Low/No-Code（nocodb）

目次

Low/No-Code（nocodb）

はじめに

ETL や BI ツールは IO やエラー制御、編集に関するコーディングをせずにデータの加工や表示ができます。特にデータソースに RDB を使えば相関データを結合したりフィルタリングする等の複雑な結果が得られる上に列名や属性をメタ情報から取得できるので省コード化が可能です。適用分野・業務によってはツールだけでシステムが構築できるかもしれません。

但し、入力データを会話的に受けたりデータの選択・更新が必要ならそれ用のアプリケーションが必要になります。そのような場面ではコーディングの量を減らす LowCode やコーディングが不要な NoCode を検討する価値があります。

Low/No-Code でアプリケーションを作るソフトウェアは EC サイト構築用の市販製品が出ていますが、オープンソースで公開されているものもあります。オープンソースの方は「社内向けツール」という位置づけのものが多く、情報共有に重点を置いています。社内向けといっても、開発が盛んな米国では ITC 人材の過半数がユーザ企業の社内に在籍[1]する環境、数百人の開発貢献者がいるオープンソースプロジェクトが多数ある等、日本でイメージする一時利用のツールとは様相が異なっており、Low Code に分類されるプロジェクトは GitHub 2022 年 1/1～9/30 の活発度で 1 位になっています[2]。各プロジェクトは類似した機能や使い勝手が多いので、代表に一つ取り上げて説明します。

1. Low/No-Code の実現方式

ローコード、ノーコードを実現する方式には幾つかありますが、大まかには以下のものがあります。

① カスタマイズ可能な CMS やブログ、情報共有のテンプレート（カスタマイズ次第でコード要）
② 組合せてアプリが作れる GUI や DB、セキュリティ、ワークフロー等の部品群（ノーコード製品）
③ 主に RDB の検索・更新を目的にしたスプレッドシート（CRUD[3]に特化）
④ 入出力項目を定義し、各種のデータソースに接続して処理を行うフレームワーク（多少のコード要）

＜プロジェクトの例＞

● ToolJet

ToolJet は寄稿者が 200 人を超えるトップクラスの人気（星の投票数の多さ）のオープンソースで、概要や機能は GitHub のサイト[4]で以下のように説明されています（2022 年 12 月時点）。

『ToolJet は、最小限のエンジニアリング作業で内部ツールを迅速に構築および展開するための **オープンソースのローコード フレームワーク**です …（中略）…

・ビジュアル アプリ ビルダー: Tables、Charts、Lists、Forms、Progressbars などの 40 以上の組み込みのレスポンシブ ウィジェット。
・マルチプレイヤー編集では、複数のユーザーが同時にアプリ ビルダーを使用できます。
・40 以上のデータ ソース:データベース、クラウド ストレージ、API に接続します。
・デスクトップとモバイル:レイアウトの幅をカスタマイズして、さまざまな画面を…（以下略）』

---

[1] 総務省トップ > 政策 > 白書 > 30 年版 > 日米の ICT 人材の比較
https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h30/html/nd114140.html

[2] Activity levels of popular topics https://ossinsight.io/2022#technology-trends

[3] Create Read Update Delete：データのライフサイクルを表す CRUD 図がよく使われます

[4] ToolJet/ToolJet https://github.com/ToolJet/ToolJet

● nocodb

「オープンソースの Airtable の代替」と説明されています。Airtable とは、2012 年に設立された Airtable 社がクラウド環境で提供しているデータベースとスプレッドシートを統合した製品です。

寄稿者は ToolJet よりも少ないですが、獲得している星の数は上回っています（2022 年 12 月時点）

動作環境は ToolJet やその他のツール／フレームワークが Docker かクラウドサービスの使用を前提（Linux 環境が必要）としているのに対し、nocodb は Windows にもインストールが可能です。

2. nocodb のセットアップ

nocodb(v0.100.2)の配布形態（ライセンス：AGPL v3[5]）は Docker の仮想イメージや Node アプリ他いくつかありますが、ここでは Windows のローカルに設置して利用する手順を説明します。

