CLIP

문서 기준일 2026-10-28

API 명세서

(주)플로터랩

Floater Lab

Copyright © 2026 Floater Lab, Inc.
본 문서의 저작권은 (주)플로터랩에 있습니다.

목 차

시작하기 3

인증 3

요청 형식 4

비동기 작업(작업 ID 폴링) 5

오류 응답 6

처리 시간·재시도 7

데이터 처리와 보관 8

버전과 상태 확인 9

무역서류 점검 10

- 연동 순서 10
- 결과 읽는 법 11
- API 로 제공되지 않는 판정 11
- 제한과 주의사항 7
- 화면 작업과 API 12
- 적용 예시 14
- 파일 업로드 (POST /api/shipping/upload) 18
- 쪽 처리 시작 (POST /api/shipping/process) 20
- 쪽 처리 결과 조회 (GET /api/shipping/process) 22
- 파일 서류 종류 분류 (POST /api/shipping/classify) 26
- 파일 핵심정보 추출 (POST /api/shipping/extract) 29
- 파일 서류 분리·추출 시작 (POST /api/shipping/review) 32
- 서류 분리·추출 결과 조회 (GET /api/shipping/review) 33
- 운영 37

CLIP API

이 문서는 CLIP 설치본에서 사용할 수 있는 API 를 담고 있습니다. 예제의 주소는 실제 설치 주소로 바꿔 사용하십시오.

시작하기

CLIP HTTP API 는 무역서류 점검을(를) 고객 시스템에서 호출할 수 있도록 제공하는 API 입니다. 이 문서는 공통으로 적용되는 규칙과 기능별 호출 방법을 설명합니다.

모든 주소는 `https://<CLIP 설치 주소>` 을 기준으로 합니다. 실제 연동 시에는 담당자가 안내한 서버 주소로 바꿔 사용하십시오.

- 모든 요청은 고객 서버에서 호출해 주십시오. 브라우저나 모바일 앱에서 직접 호출할 수 없습니다(CORS 미지원).
- 인증이 필요한 모든 요청에는 `x-clip-key` 헤더로 API 키를 보내야 합니다. 인증 세부 사항은 다음 섹션을 참고하십시오.
- 파일을 보내는 요청은 multipart/form-data 를, 그 밖의 요청은 application/json 을 사용합니다.
- 일부 기능은 작업을 만든 뒤 작업 ID 로 결과를 조회하는 방식입니다. 자세한 내용은 비동기 작업 섹션을 참고하십시오.

연동을 시작할 때는 아래 순서로 서버 상태를 먼저 확인하는 것을 권장합니다. 상태 확인 경로는 인증 없이 호출할 수 있습니다.

첫 호출 예 CURL

```
# 1. 서버 동작 확인 (인증 불필요)
curl -s https://<CLIP 설치 주소>/api/health

# 2. 키 상태와 잔여 한도 확인
curl -s https://<CLIP 설치 주소>/api/key-status -H "x-clip-key: YOUR_API_KEY"
```

첫 번째 요청은 `{"status":"ok"}` 응답이면 서버가 정상 동작하는 것입니다. 두 번째 요청에서 키가 유효하면 요금제, 사용 가능한 기능, 한도 정보가 응답으로 옵니다.

인증

API 키는 모든 인증 요청의 `x-clip-key` 헤더로 보냅니다. 키 값을 요청 본문이나 주소의 쿼리 문자열에 넣지 마십시오.

CURL

```
curl -X POST https://<CLIP 설치 주소>/api/shipping/upload -H "x-clip-key: YOUR_API_KEY" -F "file=@/path/to/document.pdf"
```

API 키는 담당자에게 요청하여 발급받습니다. 키마다 사용할 수 있는 권한이 다르므로, 아래 표에서 연동할 기능에 필요한 권한을 확인하십시오.

기능	필요한 권한	비고
무역서류 파일 업로드·처리	`files` , `ocr`	업로드한 파일은 같은 키로만 처리할 수 있습니다
무역서류 분류·추출 (`/api/shipping/classify` , `/api/shipping/extract`)	`ocr`	문서 본문을 직접 보내는 방식입니다
키 상태·잔여 한도 확인	키 활성·만료 상태	`GET https://<CLIP 설치 주소>/api/key-status`

키에는 추가 기능(분류, 키 추출, 개인정보 탐지) 사용 여부가 설정되어 있을 수 있습니다. 설정이 꺼진 기능을 요청하면 403 이 옵니다.

인증 실패는 다음과 같이 구분됩니다.

- 키 헤더가 없거나 비어 있으면 401 이 오며, 본문의 `code` 값이 `api_key_required` 입니다. 배포 설정에 따라 키 헤더 없이도 요청이 처리될 수 있으나, 연동은 항상 키를 보내는 것을 전제로 하십시오.
- 키가 잘못되었거나 비활성·만료되었거나 권한이 없으면 401 또는 403 이 옵니다. 기능 API 응답의 `error` 문자열로 세부 사유를 확인할 수 있습니다. 다만 키 상태 확인 API(`GET /api/key-status`)는 업스트림 사유를 감추고 `{"error":"Upstream error 401"}` 처럼 상태 코드만 돌려주므로 401 과 403 을 구분할 수 없습니다. 이 API 의 502 본문은 `Failed to fetch: ...` 형식입니다.
- 분당 또는 일일 사용 한도를 넘으면 429 가 옵니다.

키 누락 응답 (401) JSON

```
{"error":"우측 상단에 API Key 를 입력해 주세요.","code":"api_key_required"}
```

API 키를 브라우저, 모바일 앱, 공개 저장소 등 클라이언트 쪽 코드에 넣지 마십시오. 키가 노출되었다고 판단되면 즉시 담당자에게 폐기와 재발급을 요청하십시오.

요청 형식

파일을 보내는 API 는 multipart/form-data 형식을 사용하며 파일 필드 이름은 `file` 입니다.

파일이 없는 POST 요청은 application/json 본문을 사용합니다. `Content-Type` 헤더를 `application/json` 으로 지정하고 JSON 본문을 보내십시오. 작업 조회는 GET 요청이며 본문이 없습니다.

multipart/form-data 예 CURL

```
curl -X POST https://<CLIP 설치 주소>/api/shipping/upload -H "x-clip-key: YOUR_API_KEY" -F "file=@/path/to/document.pdf"
```

기능별로 받는 파일 형식이 다릅니다. 아래 목록은 권장 형식이며, 목록에 없는 형식을 보내면 처리가 실패하거나 결과 품질이 떨어질 수 있습니다.

기능	지원 파일 형식
무역서류 점검	PDF, PNG, JPG(JPEG), TIF(TIFF), WEBP

파일 크기와 페이지 수 상한은 계정 설정에 따릅니다. 상한을 넘는 파일은 413 으로 거절됩니다. 정확한 상한 값은 담당자에게 확인하십시오. 이미지 해상도는 기본적으로 자동 축소되어 처리됩니다. 파일 크기, 페이지 수, 픽셀 상한을 넘으면 413 으로 거절됩니다.

> 파일 업로드 요청을 보낼 때는 `Content-Type` 을 직접 지정하지 마십시오. HTTP 클라이언트가 multipart 경계 값을 포함한 헤더를 자동으로 만들도록 두어야 합니다.

비동기 작업(작업 ID 폴링)

문서 처리는 시간이 오래 걸리므로 여러 기능이 같은 방식의 비동기 작업으로 동작합니다. 기본 흐름은 다음과 같습니다.

1. 파일을 업로드 API 로 보내면 `fileId` 를 응답으로 받습니다. 업로드 응답의 HTTP 상태는 200 입니다.
2. 처리 API 를 POST 로 호출하면 `{"jobId": "..."}` 형태의 응답을 받습니다. 이 값이 작업 ID 입니다.
3. 같은 처리 API 주소에 `?job=작업ID` 를 붙여 GET 으로 호출합니다. 3초 간격으로 조회하십시오.
4. 응답의 `status` 가 `pending` 이면 다시 조회하고, `done` 또는 `error` 이면 조회를 멈춥니다.

처리 API 의 요청 본문과 페이지 단위 호출 방식은 기능별 문서를 따르십시오.

status 값	의미	처리 방법
`pending`	처리 중입니다.	3초 뒤에 다시 조회합니다.
`done`	처리가 끝났습니다.	결과는 `result` 또는 `results` 필드에 있습니다. 필드 이름은 기능에 따라 다릅니다.

status 값	의미	처리 방법
`error`	처리에 실패했습니다.	본문의 `status` 는 항상 문자열 `"error"` 이고, `error` 필드에 사유가 옵니다. 실패 여부는 조회 응답의 HTTP 상태 코드로 판단하십시오. 처리·검토 API 는 처리 엔진의 상태 코드를 그대로 쓰며(알 수 없으면 502).
`unknown` (HTTP 404)	작업이 없거나 만료되었거나 서버 재시작으로 사라졌습니다.	작업을 처음부터 다시 만들어야 합니다.

