SGT Client SDK - 사용자 가이드

v1.2.1 | 2026-04-17 | 이니넥스트

목적

이 문서는 SGT Client SDK를 사용하여 **안전한 AI 채팅 애플리케이션을 구축**하는 방법을 설명합니다.

대상 독자

- 웹 애플리케이션에서 SGT 플랫폼을 사용하려는 프론트엔드 개발자
- 안전한 AI 통신 기능을 구현해야 하는 개발자
- SGT SDK를 처음 사용하는 개발자

목차

> ⚠️ **v1.2.1 이전 버전 사용자 주의**
>
> `sdk.send()`, `sdk.secureFetch()`, `sdk.stopStream()`, `sdk.retryLast()` 는 제거되었습니다. 대신 Secure Mode를 켜 두고 브라우저 `fetch()` 를 그대로 쓰면, URL 정책에 걸리는 요청은 SDK가 알아서 암호화·서명·복호화합니다. 스트림 중단은 표준 `AbortController` 로 처리하면 됩니다.

---

1. 빠른 시작

1.1 1분 만에 시작하기

```html
<!DOCTYPE html>
<html>
<head>
  <title>SGT Chat</title>
</head>
<body>
  <div id="messages"></div>
  <input id="userInput" type="text" placeholder="메시지를 입력하세요">
  <button onclick="sendMessage()">전송</button>

  <!-- SGT SDK 로드 (UMD 전역 이름: SecureSDK) -->
  <script src="https://your-sgt-gateway.com/resources/client.sdk.min.js">
</script>

  <script>
    const sdk = new SecureSDK();
    const GATEWAY = 'https://your-sgt-gateway.com';

    async function init() {
      await sdk.initialize({
        serverUrl: GATEWAY,
        jwt: 'your-jwt-token',
        secureMode: {
          policies: [
            {
              name: 'chat',
              domain: 'your-sgt-gateway.com',
              include: ['/api/chat', '/v1/responses'],
              mode: 'chat'
            }
          ]
        }
      });
      console.log('SDK 초기화 완료');
    }

    async function sendMessage() {
      const input = document.getElementById('userInput');
      const message = input.value;
      input.value = '';

      // 일반 fetch() 호출 – SecureMode가 암호화/서명/복호화를 대신 처리
      const response = await fetch(`${GATEWAY}/api/chat`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          messages: [{ role: 'user', content: message }],
          stream: true
        })
      });

      // 스트림 응답은 복호화된 원본 SSE/NDJSON 바이트로 내려오므로 직접 파싱합니다.
```

```
      const reader = response.body.getReader();      const decoder = new
TextDecoder();      let buffer = '';      let full = '';      while (true) {
const { done, value } = await reader.read();        if (done) break;
buffer += decoder.decode(value, { stream: true });        const lines =
buffer.split('\n');        buffer = lines.pop() || '';        for (const line
of lines) {          const trimmed = line.trim();          // 메타데이터/이벤트
라인 건너뛰기          if (!trimmed || trimmed.startsWith('event:') ||
trimmed.startsWith('_meta:')) continue;          let jsonStr = trimmed;
if (trimmed.startsWith('data:')) {            jsonStr =
trimmed.slice(5).trim();            if (jsonStr === '[DONE]') continue;
}          try {            const chunk = JSON.parse(jsonStr);            //
Ollama / OpenAI 공통 content 추출 예시            const content =
chunk?.message?.content ||              chunk?.delta ||
chunk?.choices?.[0]?.delta?.content ||              '';            full +=
content;            document.getElementById('messages').innerHTML = full;
} catch (_) { /* 부분 라인은 버퍼로 이월 */ }        }      }    }    init();
</script></body></html>
```

---

2. 설치

Gateway 서버로부터 라이브러리를 로드합니다.

```
<script src="https://your-sgt-gateway.com/resources/client.sdk.min.js">
</script>
```

UMD 번들은 전역 `window.SecureSDK` 로 노출됩니다 (`rollup.config.js` 의 `output.name`).

```
const sdk = new SecureSDK();
```

`SecureSDK` 에는 커스텀 에러 클래스와 Wrapper 클래스가 정적 속성으로 붙어 있습니다: `SecureSDK.EncryptionError` , `SecureSDK.DecryptionError` , `SecureSDK.VerificationError` , `SecureSDK.SecureMode` , `SecureSDK.FetchWrapper` , `SecureSDK.UrlMatcher` .

