2. **이용약관 검토**: 자동화 도구 사용 제한 여부 확인
3. **개인정보 보호**: 수집 데이터에 개인정보 포함 여부 검토
4. **저작권**: 수집한 콘텐츠의 사용 목적 및 범위 명확화

위험 최소화 방법:
- 공개된 데이터만 수집
- 과도한 서버 부하를 주지 않는 속도로 크롤링
- 수집 목적의 명확성 (연구, 가격 비교 등)
- 필요한 데이터만 선별적으로 수집

8.2 윤리적 가이드라인

권장사항:
- 서비스 제공자의 리소스를 과도하게 소비하지 않기
- 크롤링 목적이 해당 서비스를 해치지 않는지 확인
- 가능하면 공식 API 사용 검토
- 데이터 수집 후 적절한 보안 조치

---

9. 성능 최적화

9.1 메모리 관리

구현 코드:
```javascript
class MemoryManager {
  constructor(win) {
    this.win = win;
    this.clearInterval = 50; // 50개 요청마다 정리
    this.requestCount = 0;
  }

  async checkAndClear() {
    this.requestCount++;

    if (this.requestCount >= this.clearInterval) {
      console.log('Clearing browser cache and memory...');

      // 캐시 정리
      await this.win.webContents.session.clearCache();

      // JavaScript 가비지 컬렉션 강제 실행
      await this.win.webContents.executeJavaScript(`
        if (window.gc) window.gc();
      `);

      // 불필요한 리소스 정리
      await this.win.webContents.executeJavaScript(`
        // 이미지 캐시 정리
        document.querySelectorAll('img').forEach(img => {
          img.src = '';
        });

        // 큰 DOM 노드 제거
        document.body.innerHTML = '';
```

```
      `);

      this.requestCount = 0;
    }
  }

  getMemoryUsage() {
    return process.memoryUsage();
  }
}
```

9.2 병렬 처리

구현 코드:
```javascript
class ParallelCrawler {
  constructor(maxConcurrent = 3) {
    this.maxConcurrent = maxConcurrent;
    this.activeWorkers = 0;
    this.queue = [];
  }

  async addTask(task) {
    return new Promise((resolve, reject) => {
      this.queue.push({ task, resolve, reject });
      this.processQueue();
    });
  }

  async processQueue() {
    while (this.activeWorkers < this.maxConcurrent && this.queue.length > 0) {
      const { task, resolve, reject } = this.queue.shift();
      this.activeWorkers++;

      task()
        .then(resolve)
        .catch(reject)
        .finally(() => {
          this.activeWorkers--;
          this.processQueue();
        });
    }
  }
}

// 사용 예시
```

```
const parallelCrawler = new ParallelCrawler(3);

for (const url of urls) {
  parallelCrawler.addTask(async () => {
    return await crawlPage(url);
  });
}
```

주의사항:
- 병렬 처리 시 Rate Limiting 더욱 주의
- 각 Worker는 별도의 BrowserWindow 사용 권장
- 메모리 사용량 모니터링 필수