`done` 과 `error` 응답은 한 번 읽으면 서버에서 삭제됩니다. 응답을 받은 즉시 저장하십시오. 같은 작업을 다시 조회하면 404 `unknown` 이 옵니다. 응답이 유실되었다면 같은 작업을 재조회하지 말고 작업을 처음부터 다시 만들지 여부를 판단하십시오.

모니터링이나 상태 점검에 HEAD 요청을 사용하지 마십시오. HEAD 요청도 같은 조회 처리로 들어가 결과를 소비하므로, 이후 GET 조회는 404 `unknown` 이 될 수 있습니다.

- 작업 정보는 서버 메모리에만 있으며, 마지막 상태 변경 후 1시간이 지나면 만료됩니다.
- 서버가 재시작되면 진행 중이던 작업은 모두 사라지며 조회 시 404 `unknown` 이 옵니다.
- 여러 대의 서버를 앞단 로드밸런서로 묶은 환경에서는 작업을 만든 서버와 다른 서버로 조회될 수 있습니다. 이 경우에도 404 `unknown` 이 옵니다.

작업 ID 는 결과를 조회하는 값이므로 외부에 노출하지 마십시오.

작업 생성 요청(POST)이 200 을 돌려주었다고 해서 키가 유효하다는 뜻은 아닙니다. 키가 무효하거나 권한·한도가 부족하면 작업은 만들어지고, 실패 사유는 조회 단계에서 401, 403, 413, 429 등으로 나타납니다. 연동 전에 `GET https://<CLIP 설치 주소>/api/key-status` 로 키 상태를 먼저 확인하십시오.

작업 생성 요청을 다시 보내면 새 작업이 만들어져 처리 횟수가 중복으로 소모됩니다. 재시도 규칙은 처리 시간·재시도 섹션을 따르십시오.

오류 응답

오류 응답의 본문 형식은 API 마다 다릅니다. 공통된 형식이 없으므로, 먼저 HTTP 상태 코드로 분류하고 본문에서 `code` 또는 `status` 값, 그리고 `error` 문자열을 확인하십시오. 거절한 상위 시스템의 세부 필드(부족한 권한 이름, 허용량 등)는 전달되지 않습니다.

형식	본문 예	해당 API
키 누락	`{"error":"...","code":"api_key_required"}`	인증이 필요한 모든 API

형식	본문 예	해당 API
입력 오류 (400)	`{"error":"..."}`	필드가 없거나 형식이 맞지 않을 때입니다. 메시지는 엔드포인트마다 다릅니다(각 엔드포인트의 오류 표 참고).
처리 오류 (상태 값 포함)	`{"error":"...","status":502}`	무역서류 분류·추출
처리 오류 (상태 값 없음)	`{"error":"..."}`	파일 업로드, 근거 페이지 이미지, 키 상태 확인
작업 조회 실패	`{"status":"error","error":"..."}`	비동기 작업 조회
작업 없음	`{"status":"unknown"}`	비동기 작업 조회 (HTTP 404)

처리 오류 예 (상태 값 포함) JSON

```
{"error":"처리 중 오류가 발생했습니다.","status":502}
```

상태 코드	의미	대응
400	입력 필드가 없거나 형식이 맞지 않습니다.	필드 이름과 형식을 확인하십시오. 같은 요청을 재시도하지 마십시오.
401	API 키가 없거나 잘못되었거나 비활성·만료되었습니다.	`code` 가 `api_key_required` 이면 키 헤더 누락입니다. 그 밖이면 키 값과 상태를 담당자에게 확인하십시오. 재시도하지 마십시오.
403	키 권한이 부족하거나 설정이 꺼진 기능을 요청했습니다.	필요한 권한을 담당자에게 확인하십시오. 재시도하지 마십시오.
404	작업이 없거나(`unknown`) 파일을 찾을 수 없습니다.	작업이 만료되었거나 재시작으로 사라졌는지 확인하십시오. 파일 ID 는 같은 키로 업로드한 것만 사용할 수 있습니다.
413	파일 크기, 페이지 수, 이미지 픽셀 상한 중 하나를 넘었습니다.	파일을 줄이거나 나누어 보내십시오. 상한은 담당자에게 확인하십시오. 재시도하지 마십시오.
429	분당 또는 일일 사용 한도를 넘었습니다.	60초 이상 기다린 뒤 재시도하십시오. 일일 한도는 날짜가 바뀌면 초기화되며, 기준은 한국 시간 오전 9시입니다.
5xx	서버 또는 문서 처리 엔진의 일시적인 오류입니다.	지수 백오프로 최대 2회 재시도하십시오. 이 경우 본문이 JSON 이 아닐 수 있습니다.

처리 시간·재시도

문서 1건의 판정이나 추출은 보통 수 초에서 수 분이 걸립니다. 문서 길이와 페이지 수, 스캔 여부에 따라 차이가 큽니다. 클라이언트 타임아웃은 20분 이상으로 설정하는 것을 권장합니다.

- 동시 요청은 문서 4~8건 이내로 제한하는 것을 권장합니다. 많이 보내면 한도에 빨리 도달하고 대기 시간이 길어집니다.
- 요청 1건이 내부적으로 여러 번의 처리 호출로 나뉘므로, 사용량 한도는 보낸 요청 수보다 빠르게 소모될 수 있습니다.
- 비동기 작업 조회는 3초 간격으로 하고, 전체 대기 시간에도 상한을 두어 상한을 넘으면 실패로 처리하십시오.

상황	재시도 여부	대기 방법
네트워크 오류 또는 5xx (조회 GET, 파일 업로드)	최대 2회	지수 백오프 (예: 2초, 4초)
작업 생성 POST 의 네트워크 오류	응답을 받지 못한 경우에만	5xx 를 포함해 응답을 받았다면 재시도하지 마십시오. 재시도하면 작업이 중복 생성됩니다
429	재시도합니다	60초 이상 기다린 뒤 재시도합니다. 응답에 Retry-After 헤더가 없으므로 고정 대기 시간을 사용하십시오
그 밖의 4xx	재시도하지 않습니다	요청 내용이나 키 권한을 고쳐야 합니다

작업 생성 요청(POST)을 재시도하면 새 작업이 생겨 처리 비용이 중복될 수 있습니다. 응답을 받지 못한 경우에만 재시도하고, 이미 받은 작업 ID 는 다시 생성하지 마십시오.

데이터 처리와 보관

업로드한 원본 파일과 OCR 로 추출한 텍스트는 문서 처리 엔진으로 전송되어 처리됩니다. 연동 전에 전송 경로가 고객 정책에 맞는지 보안 담당 부서와 확인하십시오.

- **기본 모드**에서는 업로드한 원본 파일이 문서 처리 엔진의 저장소에 남습니다. 파일은 API 키 단위로 격리되어 저장되며, 기간이 지나도 자동으로 삭제되지 않습니다.
- 무역서류 점검은(는) 항상 원본을 저장합니다.
- 저장된 원본을 삭제해야 하면 담당자에게 요청하십시오.

응답 JSON 과 고객 서버의 로그에도 판정 대상 문서의 내용이 들어 있을 수 있습니다. 개인정보와 기밀 문서를 다룬다면 고객 측 보관 기간과 접근 권한 정책을 함께 적용하십시오.

버전과 상태 확인

이 문서는 2026-10-28 기준입니다. API 경로에는 버전 번호가 없으며(예: `/v1` 없음), 서버 버전을 응답에서 확인할 수 없습니다. 연동 테스트 전에 담당자에게 배포된 버전을 확인하십시오.

- 응답에 새 필드가 추가될 수 있습니다. 모르는 필드는 무시하도록 파서를 작성하십시오.
- 변경 사항은 담당자를 통해 확인하십시오.

서버 동작 확인은 인증 없이 호출할 수 있습니다. 이 경로는 서버가 응답하는지만 확인하며, 문서 처리 엔진의 연결 상태나 API 키 상태는 확인하지 않습니다.

상태 확인 CURL

```
curl -s https://<CLIP 설치 주소>/api/health
```

응답은 `{"status":"ok"}` 입니다. 키 상태와 잔여 한도는 `GET https://<CLIP 설치 주소>/api/key-status` 에 `x-clip-key` 헤더를 보내 확인하십시오.

기능

무역서류 점검

무역서류 점검 API 는 선적서류와 신용장 파일을 올려 서류 종류와 핵심정보, 근거 좌표를 얻는 API 입니다. 기본 흐름은 파일을 한 번 업로드하고, 쪽 단위로 분류와 추출 작업을 시작한 뒤 결과를 조회하는 비동기 방식입니다.

> **교차 검토와 UCP 600·ISBP 821 규정 검토의 판정(통과·하자·확인 필요)은 API 로 제공되지 않습니다.** 이 API 는 문서 분류, 핵심정보, 근거 위치까지 제공합니다.

연동 순서

모든 호출에 `x-clip-key` 헤더로 API 키를 보내십시오. 여러 쪽으로 된 파일은 아래 순서를 따르는 것을 권장합니다.