---

3. 초기화와 Secure Mode

`initialize()` 는 헬스체크 → 키 교환(handshake) → Secure Mode 활성화 순으로 진행됩니다. Secure Mode 는 브라우저 `window.fetch` 를 래핑해 두고, URL 정책에 걸리는 요청에만 암호화를 끼워 넣습니다.

3.1 기본 초기화 (Secure Mode 활성화)

```
const sdk = new SecureSDK();

await sdk.initialize({
  // === 필수 옵션 ===
  serverUrl: 'https://your-sgt-gateway.com',
  jwt: 'your-jwt-token',

  // === Secure Mode 정책 (권장) ===
  secureMode: {
    policies: [
      {
        name: 'chat',                                    // 정책 이름 (로깅용)
        domain: 'your-sgt-gateway.com',                  // 글롭 지원:
'*.example.com'
        include: ['/api/chat', '/v1/responses'],         // 글롭: '/api/**'
        exclude: ['/api/v1/public/**'],                  // 선택
        mode: 'chat'                                     // 'chat' | 'proxy' (게이
트웨이 경로 선택)
      },
      {
        name: 'openapi',
        domain: '*',
        include: ['/openapi/**'],
        mode: 'proxy'
      }
    ],
    debug: false,       // true 시 [SGT:Policy] 로그 출력
    wrapFetch: true,    // 기본 true — window.fetch 래핑
    sessionId: null     // 기본 세션 ID (미지정 시 요청별 자동 생성)
  }
});
```

3.2 전체 옵션

```
await sdk.initialize({
  serverUrl: 'https://your-sgt-gateway.com',  // 필수
  jwt: 'your-jwt-token',                      // 필수

  // 암호화 알고리즘 선택
  cryptoType: 'bouncycastle',                 // 기본 (AES-256-CBC) |
'inicrypto' (SEED-CBC)

  // 타임아웃 (밀리초)
  timeoutChunk: 30000,                        // 청크 간 타임아웃
  timeoutTotal: 360000,                       // 전체 응답 타임아웃

  // 디스플레이 버퍼링 (타이핑 효과)
  enableDisplayBuffering: true,               // 기본 true
  displayInterval: 50,                        // 문자 표시 간격 (ms)

  secureMode: { policies: [ /* ... */ ] }
});

// cryptoEnabled 는 서버 handshake 응답에서 자동 수신되므로 클라이언트에서 설정하지 않
습니다.
```

3.3 게이트웨이 경로 모드 (mode)

값	게이트웨이 경로	OWASP 가드(Toxic/PII/Jailbreak)
`proxy`	`/sgt/v1/secure/api/proxy`	적용 안 됨 (범용 프록시)
`chat`	`/sgt/v1/secure/api/chat`	적용됨 (AI 채팅 안전 필터)

> **Note**
>
> 암호화/서명 여부는 어드민의 `cryptoEnabled` 핸드셰이크 플래그가 별도로 결정합니다. `mode`는 *어떤 가드 파이프라인을 통과시킬지*만 정합니다.

3.4 정책 평가 규칙

- `policies` 배열은 선언된 순서대로 훑고, 처음 걸리는 정책이 적용됩니다.
- 매칭 조건은 세 가지를 모두 만족해야 합니다: `domain` 일치, `include` 에 포함(또는 `include` 미지정), `exclude` 에 포함되지 않음.
- 어디에도 걸리지 않은 URL 은 원본 `fetch` 로 그냥 나갑니다 (암호화 없음).

3.5 Secure Mode 제어 API

```
sdk.isSecureModeEnabled();       // boolean
sdk.getSecureMode();             // SecureMode 인스턴스 (또는 null)
sdk.disableSecureMode();         // 원본 window.fetch 복원

// 원본 fetch 참조 (Secure Mode 활성 중에도 우회 호출이 필요할 때)
const { fetch: originalFetch } = sdk.getOriginalApis();
await originalFetch('/external', { /* ... */ });   // 암호화 없이
```