（1）Node.js

nocodb 実行形式を使う場合は Node.js は不要（exe ファイルに内臓）です。Node アプリとして実行する場合は Node.js（ver 14 以上）が必要なのでダウンロードサイト[6]より最新版の.zip を取得して解凍し、環境変数の PATH に加えてください。node -v コマンドで v14 以上がでればOK です。

（2）ダウンロード

以降、実行形式を使う前提です。公式サイト[7]の Binaries に書かれているコマンドを実行します。

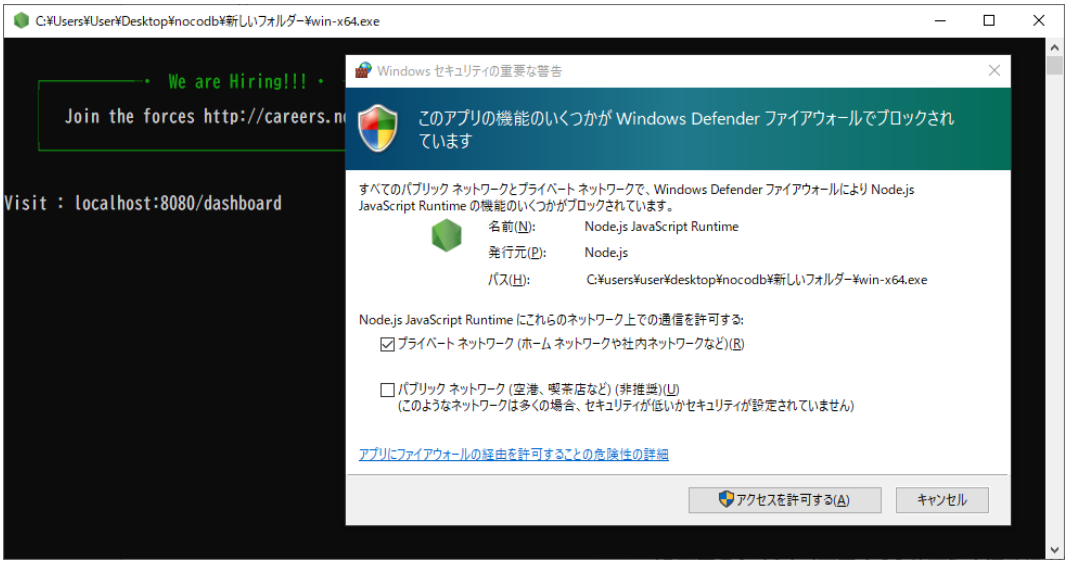
ここに書かれている iwr コマンドは Powershell で実行します。Windows10 からは curl コマンドが入っているので、コマンドプロンプトから以下のように実行することもできます。

```
C:¥Users¥User¥Desktop¥nocodb>curl -L -o win-x64.exe http://get.nocodb.com/win-x64.exe
```

（3）実行

ダウンロードした win-x64.exe を実行すると以下の警告が表示されるので、[アクセスを許可する]を押下し、ブラウザから http://localhost:8080/dashboard を開きます。