1. **파일 업로드**: `POST /api/shipping/upload` 로 파일을 한 번 올리고 `fileId` 를 받습니다. 이후 처리는 이 `fileId` 로 합니다.
2. **쪽 수 확인**: 업로드 응답에는 쪽 수가 없습니다. PDF 라면 원본 파일에서 쪽 수를 읽는 것을 권장합니다. `POST /api/shipping/classify` 의 `pages` 는 파일 전체를 동기로 처리한 뒤에 나오므로, 쪽 수 확인만을 위해 호출하면 시간과 비용이 큽니다.
3. **쪽별 분류**: 1쪽부터 마지막 쪽까지 `POST /api/shipping/process` 를 `mode` 를 `classify` 로 보내 `jobId` 를 받고, `GET /api/shipping/process?job=` 을 `done` 이 될 때까지 3초 간격으로 조회합니다. 쪽마다 서류 종류 `type` 이 나옵니다. 쪽마다 처리 요청이 따로 나가므로 쪽 수에 비례해 시간과 비용이 늘어납니다. 쪽 번호는 파일의 쪽 수 안에서만 지정하십시오.
4. **서류 묶기**: 같은 `type` 이 연속된 쪽은 한 서류로 묶습니다. 한 파일에 여러 서류가 들어 있을 수 있습니다. 서류 종류 이름은 고정 목록이 아니므로, 묶기와 분기는 `document` · `unknown` 같은 미분류 값을 기준으로 처리하십시오.
5. **핵심정보 추출**: 쪽마다 `mode` 를 `full` 로 같은 방식으로 처리합니다. `keys` 에 항목 이름, 값, 근거 좌표 번호가 나옵니다.
6. **교차 검토**: 서류 사이에서 값을 비교하는 검토는 API 로 제공되지 않습니다. 필요한 비교는 연동 측에서 구현하십시오. 자세한 내용은 「API 로 제공되지 않는 판정」을 보십시오.

한 파일에 여러 서류가 섞여 있고 쪽별 호출을 줄이고 싶다면 `POST /api/shipping/review` 로 서류별 범위와 핵심 정보를 한 번에 받을 수 있습니다. 다만 이 경로는 clip 웹 화면이 쓰지 않으므로, 서류 경계가 화면 결과와 다를 수 있습니다. 실제 샘플로 먼저 확인하십시오. 서류가 한 종류뿐인 파일은 `POST /api/shipping/classify` 또는 `POST /api/shipping/extract` 로 한 번에 처리할 수 있습니다.

결과 읽는 법

- **type**: 서류 종류 이름입니다. 이름은 분류 결과 문자열이며 고정된 목록이 아닙니다. 특정 서류가 항상 같은 이름으로 나온다고 가정하지 마십시오. 분기는 `document` · `unknown` 같은 미분류 값만 기준으로 하고, 나머지는 표시용으로 쓰십시오.
- **confidence**: classify·extract 응답의 서류 신뢰도는 현재 항상 0 으로 돌아오므로 분류 판단에 쓰지 마십시오. process 의 `keys[].confidence` 는 확률이 아니라 처리 엔진이 매긴 등급 점수입니다. 값이 낮은 항목은 사람이 다시 확인할 후보로 참고하십시오.
- **key, value**: 항목 이름과 값 문자열입니다. 값은 원문 표기 그대로이므로 금액이나 날짜의 형식은 연동 측에서 정규화하십시오.
- **informationList, processed**: 여러 줄 항목과 구조화된 값(금액·과부족 등)입니다. 해당하지 않으면 생략됩니다.
- **reference, words**: `reference` 는 값이 적힌 쪽 번호와 단어 번호 목록입니다. 좌표 `[x0, y0, x1, y1]` 은 보통 쪽 크기 대비 0~1 비율입니다. 찾는 방법은 두 형식이 다릅니다. `process` 의 `words` 는 평면 맵이므로 `words[단어 번호]` 로 찾습니다. `review` 의 `words` 는 2단계 맵이므로 `words[쪽 번호][단어 번호]` 로 찾습니다.
- **pageStart, pageEnd**: `review` 결과에서 서류가 차지하는 쪽 범위입니다. 양 끝 쪽을 포함합니다. 시작 쪽이 같은 두 서류가 있으면 쪽이 겹칠 수 있습니다.

API 로 제공되지 않는 판정

> **아래 판정은 API 응답에 포함되지 않습니다(API 로는 제공되지 않음).** clip 웹 화면은 이 API 가 준 서류 종류·키·값으로 브라우저에서 판정을 계산하며, API 는 그 입력 재료만 제공합니다.

- **교차 검토**: 서류 사이의 값 일치 여부를 보는 점검 항목의 통과·하자·확인 필요 판정
- **규정 검토**: UCP 600 과 ISBP 821 조항별 판정(통과·하자·확인 필요·해당 없음)
- **조항 원문과 번역문**: 규정 조항의 원문과 번역은 API 로 제공되지 않습니다.

이 판정이 필요하면 `process` , `extract` , `review` 로 얻은 값을 가지고 연동 측 로직으로 구현하십시오. 판정 기준은 UCP 600 과 ISBP 821 원문을 따르십시오.

제한과 주의사항

- **파일 보관**: 업로드한 파일은 서버 파일 저장소에 보관되며, 이 API 에는 삭제 기능이 없습니다. 이 API 는 파일을 저장하지 않는 옵션(저장 없는 모드)을 제공하지 않습니다.
- **파일 크기와 쪽 수**: 파일 크기 한도는 서버 설정에 따릅니다. 한도를 넘으면 413 이 나옵니다. 현재 설정은 `GET /api/shipping/dashboard` 로 확인하십시오.
- **결과 1회 조회**: `jobId` 의 `done` 또는 `error` 결과는 한 번 받으면 서버에서 삭제됩니다. 같은 `jobId` 를 다시 조회하면 404 와 `unknown` 이 나옵니다. 받은 결과는 연동 측에서 저장하십시오.
- **작업 보관 기간**: `jobId` 는 서버 메모리에 보관됩니다. 오래된 작업의 정리는 새 POST 요청이 들어올 때 이루어지므로, 새 요청이 없으면 1시간이 지나도 조회될 수 있습니다. 조회 가능 기간은 보장되지 않으니 결과는 받는 즉시 가져가십시오. 서버가 재시작되면 진행 중인 작업은 사라집니다.
- **동기 호출**: `classify` 와 `extract` 는 파일 전체를 한 번에 처리하고 응답을 기다립니다. 긴 파일은 응답까지 수 분이 걸릴 수 있습니다. 여러 서류가 섞인 파일에는 `review` 또는 쪽별 `process` 를 사용하십시오.
- **조회와 jobId**: 결과 조회 호출은 키를 검사하지 않습니다. 그래도 `jobId` 를 알면 결과를 받을 수 있으니 `jobId` 를 외부에 노출하지 마십시오.

화면 작업과 API

아래 표는 웹 화면에서 하는 작업마다 어떤 호출이 나가는지 정리한 것입니다. 연동 개발에는 `upload` 와 `process` (POST·GET) 만 필요합니다. 표의 **화면 전용** 호출은 화면이 보조 판정에 쓰는 내부 호출이므로 연동에 사용하지 마십시오.

화면에서 하는 일	호출하는 API	설명
파일 선택 또는 끌어다 놓기 (자동 심사 켜짐)	POST `https://<CLIP 설치 주소>/api/shipping/upload` (`file`), 이어서 쪽마다 POST·GET `https://<CLIP 설치 주소>/api/shipping/process` (`mode` 는 `classify` 한 번, `full` 한 번)	파일마다 한 번 업로드한 뒤 쪽마다 서류 종류를 먼저 확인하고, 핵심정보를 추출합니다. 쪽 수는 브라우저에서 세고, 결과로 받은 쪽들을 서류 단위로 묶습니다.
파일 추가 (자동 심사 꺼짐)	없음 (화면에서 처리)	파일을 대기열에 모아 둡니다. 이 단계에서는 서버를 호출하지 않습니다.
심사 시작 (대기열 처리)	POST `https://<CLIP 설치 주소>/api/shipping/upload` , POST·GET `https://<CLIP 설치 주소>/api/shipping/process` (위 첫 행과 같음)	대기열의 파일을 위 첫 행과 같은 순서로 처리합니다.
대기열에서 파일 제거, 오류 메시지 닫기	없음 (화면에서 처리)	화면 목록에서만 지웁니다. 이미 올라간 파일은 서버에 남습니다.