3.6 초기화 에러 처리

```
try {
  await sdk.initialize({ /* ... */ });
} catch (error) {
  if (error.message.includes('serverUrl')) {
    console.log('서버 URL을 확인하세요');
  } else if (error.message.includes('jwt')) {
    console.log('JWT 토큰을 확인하세요');
  } else if (error.message.includes('Health check failed')) {
    console.log('Gateway 가 실행 중인지 확인하세요');
  } else if (error.message.includes('secureMode.policies')) {
    console.log('policies 배열이 비어있거나 형식이 잘못되었습니다');
  }
}
```

---

4. API 호출 (fetch)

Secure Mode 가 켜져 있으면 모든 호출이 표준 `fetch()` 그대로입니다. 뒤에서 FetchWrapper 가 다음과 같은 변환을 대신 해 주므로, 애플리케이션 코드에서 신경 쓸 필요는 없습니다.

- GET/POST/PUT/DELETE 가릴 것 없이 Gateway 로 향하는 POST 하나로 통일됩니다.
- 원본 URL 과 메서드는 `X-Target-URL`, `X-Target-Method` 헤더에 실려 갑니다.
- Gateway 엔드포인트는 `mode` 로 갈립니다 — `chat` 은 `/sgt/v1/secure/api/chat`, `proxy` 는 `/sgt/v1/secure/api/proxy` 입니다.

- body 는 암호화 후 `{ encrypted, signed, message, signature, keyId, requestId, sessionId, ... }` 로 래핑됩니다.

4.1 일반 요청 (Non-Streaming)

응답이 돌아올 때는 원본 메시지 객체에 `_meta` 필드가 끼워 들어간 형태로 복호화되어 있습니다.

```
const response = await fetch('https://your-sgt-gateway.com/api/chat', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    messages: [{ role: 'user', content: '안녕하세요' }],
    stream: false
  })
});

const data = await response.json();

// HTTP status 가 200 이어도 본문에 error 필드가 있을 수 있으므로 함께 확인
if (data.error) {
  console.error(`[${data._meta?.code}] ${data.error}`);
  console.error('상세:', data._meta?.detail);
  return;
}

// 정상 응답: 원본 LLM 응답이 그대로 펼쳐져 있고 _meta 만 덧붙어 있음
console.log('AI 응답:', data.message?.content);
console.log('세션:', data._meta?.sessionId);
```

응답 구조:

```
// 정상
{ ...<원본 LLM 응답>, _meta: { status, sessionId, requestId, keyId, messageType,
requiredRag } }

// 에러
{ error: '<복호화된 에러 메시지>', _meta: { status, code, detail, sessionId,
requestId, keyId } }
```

4.2 GET / PUT / DELETE

```
await fetch('/api/v1/users?page=1');

await fetch('/api/v1/resources/123', {
  method: 'PUT',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ name: 'Updated' })
});

await fetch('/api/v1/resources/123', { method: 'DELETE' });
```

4.3 대화 히스토리 포함

```
const history = [
  { role: 'user', content: '안녕하세요' },
  { role: 'assistant', content: '안녕하세요! 무엇을 도와드릴까요?' },
  { role: 'user', content: '오늘 날씨는?' }
];

await fetch(`${GATEWAY}/api/chat`, {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ messages: history, stream: true })
});
```

4.4 필터 우회 옵션 (관리자 권한 필요)

```
await fetch(`${GATEWAY}/api/chat`, {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    messages: [{ role: 'user', content: '민감한 내용' }],
    overrideToxicFilter: true,    // 유해 콘텐츠 필터 우회
    overridePrivacyFilter: true,  // 개인정보 필터 우회
    stream: true
  })
});
```

---

5. 스트리밍 응답

요청 body 에 `stream: true` 가 있거나 서버가 SSE/NDJSON 으로 응답하면 FetchWrapper 는 스트리밍 경로로 빠집니다. 이 경로에서 `response.body` 는 복호화된 원본 SSE/NDJSON 텍

스트가 담긴 `ReadableStream<Uint8Array>` 이며, 각 청크 뒤에 `_meta: {...}` 한 줄이 덧붙어 있습니다.

즉 복호화된 객체가 그대로 넘어오는 것이 아니라 여전히 바이트 스트림이므로, 호출 측에서 `TextDecoder` 로 디코딩하고 라인 단위로 직접 파싱해야 합니다.