exe と同じフォルダに noco.db が作られて管理者のアカウント等の設定情報が保存されます。



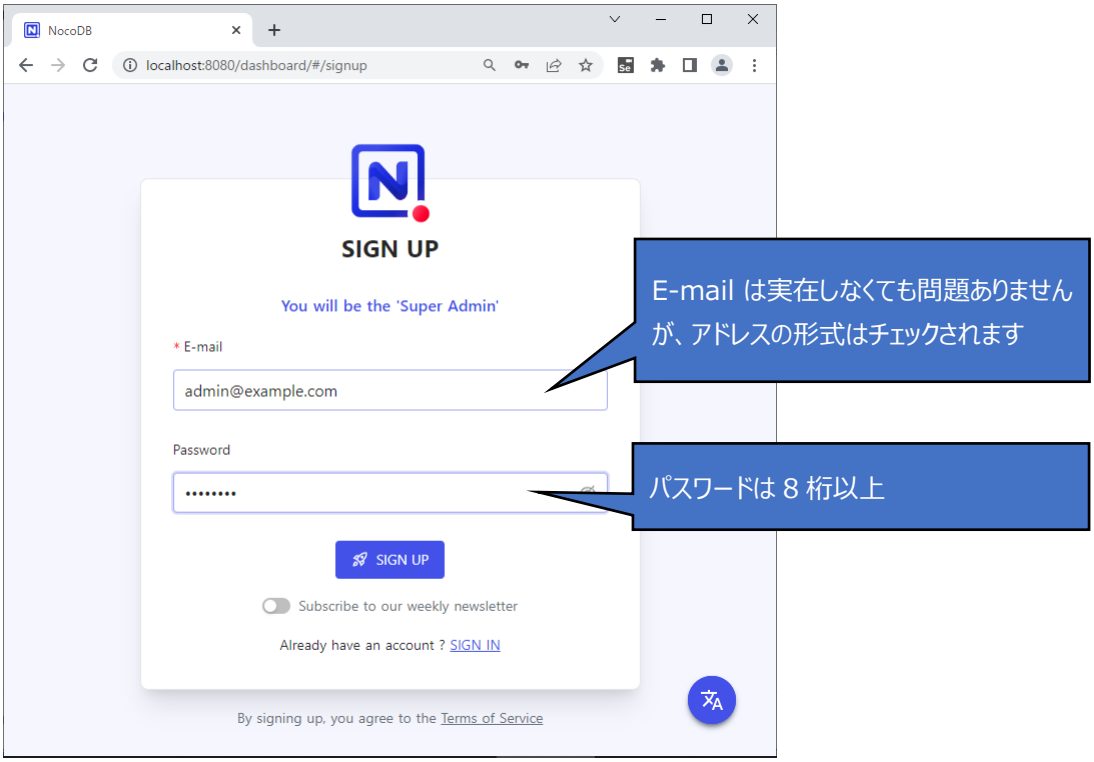
[5] AGPL v3…無償で利用・改造・再配布・商用利用可能ですが利用者へのソース公開が必要です