화면에서 하는 일	호출하는 API	설명
API 키 입력	없음 (화면에서 처리)	입력한 키는 이 브라우저에 저장되고, 이후 화면의 호출에 `x-clip-key` 헤더로 붙습니다. 키 입력 자체는 서버를 호출하지 않습니다.
API 키 입력 후 자동 재처리	위 첫 행과 같은 호출을 다시 보냄	키가 없어 401 을 받은 파일은 키를 입력하면 업로드와 처리 호출을 다시 보냅니다.
새 심사 시작	없음 (화면에서 처리)	화면의 결과와 입력값을 지웁니다. 서버에 올라간 파일과 결과는 지우지 않습니다.
서류 묶기, 서류 종류 표시, 근거 보기	없음 (화면에서 처리)	쪽 결과의 `type` 이 같고 연속인 쪽을 한 서류로 묶습니다. 근거 박스는 `words` 좌표로 그립니다.
서류 본문 문구 확인 (자동)	화면 전용 (공개 API 아님)	부기·약관 문구가 서류 본문에 있는지 확인합니다.
정정 흔적 확인 (자동, 서류 이미지)	화면 전용 (공개 API 아님)	서류 이미지에서 고친 흔적과 인증 여부를 확인합니다.
정정 흔적 2차 확인 (자동, 무효 문구가 있을 때)	화면 전용 (공개 API 아님)	무효 문구가 있는 서류에서 서명과 부서명이 있는지 한 번 더 확인합니다.
비서류적 조건 충돌 검사 (자동, 신용장이 있을 때)	화면 전용 (공개 API 아님)	신용장의 조건과 서류 기재가 명백히 어긋나는지 확인합니다.
요구 서류 충족 확인 (자동, 문서조건 항목)	화면 전용 (공개 API 아님)	신용장의 요구 서류 항목이 제출 서류로 충족되는지 확인합니다. 항목이나 서류 값이 바뀌면 잠시 뒤 다시 확인하며, 사람이 직접 입력한 항목은 덮어쓰지 않습니다.
기재값 표기 차이 확인 (자동)	화면 전용 (공개 API 아님)	값이 다를 때 같은 대상을 가리키는지 확인합니다.
점검 항목의 ✓ 수동 지정, 판정 수동 변경	없음 (화면에서 처리)	자동 판정을 사람이 고칩니다. 화면에서만 반영됩니다.
서류 제시일, 제29조 연장 날짜 입력	없음 (화면에서 처리)	입력한 날짜는 화면의 판정 계산에 쓰입니다.
결과 탭, 필터, 카드 순서 바꾸기, 전체 펼치기	없음 (화면에서 처리)	화면 보기 설정입니다.

화면에서 하는 일	호출하는 API	설명
결과 다운로드	없음 (화면에서 처리)	점검 결과와 추출 정보를 CSV 파일로 브라우저에서 만듭니다.
문구 복사	없음 (화면에서 처리)	화면의 문구를 클립보드에 복사합니다.
통과, 하자, 확인 필요 판정	없음 (화면에서 처리)	판정은 브라우저가 점검 기준과 API 결과로 계산합니다. 판정 결과는 API 응답에 없습니다.

적용 예시

한 회사가 보낸 선적서류 파일들을 폴더째로 넘겨, 파일마다 서류 종류와 핵심정보를 모으는 예시입니다. 판정(통과·하자)은 하지 않고, 서류 종류와 핵심정보만 모읍니다. 업로드한 파일은 서버에 보관되므로 이 점을 고려하십시오.

1. 폴더의 파일을 이름순으로 고릅니다. PDF, PNG, JPG, TIFF, WEBP 형식을 대상으로 합니다.
2. 파일 하나를 `POST https://<CLIP 설치 주소>/api/shipping/upload` 로 올리고 `fileId` 를 받습니다. 업로드는 다시 보내지 않습니다.
3. 쪽 수를 셉니다. API 는 쪽 수를 알려 주지 않으므로, PDF 는 `pypdf` , TIFF 는 `Pillow` 로 쪽 수를 세고 PNG·JPG·WEBP 는 1쪽으로 봅니다. 패키지가 없으면 경고를 출력하고 그 파일을 1쪽으로 처리하므로, 실행 전에 `pip install requests pypdf pillow` 로 설치하십시오.
4. 1쪽부터 마지막 쪽까지 `POST https://<CLIP 설치 주소>/api/shipping/process` 를 `mode` 를 `full` 로 보내 `jobId` 를 받습니다. 한 번에 한 쪽씩 보냅니다. 쪽 처리가 실패하면 그 쪽을 `errors` 에 기록하고 나머지 쪽은 계속 처리합니다.
5. `GET https://<CLIP 설치 주소>/api/shipping/process?job=` 을 3초 간격으로 조회합니다. `status` 가 `pending` 이면 계속 기다리고, `done` 이면 `result` 를 받습니다. `error` 또는 `unknown` 이면 그 쪽을 오류로 기록합니다. 결과는 한 번 받으면 서버에서 지워지므로 받은 값은 바로 모아 두십시오. 연결 오류는 최대 2번 다시 조회하며, 5xx 응답은 본문의 `status` 로 판단하고 다시 보내지 않습니다.
6. 쪽의 `type` 이 같고 연속인 쪽을 한 서류로 묶고, 서류별 핵심정보는 처음 나온 값을 씁니다. 요약을 출력하고, 파일 하나를 끝낼 때마다 `results.json` 에 저장합니다.

review_folder.py PYTHON

```
import json
import sys
import time
from pathlib import Path
```

```
import requests

BASE_URL = "https://<CLIP 설치 주소>".rstrip("/")
API_KEY = "YOUR_API_KEY"
HEADERS = {"x-clip-key": API_KEY}
EXTENSIONS = {".pdf", ".png", ".jpg", ".jpeg", ".tif", ".tiff", ".webp"}
CONNECT_RETRIES = 2  # 연결 오류(응답을 받지 못함)일 때만 다시 보내는 횟수
POLL_SECONDS = 3     # 결과 조회 간격
MAX_POLLS = 600      # 결과 조회 최대 횟수(약 30분)
RESULTS_FILE = Path("results.json")


def call(method, path, timeout=60, retries=0, **kwargs):
    """연결 오류일 때만 retries 번 다시 보냅니다. 5xx 응답은 다시 보내지 않고 그대로 돌려줍니다."""
    for attempt in range(retries + 1):
        try:
            return requests.request(method, BASE_URL + path, headers=HEADERS, timeout=timeout, **kwargs)
        except requests.RequestException as e:
            if attempt == retries:
                raise RuntimeError(f"{method} {path} 연결 실패: {e}")
            time.sleep(2 ** attempt)


def upload(path):
    data = path.read_bytes()
    # 업로드는 다시 보내지 않습니다. 같은 파일이 두 번 올라갈 수 있기 때문입니다.
    res = call("POST", "/api/shipping/upload", timeout=300, files={"file": (path.name, data)})
    if res.status_code != 200:
        raise RuntimeError(f"업로드 실패 {res.status_code}: {res.text[:200]}")
    file_id = res.json().get("fileId")
    if not file_id:
        raise RuntimeError("업로드 응답에 fileId 가 없습니다.")
    return file_id


def count_pages(path):
    suffix = path.suffix.lower()
    try:
        if suffix == ".pdf":
            from pypdf import PdfReader
            return len(PdfReader(str(path)).pages)
        if suffix in {".tif", ".tiff"}:
            from PIL import Image
            with Image.open(path) as im:
                return getattr(im, "n_frames", 1)
    except ImportError:
        print(f"  [경고] {path.name}: 쪽 수를 세려면 PDF 는 pypdf, TIFF 는 Pillow 가 필요합니다. pip install pypdf pillow 를 실행하십시오. 이 파일은 1쪽으로 처리합니다.")
    return 1
```

```
def process_page(file_id, page):
    res = call("POST", "/api/shipping/process", json={"fileId": file_id, "page": page, "mode": "full"})
    if res.status_code != 200:
        raise RuntimeError(f"처리 요청 실패 {res.status_code}: {res.text[:200]}")
    job_id = res.json()["jobId"]
    for _ in range(MAX_POLLS):
        time.sleep(POLL_SECONDS)
        res = call("GET", "/api/shipping/process", timeout=60, retries=CONNECT_RETRIES, params={"job": job_id})
        try:
            body = res.json()
        except ValueError:
            raise RuntimeError(f"조회 응답을 읽지 못했습니다(HTTP {res.status_code}).")
        # 실패한 작업은 HTTP 상태가 5xx 여도 본문의 status 로 판단합니다.
        status = body.get("status")
        if status == "pending":
            continue
        if status == "done":
            return body["result"]
        if status == "error":
            raise RuntimeError(body.get("error", "알 수 없는 오류"))
        raise RuntimeError(f"작업 상태를 알 수 없습니다({status}). 이미 받은 결과이거나 서버가 재시작되었을 수 있습니다.")
    raise TimeoutError("결과가 제한 시간 안에 오지 않았습니다.")


def group_documents(pages):
    """쪽의 type 이 같고 연속인 쪽을 한 서류로 묶습니다."""
    docs = []
    for p in pages:
        last = docs[-1] if docs else None
        if last and last["documentType"] == p["type"] and last["pageEnd"] == p["page"] - 1:
            last["pageEnd"] = p["page"]
            last["_pages"].append(p)
        else:
            docs.append({"documentType": p["type"], "pageStart": p["page"], "pageEnd": p["page"], "_pages": [p]})
    result = []
    for d in docs:
        values = {}
        for p in d["_pages"]:
            for item in p.get("keys", []):
                if item.get("value") and item["key"] not in values:
                    values[item["key"]] = item["value"]
        result.append({"documentType": d["documentType"], "pageStart": d["pageStart"],
                       "pageEnd": d["pageEnd"], "keyValues": values})
    return result


def process_file(path):
    file_id = upload(path)
```

```
    page_count = count_pages(path)
    pages, errors = [], []
    for page in range(1, page_count + 1):
        try:
            pages.append(process_page(file_id, page))
        except Exception as e:
            errors.append({"page": page, "error": str(e)})
    return {
        "fileName": path.name,
        "fileId": file_id,
        "pageCount": page_count,
        "documents": group_documents(pages),
        "errors": errors,
    }


def main():
    if len(sys.argv) != 2:
        sys.exit("사용법: python review_folder.py <파일 폴더>")
    folder = Path(sys.argv[1])
    files = sorted(p for p in folder.iterdir() if p.is_file() and p.suffix.lower() in EXTENSIONS)

    summary = []
    for path in files:
        print(f"[처리] {path.name}")
        try:
            summary.append(process_file(path))
        except Exception as e:
            summary.append({"fileName": path.name, "error": str(e)})
        # 파일 하나를 끝낼 때마다 저장합니다. 중간에 멈춰도 끝난 파일의 결과는 남습니다.
        RESULTS_FILE.write_text(json.dumps(summary, ensure_ascii=False, indent=2), encoding="utf-8")

    for item in summary:
        print()
        print(f"■ {item['fileName']}")
        if "error" in item:
            print(f"  실패: {item['error']}")
            continue
        for doc in item["documents"]:
            print(f"  - {doc['documentType']} (p.{doc['pageStart']}-{doc['pageEnd']})")
            for key, value in list(doc["keyValues"].items())[:5]:
                print(f"      {key}: {value}")
        for err in item["errors"]:
            print(f"  쪽 {err['page']} 실패: {err['error']}")

    print()
    print(f"결과를 {RESULTS_FILE} 에 저장했습니다.")
```

```
if __name__ == "__main__":
    main()
```

