AIScanRobo なんでも読めるくん API 統合サンプルコード

英語版については README.en.md を参照してください。

このリポジトリには、なんでも読めるくん（ASR Alpha）の API と統合するためのサンプルコードが含まれています。

サンプルコードは JavaScript で書かれており、node.js（バージョン 22 以降を推奨）が必要です。

API インターフェースは apiclient.js に含まれています。あなたのプロジェクトで JavaScript または TypeScript を使用している場合、このコードをあなた自身のプロジェクトにコピーするか、リファレンスとして使用することができます。

その他のファイルはデモの一部であり、参考用のみを目的としています。

URL 概要

- **UI:** https://alpha.aiscanrobo.netsmile.jp
- **API:** https://templateless-api-v2.aiscanrobo.netsmile.jp
- **Swagger ドキュメント:** https://templateless-api-v2.aiscanrobo.netsmile.jp/api-docs

セットアップ

サンプルコードを使用するには、API キーが必要です。

UI (https://alpha.aiscanrobo.netsmile.jp/integration?tab=samplecode) 経由でサンプルコードをダウンロードして API キーを選択した場合、.env ファイルは既にセットアップされています。

そうでない場合は、含まれている .env.sample ファイルを .env にコピーし、編集して API キーを貼り付けてください。

node.js がシステムにインストールされている必要があります。バージョン 22.18.0 以降を推奨します。

- **URL:** https://nodejs.org/en/download/

npm install を実行して依存関係をインストールし、その後 npm run dev で実行してください。

API 概要

API のメイン URL は以下の通りです：

https://templateless-api-v2.aiscanrobo.netsmile.jp

詳細な API ドキュメントについては、以下の Swagger ドキュメントを参照してください：

https://templateless-api-v2.aiscanrobo.netsmile.jp/api-docs API ドキュメントを UI で確認する場合には、https://petstore.swagger.io/#/ が利用可能です。

API キーと JWT トークン

様々なエンドポイントを呼び出すには、まず API キーを使用して JWT（JSON Web Token）を発行する必要があります。

- **POST /sign/apiKey**
- **パラメータ（Body、JSON）：**

key: <API キー>

JWT トークン生成については、apiclient.js の getToken 関数を参照してください。

JWT トークンは 1 時間有効です。API を呼び出す際、トークンを確認し、期限切れの時のみ更新することができます（_updateToken を参照）。

基本的には更新不要ですが、API 呼び出しの前に POST /sign/apiKey を呼び出して、毎回新しい JWT トークンを取得することも可能です。

ドキュメントのスキャン

システムの主な目的は、LLM OCR を使用してドキュメントをスキャンすることです。入力として、PDF、JPEG、PNG ファイルをサポートしています。

API は非同期であり、最初のステップでは入力ファイルをパラメータとしてエンドポイントを呼び出してドキュメントを作成します。これにより、後続のステップで使用されるドキュメント ID が返されます。

ドキュメントが作成された後、システムはバックグラウンドでスキャンします。API エンドポイントを呼び出してドキュメントのステータスをポーリングし、スキャンが完了するまで待つことでステータスを確認できます。

ドキュメントをスキャンする主な方法は 3 つあります：

1. **1 つの抽出設定で全ページを一括抽出：** 単一の LLM リクエストでドキュメント全体をカバーします。ドキュメント全体が連続したデータである場合（注文リストなど）に使用します。
2. **1 つの抽出設定で各ページを個別抽出：** これは、ドキュメントの各ページに対して LLM に 1 つのリクエストを送信します。各ページが独立したデータを参照する場合（レシートのコレクションなど）に使用します。
3. **各ページの抽出設定を個別に選択：** これにより、ページの任意のグループ化が可能になり、各グループは個別の LLM リクエストとして送信されます。複数のドキュメントが単一の PDF ファイルに結合されている場合に使用します。