5.1 NDJSON 스트림 (Ollama 등)

```
const response = await fetch(`${GATEWAY}/api/chat`, {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    messages: [{ role: 'user', content: '긴 설명을 해주세요' }],
    stream: true
  })
});

const reader = response.body.getReader();
const decoder = new TextDecoder();
let buffer = '';
let accumulated = '';

try {
  while (true) {
    const { done, value } = await reader.read();
    if (done) break;

    buffer += decoder.decode(value, { stream: true });
    const lines = buffer.split('\n');
    buffer = lines.pop() || '';

    for (const raw of lines) {
      const line = raw.trim();
      if (!line || line.startsWith('event:') || line.startsWith('_meta:'))
continue;

      // SSE "data:" prefix 지원
      const jsonStr = line.startsWith('data:') ? line.slice(5).trim() : line;
      if (!jsonStr || jsonStr === '[DONE]') continue;

      let chunk;
      try { chunk = JSON.parse(jsonStr); }
      catch { continue; }  // 불완전 라인

      // 에러 청크: { error: <msg>, _meta: { code, detail, ... } }
      if (chunk.error) {
        console.error(`[${chunk._meta?.code}] ${chunk.error}`);
        console.error('detail:', chunk._meta?.detail);
        break;
      }

      // 정상 청크: 공급자별 content 추출
      const content =
        chunk?.message?.content ||                    // Ollama
        chunk?.choices?.[0]?.delta?.content ||        // OpenAI
        chunk?.delta ||
        '';

      accumulated += content;
      updateChatUI(accumulated);
```

```
    }  }} finally {  reader.releaseLock();}
```

5.2 SSE 스트림 (OpenAI responses API 등)

SSE 응답은 `event:`, `data:`, `_meta:` 라인이 번갈아 들어옵니다.

```
event: response.output_text.delta
data: {"type":"response.output_text.delta","delta":"AI"}
_meta:
{"sessionId":"...","requestId":"...","keyId":123,"messageType":"normal"}
```

파싱 루프는 NDJSON 예제와 동일합니다. `event:` 와 `_meta:` 는 건너뛰고 `data:` 라인만 JSON 으로 풀면 됩니다.

5.3 메타데이터 (`_meta`)

청크마다 붙는 `_meta` 에는 다음 필드가 들어 있습니다. NDJSON 에서는 JSON 안에 포함되고, SSE 에서는 별도 라인으로 내려옵니다.

필드	설명
`status`	`'success'` 또는 `'error'`
`sessionId`, `requestId`, `keyId`	식별/추적
`messageType`	`'normal'` | `'final'` | `'thinking'`
`code`, `detail`	에러 응답일 때만

`_meta.messageType` 을 보고 UI 를 다르게 처리할 수 있습니다.

messageType	설명	UI 처리
`normal`	일반 LLM 응답	기존 메시지에 누적
`final`	최종 완료 메타데이터	완료 표시
`thinking`	추론 모델의 사고 과정	접을 수 있는 블록

```
// NDJSON 형식이면 chunk._meta 로 직접 접근
if (chunk._meta?.messageType === 'thinking') {
  updateThinkingPanel(chunk.message?.content || '');
  continue;
}
```

6. 스트림 중단 (AbortController)

스트림 중단은 표준 `AbortController` 하나로 끝납니다. signal 이 abort 되면 SDK 가 내부 스트림·reader·디스플레이 인터벌을 알아서 정리합니다.

6.1 사용자 "중단" 버튼

```
let currentController = null;

async function sendMessage(text) {
  currentController = new AbortController();

  try {
    const response = await fetch(`${GATEWAY}/api/chat`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ messages: [{ role: 'user', content: text }],
stream: true }),
      signal: currentController.signal
    });

    const reader = response.body.getReader();
    while (true) {
      const { done, value } = await reader.read();
      if (done) break;
      // ... 청크 처리
    }
  } catch (error) {
    if (error.name === 'AbortError') {
      console.log('사용자가 스트림을 중단했습니다');
    } else {
      throw error;
    }
  } finally {
    currentController = null;
  }
}

function handleStopButton() {
  if (currentController) {
    currentController.abort('user cancelled');
  }
}
```

6.2 타임아웃으로 자동 중단

```
const controller = new AbortController();
setTimeout(() => controller.abort(), 5000);   // 5초 후 자동 중단

await fetch('/api/v1/slow-endpoint', { signal: controller.signal });
```