실행하면 아래와 같이 파일별 서류 종류와 핵심정보가 출력되고, 전체 결과가 `results.json` 에 저장됩니다. 서류 종류 이름과 값은 파일에 따라 달라지며, 아래는 예시입니다.

TEXT

```
$ python review_folder.py ./shipping-docs
[처리] 001-invoice.pdf
[처리] 002-bl.pdf

■ 001-invoice.pdf
 - Commercial Invoice (p.1-2)
     Invoice No.: INV-2026-0412
     Amount: USD 48,200.00

■ 002-bl.pdf
 - Bill of Lading (p.1-1)
     B/L No.: HDMU0412xxxx

결과를 results.json 에 저장했습니다.
```

엔드포인트

POST /api/shipping/upload

파일 업로드

서류 파일을 서버에 올리고 이후 처리에 쓸 fileId 를 받습니다.

파일은 한 번만 올리면 됩니다. 받은 `fileId` 로 쪽별 처리(`process`)를 여러 번 호출할 수 있습니다.

처리 시간 · 파일 크기에 따라 수 초에서 수 분이 걸릴 수 있습니다. 파일 전달 단계는 5분이 지나면 서버에서 중단되므로 그 안에 끝나는 크기여야 합니다. 클라이언트 타임아웃은 넉넉히(10분 이상) 잡으십시오.

요청 · MULTIPART/FORM-DATA

이름	위치	타입	필수	설명
x-clip-key	header	string	필수	API 키. 발급받은 키를 넣으십시오. 예: YOUR_API_KEY
file	form	file	필수	서류 파일. PDF, PNG, JPG, TIFF, WEBP 를 권장합니다. 파일 크기 한도는 서버 설정에 따릅니다.

응답 · 200

업로드된 파일의 식별자입니다. fileName, size 는 서버가 돌려주는 경우에만 포함됩니다.

필드	타입	설명
fileId *	string	업로드된 파일 식별자. process 호출의 fileId 로 씁니다.
fileName	string	업로드된 파일 이름
size	integer	파일 크기(바이트)

응답 예 JSON

```
{
  "fileId": "f_8c21a7e4",
  "fileName": "lc-and-invoice.pdf",
  "size": 482913
}
```

오류

상태	언제	본문 예
401	x-clip-key 헤더가 없거나 비어 있을 때. 본문의 문구는 화면 안내용이므로, 판별에는 code 값(api_key_required)을 쓰십시오.	{"error":"우측 상단에 API Key 를 입력해 주세요.","code":"api_key_required"}
400	file 필드가 없거나 파일이 아닐 때	{"error":"file is required"}
413	파일 크기가 서버 설정의 한도를 넘을 때. 한도는 서버 설정에 따릅니다.	{"error":"Clip API error 413: File size exceeds limit"}
401	처리 엔진이 키를 거부할 때. 상태 코드와 본문은 처리 엔진의 응답을 따릅니다.	{"error":"Unauthorized"}
500	파일 저장소가 오류를 돌려줄 때(상태 코드는 저장소 응답을 그대로 따릅니다). 본문에는 저장소의 오류 메시지가 있으면 그대로 들어가고, 없으면 이 형태가 나옵니다.	{"error":"upload 500"}
200	저장소가 성공 응답을 했지만 fileId 를 주지 않았을 때. HTTP 상태는 200 이지만 fileId 가 없으므로 실패로 처리하십시오.	{"error":"upload 200"}
502	파일 저장소에 연결하지 못했을 때, 또는 저장소 전달이 5분을 넘어 중단되었을 때	{"error":"The operation was aborted due to timeout"}

참고

업로드한 파일은 서버 파일 저장소에 보관됩니다. 이 API 에는 파일 삭제 기능이 없으며, 이 API 는 파일을 저장하지 않는 옵션(저장 없는 모드)을 제공하지 않습니다.

예제

cURL

```curl
curl -X POST "https://<CLIP 설치 주소>/api/shipping/upload" \
  -H "x-clip-key: YOUR_API_KEY" \
  -F "file=@./lc-and-invoice.pdf"
```

Python (requests)

```python
import requests

with open("lc-and-invoice.pdf", "rb") as f:
    r = requests.post(
        "https://<CLIP 설치 주소>/api/shipping/upload",
        headers={"x-clip-key": "YOUR_API_KEY"},
        files={"file": ("lc-and-invoice.pdf", f, "application/pdf")},
        timeout=600,
    )
r.raise_for_status()
print(r.json()["fileId"])
```

JavaScript (fetch)

```javascript
import { readFile } from 'node:fs/promises';

const buf = await readFile('./lc-and-invoice.pdf');
const form = new FormData();
form.append('file', new Blob([buf], { type: 'application/pdf' }), 'lc-and-invoice.pdf');

const res = await fetch('https://<CLIP 설치 주소>/api/shipping/upload', {
  method: 'POST',
  headers: { 'x-clip-key': 'YOUR_API_KEY' },
  body: form,
});
const { fileId } = await res.json();
console.log(fileId);
```

POST /api/shipping/process

쪽 처리 시작

업로드한 파일의 한 쪽에 대해 분류 또는 핵심정보 추출 작업을 시작하고 jobId 를 받습니다.

`mode` 가 `classify` 면 서류 종류만, `full` 이면 서류 종류와 핵심정보를 얻습니다. 작업은 비동기이므로 결과는 `GET /api/shipping/process?job=` 으로 받습니다.

POST 는 키 유무와 `fileId` · `page` 필수 값만 확인합니다. 쪽 번호가 파일의 쪽 수 안에 있는지는 확인하지 않으므로, 범위 밖 쪽은 지정하지 마십시오. 처리 중의 오류는 POST 가 아니라 GET 조회의 `error` 상태로 나옵니다.

처리 시간 · 요청 자체는 즉시 응답합니다. 실제 처리 시간은 GET 조회로 기다립니다.

요청 · APPLICATION/JSON

이름	위치	타입	필수	설명
x-clip-key	header	string	필수	API 키. 발급받은 키를 넣으십시오. 예: YOUR_API_KEY
fileId	json	string	필수	업로드 응답의 fileId. 업로드할 때 쓴 키와 같은 키로 호출하십시오. 예: f_8c21a7e4
page	json	integer	필수	처리할 쪽 번호(1부터 시작). 파일의 쪽 수 안의 번호만 지정하십시오. 예: 1
mode	json	string	선택	`classify` (서류 종류만) 또는 `full` (종류와 핵심정보) 기본값: full

응답 · 200

작업 식별자. 결과가 아니라 작업 번호입니다.

필드	타입	설명
jobId *	string	작업 식별자(UUID). GET 조회에 씁니다.