ドキュメントがスキャンされた後、元のファイル、各ページの変換された JPEG、スキャン結果など、ドキュメントに含まれる様々な情報を取得できます。

ドキュメントは CSV、Excel、JSON としてエクスポートすることも可能です。

API は一般的に ASR Alpha UI ができることすべてを実装することを可能にしますが、一部の機能は、追加実装直後などの理由で API では公開されていない場合があります。

プロンプト

UI では「抽出設定」という用語が使用されます。API では、これはプロンプトと呼ばれます。この文書では、一般的に「プロンプト」という用語を使用します。

ドキュメントをスキャンできるようになる前に、少なくとも 1 つのプロンプトを作成する必要があります。まず UI を使用してプロンプトを作成し、調整をお願いします。

UI でプロンプトを作成、編集する場合には、https://alpha.aiscanrobo.netsmile.jp/document-type を使用してください。

プロンプトの設定方法は 2 つあります。1 つは、抽出したい内容を自由に文章で書く方法で、「シンプル」タブから作成します。もう 1 つは、項目名やテーブルの列名を具体的に指定して、細かく指示を出す方法です。

項目名やテーブルの列名を具体的に指定して、細かく指示を出す方法を使用する場合、LLM はプロンプトで指定した順序で結果を返します。

簡単なテストの場合、「全ての項目を取得する」のようなプロンプトがうまく機能します。

ドキュメントをスキャンするには、使用したいプロンプトのプロンプト ID を提供する必要があります。プロンプトの ID は UI の「抽出設定一覧」で左端の列（「ID」という名前）に表示されます。

UI 使用せずに、API を使用してプロンプトを作成および検索することもできます。詳細については、この文書の下部の「更に活用する場合」セクションを参照してください。

ドキュメント作成

ドキュメントを作成するために使用される方法は、グループタイプによって異なります：

1. 各ページを個別に処理（1 つの抽出設定で全ページを一括抽出）
2. ドキュメント全体を単一のグループとして処理（1 つの抽出設定で各ページを個別抽出）
3. カスタムグループ化（各ページの抽出設定を個別に選択）

各ページを個別に処理

これは API 実装において最も簡単なグループタイプであり、POST /documents エンドポイントを呼び出すことで実現できます。このエンドポイントは、スキャンするファイル（単一の PDF または JPEG/PNG ファイルのセット）を引数として受け取ります。

POST /documents エンドポイントは、以下のステップを自動的に実行するオールインワンエンドポイントです：

- ファイルをアップロードし、データベースにドキュメントを作成
- （PDF の場合）PDF の各ページを JPEG に変換（後の処理に必要）
- グループを作成、PDF の各ページに 1 つずつ
- ファイルをスキャンのためにキューに入れる（すべてのページに同じプロンプトを使用）

simple.js のサンプルコードでこれを行う方法を示しており、API クライアントコードは apiclient.js にあります。

- **POST /documents**
- **パラメータ:** Body、FormData

パラメータ詳細：

- file: 単一の PDF ファイルまたは JPEG もしくは PNG ファイルのセット。添付されたファイルは単一のドキュメントとして作成されます。複数の JPEG もしくは PNG ファイルを別々のドキュメントとして処理する場合もしくは複数の PDF ファイルを処理する場合には、file パラメータを複数回追加してください。
- origin（オプション）: これは asralpha に設定する必要があります。ドキュメントがどのシステムから発信されたかを識別するために使用します。同じ API は DocuMatch（dm）などの他の製品にも使用されます。
- pdfConversionMethod: standard に設定する必要があります。
- scanMode: なんでも読めるくんでの使用の場合、llm に設定する必要があります。
- useOcr: 読取処理時にドキュメントと併せて標準 OCR の結果を LLM に渡し、使用するかどうかを設定するブール値。ほとんどの場合、これを true に設定すると、より高い精度が得られる可能性があります。
- omitTemplateDetection（オプション）: AIScanRobo テンプレートと統合し、LLM に送信する前に ASR でテンプレート検出を試行するには、これを false に設定してください。テンプレートが見つかった場合、ドキュメントは LLM を使用する代わりに ASR を使用してスキャンされます。LLM のみでスキャンするには、パラメータを省略するか true に設定してください。
- promptId: このパラメータは、ドキュメントのスキャンに使用する数値プロンプト ID に設定する必要があります。プロンプト ID を設定するか、自動識別を使用する場合には値を -1 に設定してください。