6.3 재시도

`sdk.retryLast()` 는 더 이상 없습니다. 재시도가 필요하면 같은 요청을 `fetch()` 로 다시 부르면 됩니다. 중단 시점까지 받은 내용을 화면에 남겨 두고 싶다면 누적 문자열을 앱 상태로 따로 보관해 두었다가 UI 에 유지하세요.

```
async function retryLast(lastRequestBody) {
  return await fetch(`${GATEWAY}/api/chat`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(lastRequestBody)
  });
}
```

---

7. 에러 처리

SDK는 서버 / 암호화 / 서명 단계에서 발생한 에러를 세 가지 경로로 노출합니다.

7.1 스트림 중 에러 청크

스트리밍 도중 서버가 에러를 보내면 해당 라인은 다음 구조의 JSON이 됩니다.

```
// 라인 파싱 결과
{
  error: '<복호화된 사용자 메시지>',
  _meta: {
    status: 'error',
    code: '2001',           // 비즈니스 에러 코드
    detail: { ... },        // OWASP predictions 등 복호화된 상세
    sessionId, requestId, keyId
  }
}
```

```
try {
  const chunk = JSON.parse(jsonStr);
  if (chunk.error) {
    console.error('코드:', chunk._meta?.code);
    console.error('메시지:', chunk.error);
    console.error('상세:', chunk._meta?.detail);
    console.log('에러 전까지 누적된 내용:', accumulated);
    break;
  }
  // 정상 청크 처리 ...
} catch (_) {}
```

Note

부분 응답은 SDK가 별도로 보존하지 않습니다. 필요하면 앱에서 `accumulated` 문자열을 직접 유지하세요.

7.2 일반 응답 에러 구조

스트리밍이 아닌 경우에도 에러 형태는 동일합니다. HTTP status 가 200 으로 돌아올 수도 있으니, 성공/실패 판별은 본문의 `error` 필드 유무로 하세요.

```
const response = await fetch('/api/chat', { /* ... */ });
const data = await response.json();
// 성공: { ...<원본 응답>, _meta: { status: 'success', ... } }
// 실패: { error: '<메시지>', _meta: { status: 'error', code: '2001', detail:
{...} } }

if (data.error) {
  console.error(data._meta?.code, data.error);
}
```

7.3 네트워크 / 보안 예외

FetchWrapper는 암호화·복호화·검증 실패 시 커스텀 예외를 throw 합니다.

```
try {
  const response = await fetch('/api/chat', { /* ... */ });
  // ...
} catch (error) {
  if (error.name === 'AbortError') {
    // 사용자 중단
  } else if (error instanceof SecureSDK.EncryptionError) {
    // 요청 암호화/서명 실패
  } else if (error instanceof SecureSDK.DecryptionError) {
    // 응답 복호화 실패 → SDK 재초기화 권장
    await sdk.initialize({ /* ... */ });
  } else if (error instanceof SecureSDK.VerificationError) {
    // 응답 서명 검증 실패 → 재초기화 또는 관리자 문의
  } else {
    console.error('네트워크 오류:', error.message);
  }
}
```

7.4 주요 에러 코드

코드	설명	재시도	처리 방법
2001	유해 콘텐츠 감지	가능	`overrideToxicFilter: true`
2002	파일 내 유해 콘텐츠	가능	파일 수정 후 재업로드
2101	개인정보(PII) 감지	가능	`overridePrivacyFilter: true`
2102	파일 내 PII	가능	파일 수정 후 재업로드
2201	탈옥(Jailbreak) 시도	불가	입력 수정
2301	금지어 감지	불가	입력 수정
3001	파일 없음	불가	파일 경로 확인
3004	파일 크기 초과	불가	파일 크기 축소
4002	복호화 실패	불가	SDK 재초기화
4003	서명 검증 실패	불가	SDK 재초기화
5001	토큰 제한 초과	불가	입력 축소
5004	요청 횟수 초과	불가	일정 시간 대기
8001	서버 내부 오류	불가	재시도
8006	LLM 연결 실패	불가	재시도