---

10. 완전한 구현 예시

10.1 통합 크롤러 클래스

구현 코드:
```javascript
const { app, BrowserWindow, session } = require('electron');
const path = require('path');

class CompleteCrawler {
  constructor(config = {}) {
    this.config = {
      maxRequestsPerMinute: config.maxRequestsPerMinute || 8,
      minDelay: config.minDelay || 3000,
      maxDelay: config.maxDelay || 7000,
      maxRetries: config.maxRetries || 3,
      userAgent: config.userAgent || 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36',
      ...config
    };

    this.rateLimiter = new CrawlerRateLimiter(this.config);
    this.errorHandler = new CrawlerErrorHandler(this.config.maxRetries);
    this.monitor = new CrawlerMonitor();
    this.riskStrategy = new RiskMitigationStrategy();
    this.memoryManager = null;
  }

  async initialize() {
    await app.whenReady();

    this.win = new BrowserWindow({
```

```
    width: 1920,
    height: 1080,
    show: false, // 헤드리스 모드
    webPreferences: {
      preload: path.join(__dirname, 'preload.js'),
      nodeIntegration: false,
      contextIsolation: true,
      webSecurity: true
    }
  });

  this.memoryManager = new MemoryManager(this.win);

  // 헤더 설정
  this.setupHeaders();

  // 리소스 차단
  this.setupResourceBlocking();

  console.log('Crawler initialized');
}

setupHeaders() {
  const ses = this.win.webContents.session;

  ses.setUserAgent(this.config.userAgent);

  ses.webRequest.onBeforeSendHeaders((details, callback) => {
    details.requestHeaders['Accept-Language'] = 'ko-KR,ko;q=0.9';
    details.requestHeaders['Accept-Encoding'] = 'gzip, deflate, br';
    details.requestHeaders['Sec-Fetch-Dest'] = 'document';
    details.requestHeaders['Sec-Fetch-Mode'] = 'navigate';
    details.requestHeaders['Sec-Fetch-Site'] = 'same-origin';

    callback({ requestHeaders: details.requestHeaders });
  });
}

setupResourceBlocking() {
  const ses = this.win.webContents.session;
  const blockList = [
    'google-analytics.com',
    'googletagmanager.com',
    'facebook.net',
    'doubleclick.net'
  ];
```

```
  ses.webRequest.onBeforeRequest((details, callback) => {
    const shouldBlock = blockList.some(domain =>
      details.url.includes(domain)
    );
    callback({ cancel: shouldBlock });
  });
}

async crawl(url) {
  return await this.errorHandler.executeWithRetry(async () => {
    // Rate limiting
    await this.rateLimiter.throttle();

    const startTime = Date.now();

    try {
      // 페이지 로드
      await this.win.loadURL(url);
      await this.waitForPageLoad();
      await randomDelay(2000, 4000);

      // 데이터 추출
      const data = await this.extractData();

      // 모니터링
      const delay = Date.now() - startTime;
      this.monitor.recordRequest(true, delay);

      // 메모리 관리
      await this.memoryManager.checkAndClear();

      // 위험 평가
      const risk = await this.riskStrategy.evaluateRisk(this.monitor);
      if (risk === 'STOP') {
        throw new Error('Risk threshold exceeded. Stopping crawler.');
      } else if (risk === 'SLOW_DOWN') {
        await this.riskStrategy.handleSlowDown(this.rateLimiter);
      }

      return data;

    } catch (error) {
      const delay = Date.now() - startTime;
      this.monitor.recordRequest(false, delay);
      throw error;
    }
  });
```

```
}

async waitForPageLoad() {
  await this.win.webContents.executeJavaScript(`
    new Promise((resolve) => {
      if (document.readyState === 'complete') {
        resolve();
      } else {
        window.addEventListener('load', resolve);
      }
    });
  `);
}

async extractData() {
  // 구체적인 데이터 추출 로직은 사이트마다 다름
  return await this.win.webContents.executeJavaScript(`
    ({
      title: document.title,
      url: window.location.href,
      timestamp: Date.now()
    });
  `);
}

async crawlMultiple(urls) {
  const results = [];

  for (const url of urls) {
    try {
      const data = await this.crawl(url);
      results.push({ success: true, url, data });
    } catch (error) {
      console.error(`Failed to crawl ${url}:`, error.message);
      results.push({ success: false, url, error: error.message });
    }
  }

  return results;
}

getStats() {
  return this.monitor.getReport();
}

printStats() {
  this.monitor.printReport();
```

```
  }

  async cleanup() {
    if (this.win) {
      this.win.close();
    }
    app.quit();
  }
}

// 사용 예시
async function main() {
  const crawler = new CompleteCrawler({
    maxRequestsPerMinute: 6,
    minDelay: 5000,
    maxDelay: 10000
  });

  await crawler.initialize();

  const urls = [
    'https://example.com/page1',
    'https://example.com/page2',
    'https://example.com/page3'
  ];

  const results = await crawler.crawlMultiple(urls);

  console.log('Crawling complete. Results:', results);
  crawler.printStats();

  await crawler.cleanup();
}

// Electron app 시작
app.on('ready', main);
```

---

11. 결론 및 권장사항

11.1 핵심 요약

필수 구현 사항:
1. User Agent 및 헤더 정상화
2. WebDriver 속성 제거

3. Rate Limiting (분당 6-10회)
4. 랜덤 지연 (2-7초)
5. 자연스러운 행동 패턴

성공을 위한 핵심 원칙:
- **보수적 접근**: 느리게 시작하여 점진적으로 속도 조정
- **모니터링**: 실시간으로 성공률과 에러 추적
- **적응성**: 차단 징후 발견 시 즉시 전략 변경
- **지속성**: 장기적인 세션 유지로 신뢰도 구축

11.2 단계별 도입 계획

1단계 (필수):
- 기본 봇 감지 우회 (User Agent, WebDriver)
- Rate Limiting 구현
- 에러 핸들링

2단계 (권장):
- 자연스러운 행동 패턴 추가
- 세션 관리 및 워밍업
- 모니터링 시스템

3단계 (선택):
- 프록시 로테이션
- 병렬 처리
- 고급 메모리 최적화

11.3 추가 학습 자료

관련 기술 문서:
- Electron 공식 문서: https://www.electronjs.org/docs
- Chromium DevTools Protocol
- HTTP 헤더 표준 (RFC 7231)

봇 감지 기술 이해:
- Canvas Fingerprinting 원리
- TLS Fingerprinting
- Behavioral Analysis

11.4 최종 체크리스트

크롤러 배포 전 최종 점검:
- [ ] 모든 자동화 흔적 제거 확인
- [ ] Rate Limiting 적절히 설정
- [ ] 에러 핸들링 및 로깅 구현
- [ ] 메모리 누수 테스트 완료
- [ ] 법적 검토 완료