プロンプトをプログラム的に作成および管理したい場合は、後述の「更に活用する場合」 -

> 「プロンプト」を参照してください。

プロンプト検出について： promptId が -1 に設定されている場合、プロンプト検出が実行されます。過去の読取結果から各フォーマットに合わせたプロンプトを使用するため、この機能を使用するには、なんでも読めるくんで初めて処理するフォーマットは一度 UI 上にアップロードし、手動でプロンプト ID を選択する必要があります。
過去の読取結果に新しいドキュメントと類似したドキュメントがない場合、スキャンは失敗し、ドキュメントを再スキャンして promptId に明示的な ID を設定する必要があります。

レスポンス例：
JSON

```
{
  "id": "8294",
  "status": "processingPending",
  "scanMode": "llm",
  "pdfConversionMethod": "standard",
  <その他の省略されたフィールド>
}
```

返された id 値を使用してドキュメントにアクセスします。結果のポーリング方法とデータのエクスポート方法については以下を参照してください。

ドキュメント全体を単一のグループとして処理 / カスタムグループ化
API を使用する場合、「ドキュメント全体を単一のグループとして処理」は、単一のグループのみを作成する「カスタムグループ化」の変種です。
これを実装するには、いくつかの API 呼び出しが必要です。
single-group.js と custom-groups.js のサンプルコードを参照して、これを実装する方法を確認してください。
各ページに個別のグループを作成し、各ページに異なるプロンプト ID を使用したい場合もこの方法を使用できます。

ステップ 1: ドキュメント作成

- POST /documents/create
- **パラメータ:** Body、FormData

入力パラメータとレスポンスは POST /documents と全く同じです。
このエンドポイントはデータベースにドキュメントを作成し、その ID を返すだけです。
PDF を JPEG に変換せず、スキャンも実行しません。
apiclient.js の関数 createDocument を参照してください。

ステップ 2: 処理（変換）

- **PATCH /documents/{documentId}/reprocess**
- **パラメータ:** Body、JSON

パラメータ詳細：

- pdfConversionMethod: standard に設定する必要があります。
- scanAfterProcessing: ブール値。false に設定する必要があります。true に設定すると、システムは POST /documents/create に送信されたパラメータに従って各ページに 1 つのグループでドキュメントをスキャンします。

これにより、PDF の各ページが JPEG に変換され、ドキュメントの各ページのデータベース構造が作成されます。ドキュメントが JPEG または PNG のセットの場合も、データベース構造の作成が必要となるため設定する必要があります。
apiclient.js の関数 processDocument を参照してください。

ステップ 3: ページグループの作成
ページグループを作成するためには、まずドキュメントの各ページのページ ID を取得する必要があります。これがどのように行われるかは custom-groups.js を参照してください：

1. まず GET /documents/{documentId} を呼び出し
2. pages 配列を読み取り、各ページのページ ID を取得

- **PATCH /documents/{documentId}/groups**
- **パラメータ:** Body、JSON

JSON には単一のフィールド groups が含まれ、作成するすべてのグループが含まれている必要があります。

例 1: 3 ページドキュメントの 3 ページをカバーする単一グループ
JSON

```
[
  { "pages": [16301, 16302, 16303], "promptId": 146 }
```

]

例 2: 4 ページドキュメントの複数グループ
JSON
```
[
    { "pages": [16305], "promptId": 146 },
    { "pages": [16306, 16307], "promptId": 147 },
    { "pages": [16308], "promptId": 148 }
]
```

apiclient.js の関数 createPageGroups を参照してください。

注意: グループを作成する際に一部のページをスキップすることは可能です。これらはスキャン時に無視されます。ただし、同じページを複数のグループに追加することはできず、グループは昇順に配置する必要があります（つまり、ページ 1 を持つグループはページ 2 を持つグループより前にリストされる必要があります)。