응답 예 JSON

```
{
  "jobId": "3f2b9c1e-6a4d-4e8b-9f0a-2d7c5b1e8a40"
}
```

오류

상태	언제	본문 예
401	x-clip-key 헤더가 없거나 비어 있을 때. 본문의 문구는 화면 안내용이므로, 판별에는 code 값(api_key_required)을 쓰십시오.	{"error":"우측 상단에 API Key 를 입력해 주세요.","code":"api_key_required"}
400	fileId 가 없거나 page 가 1 미만일 때	{"error":"fileId and page are required"}

예제

cURL CURL

```
curl -X POST "https://<CLIP 설치 주소>/api/shipping/process" \
  -H "x-clip-key: YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"fileId":"f_8c21a7e4","page":1,"mode":"classify"}'
```

Python (requests) PYTHON

```
import requests

r = requests.post(
    "https://<CLIP 설치 주소>/api/shipping/process",
    headers={"x-clip-key": "YOUR_API_KEY"},
    json={"fileId": "f_8c21a7e4", "page": 1, "mode": "classify"},
    timeout=60,
)
r.raise_for_status()
print(r.json()["jobId"])
```

JavaScript (fetch) JAVASCRIPT

```
const res = await fetch('https://<CLIP 설치 주소>/api/shipping/process', {
  method: 'POST',
  headers: { 'x-clip-key': 'YOUR_API_KEY', 'Content-Type': 'application/json' },
  body: JSON.stringify({ fileId: 'f_8c21a7e4', page: 1, mode: 'classify' }),
});
const { jobId } = await res.json();
console.log(jobId);
```

GET /api/shipping/process

쪽 처리 결과 조회

jobId 의 상태를 조회합니다. 처리가 끝나면 쪽의 서류 종류와 핵심정보를 돌려줍니다.

`status` 가 `pending` 이면 아직 처리 중입니다. 3초 간격으로 다시 조회하십시오. `done` 이면 `result` 를 받습니다. 결과(오류 결과 포함)는 한 번 받으면 서버에서 삭제됩니다.

처리 시간 · 처리 시간은 파일 크기와 쪽 수에 따라 수 초에서 수 분입니다. 클라이언트 타임아웃은 20분 이상 권장합니다.

요청

이름	위치	타입	필수	설명
x-clip-key	header	string	선택	이 조회는 키를 검사하지 않습니다. 같은 키로 보내는 것을 권장합니다. 예: YOUR_API_KEY
job	query	string	필수	POST 응답의 jobId 예: 3f2b9c1e-6a4d-4e8b-9f0a-2d7c5b1e8a40

응답 · 200

`status` 가 `pending` , `done` , `error` , `unknown` 중 하나입니다. `done` 일 때만 `result` 가 있고, `error` 일 때만 `error` 메시지가 있습니다.

필드	타입	설명
status *	"pending" | "done" | "error" | "unknown"	작업 상태
result	object	status 가 done 일 때의 쪽 처리 결과
result.page *	integer	처리한 쪽 번호
result.type *	string	서류 종류 이름(분류 결과). 분류하지 못하면 document 입니다. 값은 고정 목록이 아닙니다.
result.keys *	object[]	추출된 핵심정보 목록. classify 모드에서는 비어 있습니다.
result.keys[].key *	string	항목 이름(예: L/C No., Amount). 서류 종류마다 다릅니다.
result.keys[].value *	string	추출된 값 문자열. 값을 찾지 못하면 빈 문자열입니다.
result.keys[].informationList	string[]	여러 줄로 된 항목(예: 서류 목록, 조건 목록)의 줄 단위 값. 해당하지 않으면 생략됩니다.
result.keys[].processed	object	금액·과부족처럼 구조화된 값(예: 금액은 currency·amount, 과부족은 plus·minus). 해당하지 않으면 생략됩니다.
result.keys[].reference *	{ [키]: integer[] }	값이 적힌 쪽 번호(문자열 키)와 단어 번호 목록. 번호로 words 에서 좌표를 찾습니다.
result.keys[].confidence	number	항목의 등급 점수(0~1). 확률이 아니라 처리 엔진이 매긴 등급 점수입니다. 값이 낮은 항목은 사람이 다시 확인할 후보로 참고하십시오. 값이 없는 응답도 있으니 없으면 표시를 생략하십시오.
result.words *	{ [키]: number[] }	근거 단어의 좌표. 키 이름이 단어 번호(문자열)이고, 값이 좌표입니다(쪽 번호 단계가 없는 평면 맵). keys 의 reference 가 가리키는 단어만 들어 있습니다.
error	string	status 가 error 일 때의 오류 메시지

응답 예 JSON


```
{
  "status": "done",
  "result": {
    "page": 1,
    "type": "letter of credit",
    "keys": [
      {
        "key": "L/C No.",
        "value": "LC2026-0042",
        "reference": {
          "1": [
            12
          ]
        },
        "confidence": 0.97
      },
      {
        "key": "Amount",
        "value": "USD 125,000.00",
        "processed": {
          "currency": "USD",
          "amount": 125000
        },
        "reference": {
          "1": [
            18,
            19
          ]
        },
        "confidence": 0.95
      }
    ],
    "words": {
      "12": [
        0.11,
        0.08,
        0.31,
        0.1
      ],
      "18": [
        0.2,
        0.42,
        0.3,
        0.44
      ],
      "19": [
        0.32,
```

```
        0.42,
        0.42,
        0.44
      ]
    }
  }
}
```

오류

상태	언제	본문 예
404	jobId 를 찾을 수 없을 때. 잘못된 값, 이미 한 번 받은 값, 서버 재시작으로 사라진 값, 정리된 값이 해당합니다. 본문의 status 가 unknown 입니다.	{"status":"unknown"}
401	처리 엔진이 키를 거부할 때. 상태 코드는 처리 엔진의 응답을 따르며, 본문의 status 는 error 입니다.	{"status":"error","error":"process 401"}
404	처리 엔진이 파일이나 쪽을 찾지 못했을 때. 본문의 status 가 error 이므로, unknown(작업 없음)과 구분하십시오.	{"status":"error","error":"process 404"}
413	파일 크기가 서버 설정의 한도를 넘을 때. 한도는 서버 설정에 따릅니다.	{"status":"error","error":"File size exceeds limit"}
429	요청이 너무 많을 때	{"status":"error","error":"process 429"}
502	처리 엔진에 연결하지 못했을 때, 또는 상태를 알 수 없을 때	{"status":"error","error":"fetch failed"}

예제

cURL CURL

```
curl "https://<CLIP 설치 주소>/api/shipping/process?job=3f2b9c1e-6a4d-4e8b-9f0a-2d7c5b1e8a40" \
  -H "x-clip-key: YOUR_API_KEY"
```

Python (requests) PYTHON

```python
import time
import requests

headers = {"x-clip-key": "YOUR_API_KEY"}
job = "3f2b9c1e-6a4d-4e8b-9f0a-2d7c5b1e8a40"

while True:
    r = requests.get("https://<CLIP 설치 주소>/api/shipping/process", params={"job": job}, headers=headers, timeo
ut=30)
    body = r.json()
    if body["status"] == "done":
        print(body["result"])
        break
    if body["status"] != "pending":
        raise RuntimeError(body.get("error", body["status"]))
    time.sleep(3)
```

JavaScript (fetch) JAVASCRIPT

```javascript
const headers = { 'x-clip-key': 'YOUR_API_KEY' };
const job = '3f2b9c1e-6a4d-4e8b-9f0a-2d7c5b1e8a40';

let body;
for (;;) {
  const res = await fetch('https://<CLIP 설치 주소>/api/shipping/process?job=' + job, { headers });
  body = await res.json();
  if (body.status !== 'pending') break;
  await new Promise((r) => setTimeout(r, 3000));
}
if (body.status !== 'done') throw new Error(body.error ?? body.status);
console.log(body.result);
```

POST /api/shipping/classify

파일 서류 종류 분류

파일 1건을 한 번에 처리해 서류 종류와 쪽 수를 돌려줍니다. 응답까지 기다리는 동기 호출입니다.

핵심정보는 추출하지 않습니다. 파일에 서류가 한 종류만 들어 있을 때 쓰십시오. 여러 서류가 섞인 파일은 `review` 를 사용하십시오. 파일 전체를 동기로 처리하므로 쪽 수가 많으면 시간과 비용이 큽니다. 쪽 수 확인만을 목적으로 호출하지 마십시오.

처리 시간 · 처리 시간은 파일 크기와 쪽 수에 따라 수 초에서 수 분입니다. 클라이언트 타임아웃은 20분 이상 권장합니다. 서버 내부에서 처리 엔진 호출이 실패하면 재시도가 최대 3회까지 이루어지므로, 실패 응답이 늦게 올 수 있고 OCR 이 중복 실행될 수 있습니다.

요청 · MULTIPART/FORM-DATA

이름	위치	타입	필수	설명
x-clip-key	header	string	필수	API 키. 발급받은 키를 넣으십시오. 예: YOUR_API_KEY
file	form	file	필수	서류 파일. 파일 크기 한도는 서버 설정에 따릅니다.

응답 · 200

파일의 대표 서류 종류와 신뢰도, 쪽 수입니다.