- [ ] 모니터링 대시보드 준비
- [ ] 긴급 중단 메커니즘 구현
- [ ] 백업 및 복구 계획 수립

부록 A: preload.js 전체 코드

```javascript
// preload.js
window.addEventListener('DOMContentLoaded', () => {
  // WebDriver 속성 제거
  Object.defineProperty(navigator, 'webdriver', {
    get: () => undefined,
    configurable: true
  });

  delete navigator.__proto__.webdriver;

  // Chrome 객체 추가
  if (!window.chrome) {
    window.chrome = {
      runtime: {},
      loadTimes: function() {},
      csi: function() {},
      app: {}
    };
  }

  // Plugins 정상화
  Object.defineProperty(navigator, 'plugins', {
    get: () => {
      return [
        {
          0: { type: 'application/pdf', suffixes: 'pdf', description: 'Portable Document Format' },
          name: 'Chrome PDF Plugin',
          filename: 'internal-pdf-viewer',
          description: 'Portable Document Format',
          length: 1
        },
        {
          0: { type: 'application/x-google-chrome-pdf', suffixes: 'pdf', description: '' },
          name: 'Chrome PDF Viewer',
          filename: 'mhjfbmdgcfjbbpaeojofohoefgiehjai',
          description: '',
          length: 1
        },
        {
```

```
        0: { type: 'application/x-nacl', suffixes: '', description: 'Native Client Executable' },
        1: { type: 'application/x-pnacl', suffixes: '', description: 'Portable Native Client Executable' },
        name: 'Native Client',
        filename: 'internal-nacl-plugin',
        description: '',
        length: 2
      }
    ];
  }
});

// Languages 정상화
Object.defineProperty(navigator, 'languages', {
  get: () => ['ko-KR', 'ko', 'en-US', 'en']
});

// Permission API 정상화
const originalQuery = window.navigator.permissions.query;
window.navigator.permissions.query = (parameters) => {
  return parameters.name === 'notifications'
    ? Promise.resolve({ state: Notification.permission })
    : originalQuery(parameters);
};

// Viewport 정상화
Object.defineProperty(window, 'outerWidth', {
  get: () => window.innerWidth
});

Object.defineProperty(window, 'outerHeight', {
  get: () => window.innerHeight
});

// Canvas Fingerprinting 대응
const originalGetImageData = CanvasRenderingContext2D.prototype.getImageData;
CanvasRenderingContext2D.prototype.getImageData = function(...args) {
  const imageData = originalGetImageData.apply(this, args);

  for (let i = 0; i < imageData.data.length; i += 4) {
    const noise = Math.floor(Math.random() * 3) - 1;
    imageData.data[i] = Math.max(0, Math.min(255, imageData.data[i] + noise));
  }

  return imageData;
};

// Electron 흔적 제거
```

```
  if (typeof process !== 'undefined') {
    delete window.process;
  }
  delete window.require;
  delete window.module;
  delete window.exports;

  console.log('Anti-detection measures applied');
});
```

부록 B: 유틸리티 함수 모음
```javascript
// utils.js

// 랜덤 지연
function randomDelay(min = 1000, max = 3000) {
  return new Promise(resolve =>
    setTimeout(resolve, min + Math.random() * (max - min))
  );
}

// 랜덤 User Agent 생성
function generateRandomUserAgent() {
  const versions = ['120', '121', '122', '123'];
  const version = versions[Math.floor(Math.random() * versions.length)];

  return `Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/${version}.0.0.0 Safari/537.36`;
}

// URL 정규화
function normalizeUrl(url) {
  try {
    const urlObj = new URL(url);
    return urlObj.href;
  } catch (error) {
    throw new Error(`Invalid URL: ${url}`);
  }
}

// 안전한 JSON 파싱
function safeJsonParse(text, defaultValue = null) {
  try {
    return JSON.parse(text);
```

```
  } catch (error) {
    console.error('JSON parse error:', error.message);
    return defaultValue;
  }
}

// 파일 저장
const fs = require('fs').promises;

async function saveResults(data, filename) {
  try {
    await fs.writeFile(
      filename,
      JSON.stringify(data, null, 2),
      'utf8'
    );
    console.log(`Results saved to ${filename}`);
  } catch (error) {
    console.error('Error saving results:', error.message);
  }
}

// 날짜 포맷팅
function formatTimestamp(timestamp = Date.now()) {
  return new Date(timestamp).toISOString().replace(/[:.]/g, '-');
}

module.exports = {
  randomDelay,
  generateRandomUserAgent,
  normalizeUrl,
  safeJsonParse,
  saveResults,
  formatTimestamp
};
```

문서 개정 이력

버전	날짜	변경 내용	작성자
1.0	2025-12-05	초기 문서 작성	-

면책 조항: 본 문서는 기술 교육 목적으로 작성되었습니다. 크롤링 활동은 대상 웹사이트의 이용약관 및 관련 법규를 준수해야 합니다. 무분별한 크롤링은 법적 책임을 초래할 수 있으며, 서비스 제공자에게 피해를 줄 수 있습니다.