ステップ 4: スキャンのためのキュー登録
グループを作成した後、ドキュメントをスキャンのために送信します：

- **POST /documents/{documentId}/scan**
- **パラメータ:** Body、JSON

パラメータ詳細：
- scanMode: llm に設定
- useOcr: ブール値
-

apiclient.js の関数 scanDocument を参照してください。

ドキュメントステータスの読み取り
API は非同期です。上記で説明したようにドキュメントを作成した後、ドキュメントはバックエンドでスキャンのためにキューに入れられます。
GET /documents/{documentId} エンドポイントを使用してドキュメントの現在のステータスを読み取ります。フィールド status は、ドキュメントのスキャンが完了すると scanned になります。
ドキュメントのスキャン完了を待つ方法（ポーリング）の例については、apiclient.js の

waitForDocumentToBeScanned を参照してください。
10 秒間隔以上でポーリングすることをお勧めします。API がブロックする可能性があるため、非常に短い間隔でポーリングしないでください。

完全なドキュメントデータの読み取り
GET /documents/{documentId} を使用して完全なドキュメントを取得します。
status が scanned の場合、結果を含むドキュメントに関するすべての詳細がレスポンスに返されます。
レスポンスには、ドキュメントの各グループの 1 つのエントリを含む配列 pageGroups があります。
各グループ内に、グループのスキャン結果が JSON データとして含まれます：

- extractedItems: LLM から抽出されたフィールドデータ
- correctedItems: ドキュメントが UI で修正されている場合のフィールドデータ、そうでなければ null
- extractedTables: LLM から抽出されたテーブルデータ
- correctedTables: ドキュメントが UI で修正されている場合のテーブルデータ、そうでなければ null
-

JSON データの出力については、「ドキュメントのエクスポート」を参照してください。

ドキュメントの再スキャン
POST /documents/{documentId}/scan を使用してドキュメントを再スキャンできます。グループセットをリセットする場合、オプションで groups エンドポイントを呼び出すことで、まず異なるグループセットをリセットし、再作成することができます。

ドキュメントのエクスポート
ドキュメントのデータは、JSON、CSV、Excel としてエクスポートできます。単一ドキュメントのエクスポートと一括エクスポートの両方のエンドポイントがあります。
両方のエンドポイントで、エクスポートタイプは Accept パラメータの設定によって制御されます：

- **CSV:** text/csv
- **EXCEL:** application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
- **JSON:** application/json

出力されるデータの出力単位も制御できます：

- **結合:** すべてのデータを単一ファイルでエクスポート
- **グループ別:** 各グループを個別にエクスポート。各グループに対してファイルが作成され、データは ZIP ファイルとしてダウンロードされます。このオプションを使用するには、Accept ヘッダーに ,application/zip を追加する必要があります。
-

例：

- Accept: text/csv,application/zip - ZIP ファイル内のグループ別 CSV ファイルをエクスポート
- Accept: application/vnd.openxmlformats-officedocument.spreadsheetml.sheet - すべてのグループ（またはグループエクスポートのファイル）のデータを結合した単一の Excel ファイルをエクスポート

単一ドキュメントエクスポート

- **GET /documents/{documentId}/groups/export**
- ***パラメータ:*** クエリ

パラメータ詳細：

- csvEncoding: CSV ファイルのエンコーディングを設定。utf-8-bom、utf-8、Shift_JIS のいずれか（CSV ファイルのみ）
- csvNewlineOptions: CSV ファイルの改行フォーマットを設定。windows（CRLF）または unix（LF）のいずれか（CSV ファイルのみ）
- filename: エクスポートに使用するファイル名。{documentId}-{documentName} のようなテンプレート文字列が可能
- includeMetadata: ファイル名やプロンプト ID などの情報を含む追加のメタデータを、エクスポートされたファイルの追加列として含めるかどうかを制御するブール値フラグ