[6] Node.js ダウンロードサイト https://nodejs.org/ja/download/

[7] nocodb ダウンロードコマンド https://github.com/nocodb/nocodb#binaries

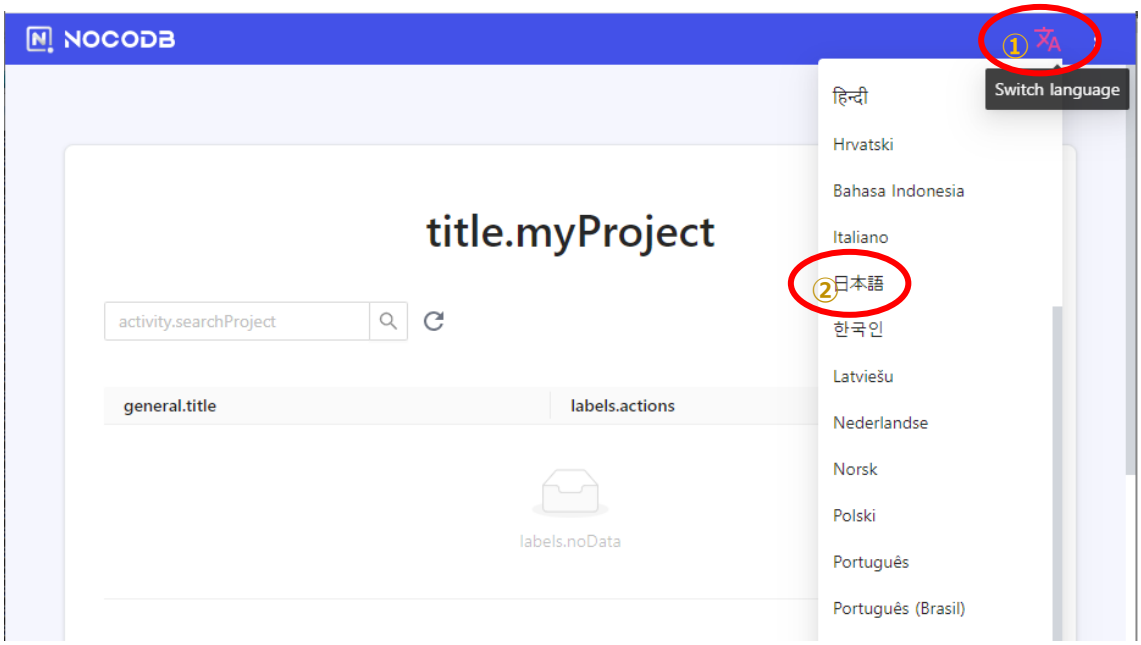
2.1. 初回 SIGN UP／管理者登録

初回起動時に local:8080/dahsboad を開くと SIGN UP 画面がでます（次回は SIGN IN=ログイン）。
ここで入力した E-mail と Password が管理者として登録されます。