필드	타입	설명
documentType *	string	서류 종류 이름. 분류하지 못하면 unknown 또는 document 가 올 수 있습니다. 값은 고정 목록이 아닙니다.
confidence *	number	서류 분류 신뢰도(0~1). 현재는 값이 제공되지 않아 항상 0 으로 돌아오므로 판단에 쓰지 마십시오.
pages *	integer	파일의 쪽 수

응답 예 JSON


```
{
  "documentType": "commercial invoice",
  "confidence": 0,
  "pages": 3
}
```


오류

상태	언제	본문 예
401	x-clip-key 헤더가 없거나 비어 있을 때. 본문의 문구는 화면 안내용이므로, 판별에는 code 값(api_key_required)을 쓰십시오.	{"error":"우측 상단에 API Key 를 입력해 주세요.","code":"api_key_required"}
400	file 필드가 없거나 파일이 아닐 때	{"error":"file is required"}
401	키가 잘못되었을 때. 처리 엔진의 응답 상태를 그대로 전달합니다.	{"error":"Clip API error 401: Unauthorized","status":401}
403	키에 이 작업 권한이 없을 때. 처리 엔진의 응답 상태를 그대로 전달합니다.	{"error":"Clip API error 403: Forbidden","status":403}

상태	언제	본문 예
404	처리 엔진이 대상을 찾지 못했을 때. 처리 엔진의 응답 상태를 그대로 전달합니다.	{"error":"Clip API error 404: Not Found","status":404}
413	파일 크기가 서버 설정의 한도를 넘을 때. 한도는 서버 설정에 따르며, 현재 한도는 dashboard 에서 확인할 수 있습니다.	{"error":"Clip API error 413: File size exceeds limit","status":413}
429	요청이 너무 많을 때. 잠시 뒤 다시 보내십시오.	{"error":"Clip API error 429: Too Many Requests","status":429}
500	처리 엔진이 오류를 돌려줄 때(상태 코드는 그대로 전달됩니다)	{"error":"Clip API error 500: Internal Server Error","status":500}
502	처리 엔진에 연결하지 못했을 때, 또는 처리 엔진의 응답 형식이 맞지 않을 때	{"error":"Network error: fetch failed","status":502}
502	응답 형식이 맞지 않을 때(메시지에 Invalid OCR response shape 가 붙습니다)	{"error":"Invalid OCR response shape: <상세 메시지>","status":502}

예제

cURL CURL

```
curl -X POST "https://<CLIP 설치 주소>/api/shipping/classify" \
  -H "x-clip-key: YOUR_API_KEY" \
  -F "file=@./commercial-invoice.pdf"
```

Python (requests) PYTHON

```
import requests

with open("commercial-invoice.pdf", "rb") as f:
    r = requests.post(
        "https://<CLIP 설치 주소>/api/shipping/classify",
        headers={"x-clip-key": "YOUR_API_KEY"},
        files={"file": ("commercial-invoice.pdf", f, "application/pdf")},
        timeout=1200,
    )
r.raise_for_status()
print(r.json())
```

JavaScript (fetch) JAVASCRIPT

```
import { readFile } from 'node:fs/promises';

const buf = await readFile('./commercial-invoice.pdf');
const form = new FormData();
form.append('file', new Blob([buf], { type: 'application/pdf' }), 'commercial-invoice.pdf');

const res = await fetch('https://<CLIP 설치 주소>/api/shipping/classify', {
  method: 'POST',
  headers: { 'x-clip-key': 'YOUR_API_KEY' },
  body: form,
});
console.log(await res.json());
```

POST /api/shipping/extract

파일 핵심정보 추출

파일 1건을 한 번에 처리해 서류 종류와 핵심정보를 평면 목록으로 돌려줍니다. 동기 호출입니다.

`data` 는 항목 이름을 키로 하는 평면 객체입니다. 쪽 구분 없이 합친 값이므로 서류가 한 종류만 들어 있는 파일에 맞습니다.

처리 시간 · 처리 시간은 파일 크기와 쪽 수에 따라 수 초에서 수 분입니다. 클라이언트 타임아웃은 20분 이상 권장합니다. 서버 내부에서 처리 엔진 호출이 실패하면 재시도가 최대 3회까지 이루어지므로, 실패 응답이 늦게 올 수 있고 OCR 이 중복 실행될 수 있습니다.

요청 · MULTIPART/FORM-DATA

이름	위치	타입	필수	설명
x-clip-key	header	string	필수	API 키. 발급받은 키를 넣으십시오. 예: YOUR_API_KEY
file	form	file	필수	서류 파일. 파일 크기 한도는 서버 설정에 따릅니다.

응답 · 200

서류 종류, 신뢰도, 쪽 수, 그리고 핵심정보입니다.

필드	타입	설명
documentType *	string	서류 종류 이름. 값은 고정 목록이 아닙니다.
confidence *	number	서류 분류 신뢰도(0~1). 현재는 값이 제공되지 않아 항상 0 으로 돌아오므로 판단에 쓰지 마십시오.

필드	타입	설명
pages *	integer	파일의 쪽 수
data *	{ [키]: string }	항목 이름 → 값 문자열. 값이 비어 있는 항목은 빠집니다. 같은 이름의 항목이 여러 쪽에 있으면 뒤의 값이 남습니다.

응답 예 JSON


```
{
  "documentType": "commercial invoice",
  "confidence": 0,
  "pages": 3,
  "data": {
    "Invoice No.": "INV-2026-0931",
    "Total Amount": "USD 125,000.00"
  }
}
```


오류

상태	언제	본문 예
401	x-clip-key 헤더가 없거나 비어 있을 때. 본문의 문구는 화면 안내용이므로, 판별에는 code 값(api_key_required)을 쓰십시오.	{"error":"우측 상단에 API Key 를 입력해 주세요.","code":"api_key_required"}
400	file 필드가 없거나 파일이 아닐 때	{"error":"file is required"}
401	키가 잘못되었을 때. 처리 엔진의 응답 상태를 그대로 전달합니다.	{"error":"Clip API error 401: Unauthorized","status":401}
403	키에 이 작업 권한이 없을 때. 처리 엔진의 응답 상태를 그대로 전달합니다.	{"error":"Clip API error 403: Forbidden","status":403}
404	처리 엔진이 대상을 찾지 못했을 때. 처리 엔진의 응답 상태를 그대로 전달합니다.	{"error":"Clip API error 404: Not Found","status":404}
413	파일 크기가 서버 설정의 한도를 넘을 때. 한도는 서버 설정에 따르며, 현재 한도는 dashboard 에서 확인할 수 있습니다.	{"error":"Clip API error 413: File size exceeds limit","status":413}
429	요청이 너무 많을 때. 잠시 뒤 다시 보내십시오.	{"error":"Clip API error 429: Too Many Requests","status":429}
500	처리 엔진이 오류를 돌려줄 때(상태 코드는 그대로 전달됩니다)	{"error":"Clip API error 500: Internal Server Error","status":500}
502	처리 엔진에 연결하지 못했을 때, 또는 처리 엔진의 응답 형식이 맞지 않을 때	{"error":"Network error: fetch failed","status":502}

상태	언제	본문 예
502	응답 형식이 맞지 않을 때(메시지에 Invalid OCR response shape 가 붙습니다)	{"error":"Invalid OCR response shape: <상세 메시지>","status":502}

예제

cURL CURL

```
curl -X POST "https://<CLIP 설치 주소>/api/shipping/extract" \
  -H "x-clip-key: YOUR_API_KEY" \
  -F "file=@./commercial-invoice.pdf"
```

Python (requests) PYTHON

```
import requests

with open("commercial-invoice.pdf", "rb") as f:
    r = requests.post(
        "https://<CLIP 설치 주소>/api/shipping/extract",
        headers={"x-clip-key": "YOUR_API_KEY"},
        files={"file": ("commercial-invoice.pdf", f, "application/pdf")},
        timeout=1200,
    )
r.raise_for_status()
print(r.json()["data"])
```

JavaScript (fetch) JAVASCRIPT

```
import { readFile } from 'node:fs/promises';

const buf = await readFile('./commercial-invoice.pdf');
const form = new FormData();
form.append('file', new Blob([buf], { type: 'application/pdf' }), 'commercial-invoice.pdf');

const res = await fetch('https://<CLIP 설치 주소>/api/shipping/extract', {
  method: 'POST',
  headers: { 'x-clip-key': 'YOUR_API_KEY' },
  body: form,
});
const { data } = await res.json();
console.log(data);
```

POST /api/shipping/review

파일 서류 분리·추출 시작

파일 1건을 서류별로 나누고 핵심정보를 추출하는 작업을 시작해 jobId 를 받습니다.

한 파일에 여러 서류(예: 신용장과 첨부 서류)가 섞여 있을 때 쪽 단위 호출 없이 서류별 결과를 받을 수 있습니다. `mode` 가 `classify` 면 서류 범위만 받고 키는 받지 않습니다.

처리 시간 · 요청 자체는 즉시 응답합니다. 처리는 GET 조회로 기다립니다.

요청 · MULTIPART/FORM-DATA

이름	위치	타입	필수	설명
x-clip-key	header	string	필수	API 키. 발급받은 키를 넣으십시오. 예: YOUR_API_KEY
file	form	file	필수	서류 파일. 파일 크기 한도는 서버 설정에 따릅니다.
mode	form	string	선택	`classify` (서류 범위만) 또는 `full` (범위와 핵심정보) 기본값: full

응답 · 200

작업 식별자입니다. 결과는 GET 으로 받습니다.