apiclient.js の exportDocuments を参照してください。

一括ドキュメントエクスポート

- **GET /documents/export**
- ***パラメータ:*** クエリ

パラメータ詳細：

- csvEncoding: CSV ファイルのエンコーディングを設定。
- csvNewlineOptions: CSV ファイルの改行フォーマットを設定。
- filename: エクスポートに使用するファイル名。
- includeMetadata: 追加のメタデータを含めるかどうか。
- documentId: 一括エクスポートの場合、このパラメータはパスパラメータではなくクエリパラメータとして設定されます。エクスポートする各ドキュメントに対して一度繰り返します。

apiclient.js の exportDocuments を参照してください。

ドキュメントの検索

- **GET /documents/search**
- **Parameters:** Query

検索エンドポイントはページ分割されています。ページネーションは以下のクエリパラメータで制御されます：

- page（デフォルトは 1 ページ目）：取得するページ
- pageSize（デフォルトは 100）：各呼び出しで返すアイテムの数。最大値は 500 です。

例: /documents/search?page=2&pageSize=50
フィルタリングには以下のクエリパラメータが利用可能です：

- ids: ドキュメント ID のコンマ区切りリスト。
- name: ドキュメント名（部分一致）。
- status: ドキュメントのステータス。
    - 利用可能なステータス値: scanPending, scanError, processingError, scanned, unprocessed, processed
- confirmed: ドキュメントが確定されているかどうかを示すブール値。
- createdBy: 作成者のユーザーID。
- origin: ドキュメントのソース。
- scanMode: 使用されたスキャンモード。
- promptIds: 関連するプロンプト ID のリスト。
- tagIds: タグ ID のコンマ区切りリスト。
- createdAfter / createdBefore: 作成日時によるフィルタ。
- updatedAfter / updatedBefore: 更新日時によるフィルタ。

- downloaded: ドキュメントがダウンロードされたかどうか。

返されるドキュメントのソート順を制御できます：

- sortBy: 結果をソートするためのフィールド。
- sortOrder: 結果のソート順（a は昇順、d は降順)。

apiclient.js の searchDocuments を参照してください。

更に活用する場合

以下のエンドポイントはサンプルコードではカバーされていません。

プロンプト API

プロンプトには CRUD と検索エンドポイントが利用可能です。

- **GET /prompts/{promptId}** 既存のプロンプトの詳細を返します。
- **POST /prompts** 新しいプロンプトを作成します。 **パラメータ:** Body、JSON

基本パラメータ例:

JSON

```
{
  "name": "string",
  "documentType": "string",
  "exportTemplate": "string",
  "extractTable": "boolean",
  "userCustomInstructions": "string",
  "fieldsPrompt": "string",
  "tablePrompt": "string",
  "textualCombinedPrompt": "string"
}
```

- name と documentType は必須です。
- exportTemplate はオプションで、デフォルトファイル名を設定します。
- extractTable はテーブル抽出を行う場合に true に設定します。

シンプルプロンプトの作成例:

textualCombinedPrompt を使用します。

JSON

```
{
```

```
  "name": "テスト",
  "documentType": "ドキュメント",
  "textualCombinedPrompt": "全ての項目を取得する"
}
```

詳細なプロンプトの作成:
fieldsPrompt と tablePrompt を使用します。
fieldsPrompt の例:
JSON
```
"fieldsPrompt": [
  { "item": "項目 1" },
  { "item": "項目 2", "instruction": "..." }
]
```

- **DELETE /prompts/{promptId}** プロンプトを削除
- **PATCH /prompts/{promptId}** プロンプトを更新
- **GET /prompts/search** プロンプトの検索。ページネーションとフィルタリングが可能です。

フィルタリングパラメータ:
ids, name, tagIds, documentType, extractTable, manualOrder, createdBy, createdAfter など。
ソート制御:
sortBy: 結果をソートするためのフィールド。sortOrder: 結果のソート順。