7.5 필터 우회 재시도

유해 콘텐츠(2001)나 PII(2101) 는 관리자 권한이 있다면 `override*Filter` 로 우회해 재호출할 수 있습니다. 스트림을 읽다가 에러 라인을 만나면 같은 body 에 우회 옵션만 더해 다시 부릅니다.

```
async function sendWithFallback(body) {
  const response = await fetch(`${GATEWAY}/api/chat`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ ...body, stream: true })
  });

  const reader = response.body.getReader();
  const decoder = new TextDecoder();
  let buffer = '';

  while (true) {
    const { done, value } = await reader.read();
    if (done) return;

    buffer += decoder.decode(value, { stream: true });
    const lines = buffer.split('\n');
    buffer = lines.pop() || '';

    for (const raw of lines) {
      const line = raw.trim();
      if (!line || line.startsWith('event:') || line.startsWith('_meta:'))
continue;

      const jsonStr = line.startsWith('data:') ? line.slice(5).trim() : line;
      if (!jsonStr || jsonStr === '[DONE]') continue;

      let chunk;
      try { chunk = JSON.parse(jsonStr); } catch { continue; }

      if (chunk.error) {
        const code = chunk._meta?.code;
        if (code === '2001' && confirm('유해 콘텐츠 감지. 우회 진행?')) {
          return sendWithFallback({ ...body, overrideToxicFilter: true });
        }
        if (code === '2101' && confirm('PII 감지. 우회 진행?')) {
          return sendWithFallback({ ...body, overridePrivacyFilter: true });
        }
        return;
      }
      // 정상 청크 처리 ...
    }
  }
}
```

---

8. 고급 기능

8.1 로그 레벨

```
sdk.setLogLevel('trace');    // 모든 로그
sdk.setLogLevel('debug');    // 개발
sdk.setLogLevel('info');     // 기본
sdk.setLogLevel('warn');
sdk.setLogLevel('error');    // 프로덕션
sdk.setLogLevel('silent');   // 로그 비활성화
```

8.2 타임아웃 커스터마이징

```
await sdk.initialize({
  /* ... */
  timeoutChunk: 10000,      // 짧은 응답: 10초
  timeoutTotal: 30000       // 30초
});

// 긴 추론 응답을 다루는 경우
await sdk.initialize({
  /* ... */
  timeoutChunk: 60000,
  timeoutTotal: 600000      // 10분
});
```

8.3 암호화 타입

```
await sdk.initialize({
  /* ... */
  cryptoType: 'bouncycastle'    // 기본: AES-256-CBC
  // cryptoType: 'inicrypto'    // 한국 표준: SEED-CBC
});
```

8.4 Secure Mode 우회 (외부 API 호출)

정책에 안 걸리는 URL 은 자동으로 원본 `fetch` 로 나가지만, 특정 호출만 확실하게 우회하고 싶을 때 원본 참조를 직접 쓸 수 있습니다.

```
const { fetch: originalFetch } = sdk.getOriginalApis();
await originalFetch('https://external-api.com/data');
```

8.5 Secure Mode 일시 해제

```
sdk.disableSecureMode();   // 이후 window.fetch 는 원본 동작

// 재활성화가 필요하면 initialize() 재호출
await sdk.initialize({ /* ... */, secureMode: { policies: [...] } });
```

8.6 X-Target-URL 헤더를 통한 동적 라우팅

Gateway 는 `X-Target-URL` 헤더에 적힌 주소로 요청을 실제 업스트림 LLM 에 포워딩합니다. 보통은 FetchWrapper 가 이 헤더를 자동으로 채우지만, 업스트림을 직접 지정하고 싶다면 직접 실어 보내도 됩니다.

```
await fetch(`${GATEWAY}/proxy`, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'X-Target-URL': 'http://localhost:11434/api/chat'
  },
  body: JSON.stringify({ messages, stream: true })
});
```

8.7 세션 ID 관리

세션 ID 는 세 가지 방식으로 지정할 수 있습니다. 아무 것도 지정하지 않으면 SDK 가 요청마다 UUID 를 새로 만듭니다.

- `secureMode.sessionId` 로 기본값을 미리 잡아 두기
- 요청별로 `X-Session-Id` 헤더 지정
- (위 두 가지 모두 없으면) 자동 생성

```
await fetch('/api/chat', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'X-Session-Id': 'my-session-' + userId
  },
  body: JSON.stringify({ /* ... */ })
});
```