필드	타입	설명
jobId *	string	작업 식별자(UUID)

응답 예 JSON

```
{
  "jobId": "9d4e1a7b-2c3f-4a6e-8b1d-5f0e7c9a2b11"
}
```

오류

상태	언제	본문 예
401	x-clip-key 헤더가 없거나 비어 있을 때. 본문의 문구는 화면 안내용이므로, 판별에는 code 값(api_key_required)을 쓰십시오.	{"error":"우측 상단에 API Key 를 입력해 주세요.","code":"api_key_required"}
400	file 필드가 없거나 파일이 아닐 때	{"error":"file is required"}

참고

이 경로는 clip 웹 화면이 쓰지 않는 경로입니다. 서류 경계는 파일 전체 분류로 정해지므로 화면의 결과와 다를 수 있습니다. 연동 전에 실제 샘플 파일로 응답 형식과 결과를 먼저 확인하십시오.

예제

cURL

```curl
curl -X POST "https://<CLIP 설치 주소>/api/shipping/review" \
  -H "x-clip-key: YOUR_API_KEY" \
  -F "file=@./letter-of-credit-bundle.pdf" \
  -F "mode=full"
```

Python (requests)

```python
import requests

with open("letter-of-credit-bundle.pdf", "rb") as f:
    r = requests.post(
        "https://<CLIP 설치 주소>/api/shipping/review",
        headers={"x-clip-key": "YOUR_API_KEY"},
        files={"file": ("letter-of-credit-bundle.pdf", f, "application/pdf")},
        data={"mode": "full"},
        timeout=60,
    )
r.raise_for_status()
print(r.json()["jobId"])
```

JavaScript (fetch)

```javascript
import { readFile } from 'node:fs/promises';

const buf = await readFile('./letter-of-credit-bundle.pdf');
const form = new FormData();
form.append('file', new Blob([buf], { type: 'application/pdf' }), 'letter-of-credit-bundle.pdf');
form.append('mode', 'full');

const res = await fetch('https://<CLIP 설치 주소>/api/shipping/review', {
  method: 'POST',
  headers: { 'x-clip-key': 'YOUR_API_KEY' },
  body: form,
});
const { jobId } = await res.json();
console.log(jobId);
```

GET /api/shipping/review

서류 분리·추출 결과 조회

jobId 의 상태를 조회합니다. 완료되면 서류별 범위와 핵심정보, 근거 좌표를 돌려줍니다.

`status` 가 `pending` 이면 3초 간격으로 다시 조회하십시오. `done` 이면 `result` 를 받습니다. 결과(오류 결과 포함)는 한 번 받으면 서버에서 삭제됩니다.

처리 시간 · 처리 시간은 파일 크기와 쪽 수에 따라 수 초에서 수 분입니다. 클라이언트 타임아웃은 20분 이상 권장합니다.

요청

이름	위치	타입	필수	설명
x-clip-key	header	string	선택	이 조회는 키를 검사하지 않습니다. 같은 키로 보내는 것을 권장합니다. 예: YOUR_API_KEY
job	query	string	필수	POST 응답의 jobId 예: 9d4e1a7b-2c3f-4a6e-8b1d-5f0e7c9a2b11

응답 · 200

`status` 가 `pending` , `done` , `error` , `unknown` 중 하나입니다. `done` 일 때만 `result` 가 있습니다.

필드	타입	설명
status *	"pending" | "done" | "error" | "unknown"	작업 상태
result	object	status 가 done 일 때의 결과
result.documents *	object[]	서류 목록(시작 쪽 순서). 범위는 대체로 앞 서류의 끝 다음 쪽부터 이어집니다. 다만 시작 쪽이 같은 두 서류가 있으면 그 쪽이 두 서류에 모두 들어갈 수 있습니다.
result.documents[].type *	string	서류 종류 이름. 값은 고정 목록이 아닙니다.
result.documents[].pageStart *	integer	서류의 첫 쪽 번호
result.documents[].pageEnd *	integer	서류의 마지막 쪽 번호(포함)
result.documents[].keys *	object[]	이 서류의 핵심정보. classify 모드에서는 비어 있습니다. 여러 서류에 걸친 항목은 각 서류에 복제됩니다.
result.documents[].keys[].key *	string	항목 이름(예: L/C No., Amount). 서류 종류마다 다릅니다.
result.documents[].keys[].value *	string	추출된 값 문자열. 값을 찾지 못하면 빈 문자열입니다.
result.documents[].keys[].informationList	string[]	여러 줄로 된 항목(예: 서류 목록, 조건 목록)의 줄 단위 값. 해당하지 않으면 생략됩니다.

필드	타입	설명
result.documents[].keys[].processed	object	금액·과부족처럼 구조화된 값(예: 금액은 currency·amount, 과부족은 plus·minus). 해당하지 않으면 생략됩니다.
result.documents[].keys[].reference *	{ [키]: integer[] }	값이 적힌 쪽 번호(문자열 키)와 단어 번호 목록. 번호로 words 에서 좌표를 찾습니다.
result.words *	{ [키]: { [키]: number[] } }	근거 단어 좌표. 2단계 맵입니다. 바깥 키는 쪽 번호(문자열), 안쪽 키는 단어 번호(문자열), 값은 좌표입니다. classify 모드에서는 비어 있습니다.
result.pageCount *	integer	파일의 전체 쪽 수
error	string	status 가 error 일 때의 오류 메시지

응답 예 JSON


```
{
  "status": "done",
  "result": {
    "documents": [
      {
        "type": "letter of credit",
        "pageStart": 1,
        "pageEnd": 2,
        "keys": [
          {
            "key": "L/C No.",
            "value": "LC2026-0042",
            "reference": {
              "1": [
                12
              ]
            }
          }
        ]
      },
      {
        "type": "commercial invoice",
        "pageStart": 3,
        "pageEnd": 3,
        "keys": [
          {
            "key": "Invoice No.",
            "value": "INV-2026-0931",
            "reference": {
              "3": [
                40
```

```
            ]
          }
        }
      ]
    }
  ],
  "words": {
    "1": {
      "12": [
        0.11,
        0.08,
        0.31,
        0.1
      ]
    },
    "3": {
      "40": [
        0.2,
        0.5,
        0.4,
        0.52
      ]
    }
  },
  "pageCount": 3
 }
}
```


오류

상태	언제	본문 예
404	jobId 를 찾을 수 없을 때(잘못된 값, 이미 받은 값, 서버 재시작으로 사라진 값, 정리된 값). 본문의 status 가 unknown 입니다.	{"status":"unknown"}
401	처리 엔진이 키를 거부할 때. 상태 코드는 처리 엔진의 응답을 따르며, 본문의 status 는 error 입니다.	{"status":"error","error":"review 401"}
404	처리 엔진이 파일을 찾지 못했을 때. 본문의 status 가 error 이므로 unknown 과 구분하십시오.	{"status":"error","error":"review 404"}
413	파일 크기가 서버 설정의 한도를 넘을 때. 한도는 서버 설정에 따릅니다.	{"status":"error","error":"File size exceeds limit"}
429	요청이 너무 많을 때	{"status":"error","error":"review 429"}
502	처리 엔진에 연결하지 못했을 때, 또는 상태를 알 수 없을 때	{"status":"error","error":"fetch failed"}

예제

cURL CURL

```
curl "https://<CLIP 설치 주소>/api/shipping/review?job=9d4e1a7b-2c3f-4a6e-8b1d-5f0e7c9a2b11" \
  -H "x-clip-key: YOUR_API_KEY"
```

Python (requests) PYTHON

```
import time
import requests

headers = {"x-clip-key": "YOUR_API_KEY"}
job = "9d4e1a7b-2c3f-4a6e-8b1d-5f0e7c9a2b11"

while True:
    r = requests.get("https://<CLIP 설치 주소>/api/shipping/review", params={"job": job}, headers=headers, timeout
=30)
    body = r.json()
    if body["status"] == "done":
        print(body["result"]["documents"])
        break
    if body["status"] != "pending":
        raise RuntimeError(body.get("error", body["status"]))
    time.sleep(3)
```

JavaScript (fetch) JAVASCRIPT

```
const headers = { 'x-clip-key': 'YOUR_API_KEY' };
const job = '9d4e1a7b-2c3f-4a6e-8b1d-5f0e7c9a2b11';

let body;
for (;;) {
  const res = await fetch('https://<CLIP 설치 주소>/api/shipping/review?job=' + job, { headers });
  body = await res.json();
  if (body.status !== 'pending') break;
  await new Promise((r) => setTimeout(r, 3000));
}
if (body.status !== 'done') throw new Error(body.error ?? body.status);
console.log(body.result.documents);
```

운영

- GET `https://<CLIP 설치 주소>/api/shipping/health` : 인증 없이 `{"status":"ok"}` 를 돌려줍니다. 서버 생존 확인에 씁니다.

- GET `https://<CLIP 설치 주소>/api/shipping/dashboard` : `x-clip-key` 를 보내면 그 키의 권한, 잔여 한도, 설정 정보를 그대로 돌려줍니다. 키와 한도 확인에 씁니다. 응답 필드는 변경될 수 있습니다.


Floater Lab