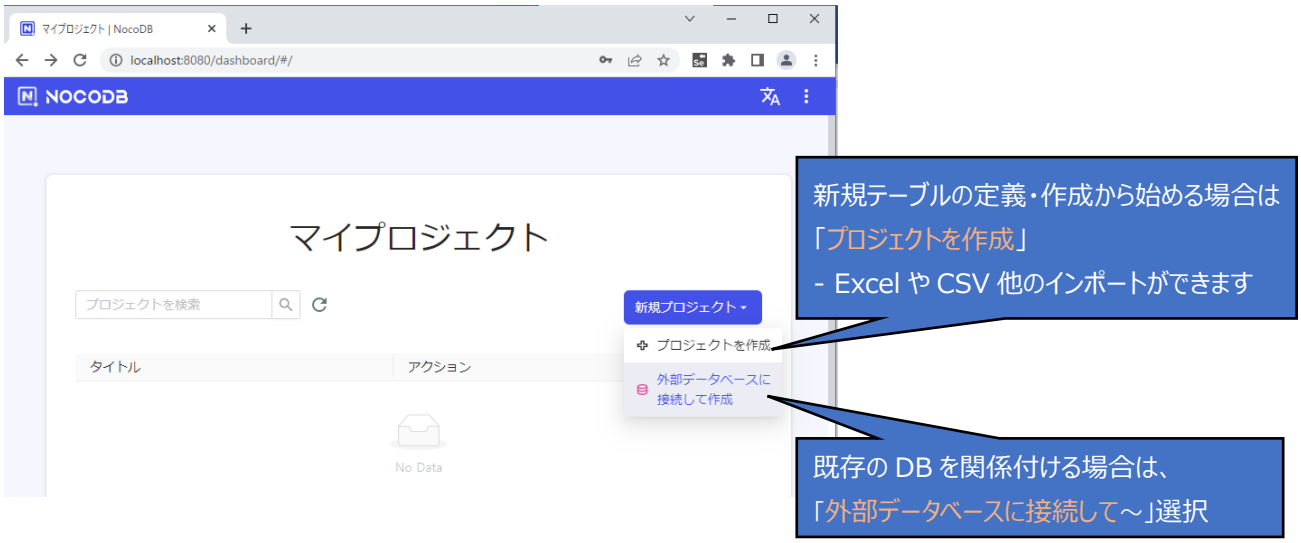
2.2. 日本語化

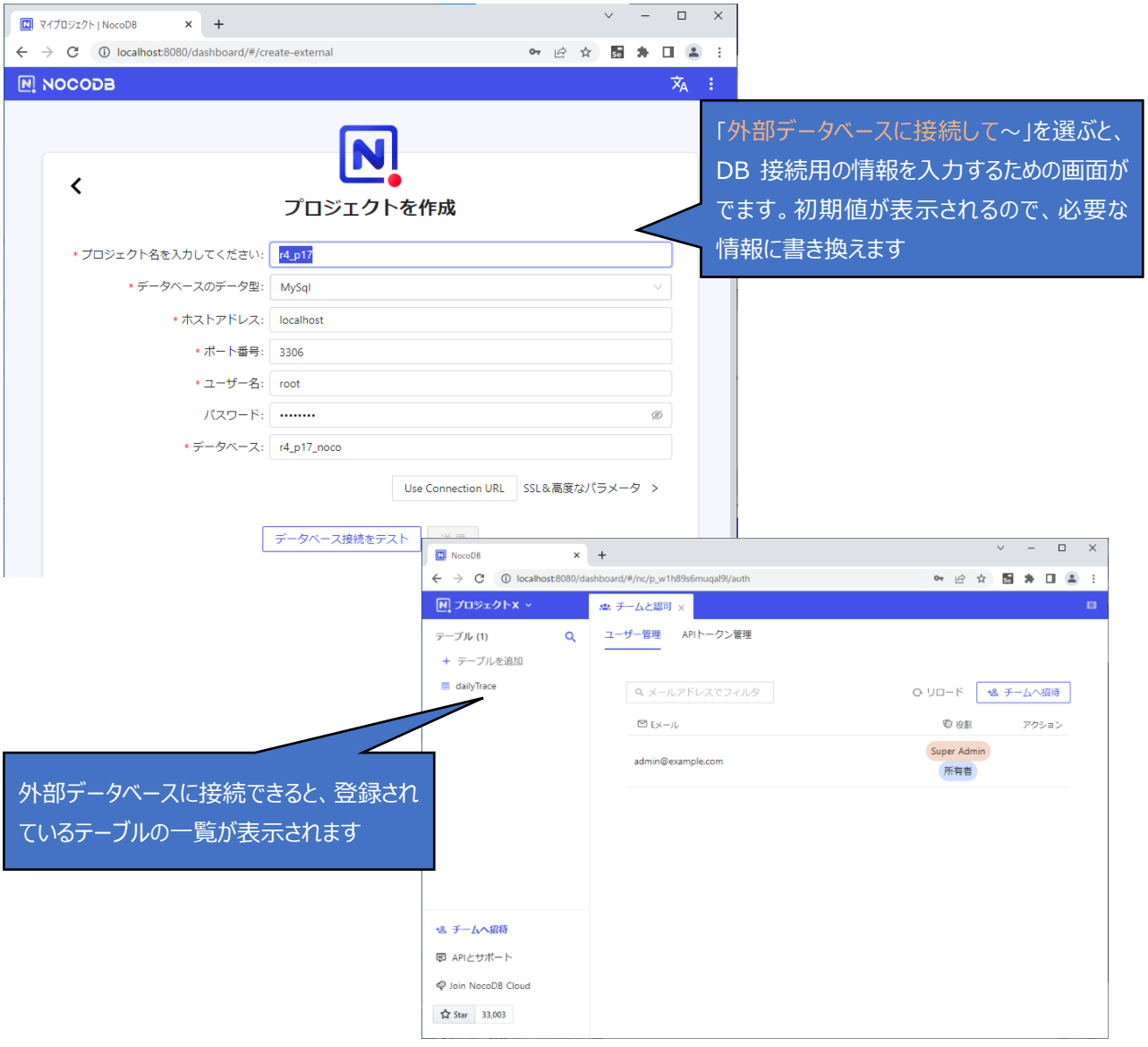
サインアップ後に表示される画面の右上①から日本語②を選択すると日本語表示になります。



2.3. プロジェクトの作成

作業はプロジェクトの単位で行います。プロジェクトはデータベースに関係づけて作成します。



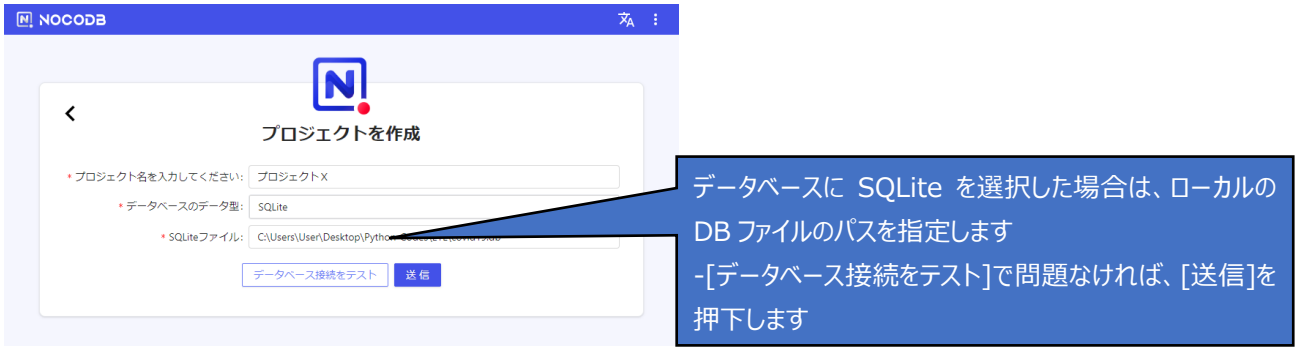


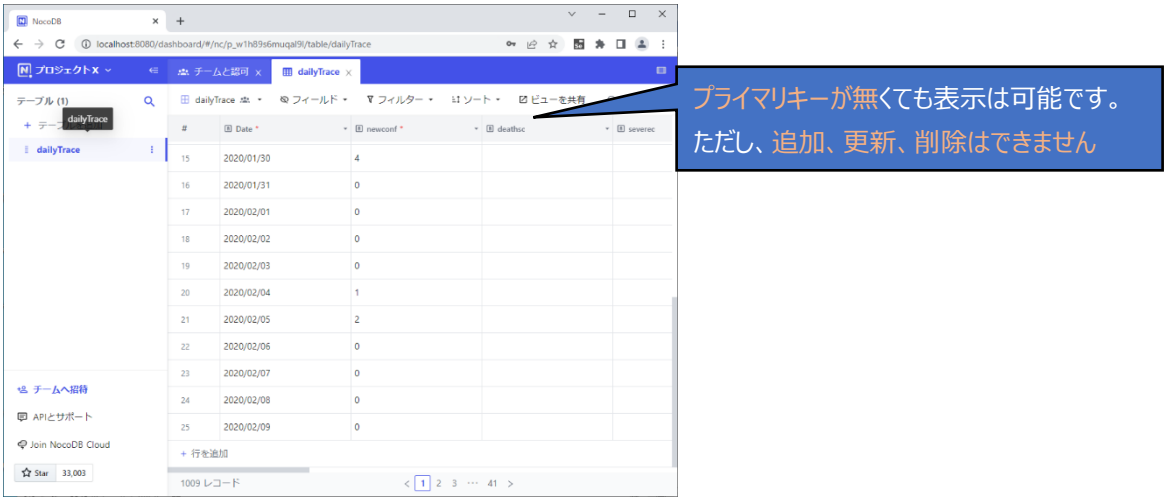
3. プロジェクトの運用（CreateReadUpdateDelete）

プロジェクトに関係付いた DB・スキーマに対してテーブルや行の追加／更新／削除を行う機能がありますが、実際に行うことができるのは利用者の権限とデータベースの機能に制限されます。
【nocodb の制限】プライマリキーが付いてないテーブルの行追加・更新・削除はできません

3.1. SQLite3

SQLite3 は手軽に使えて便利です。但し、プライマリキーを含む制約条件は後付けできない（定義時のみ設定可能）ため、プライマリキーが無い既存のテーブルは更新ができません。



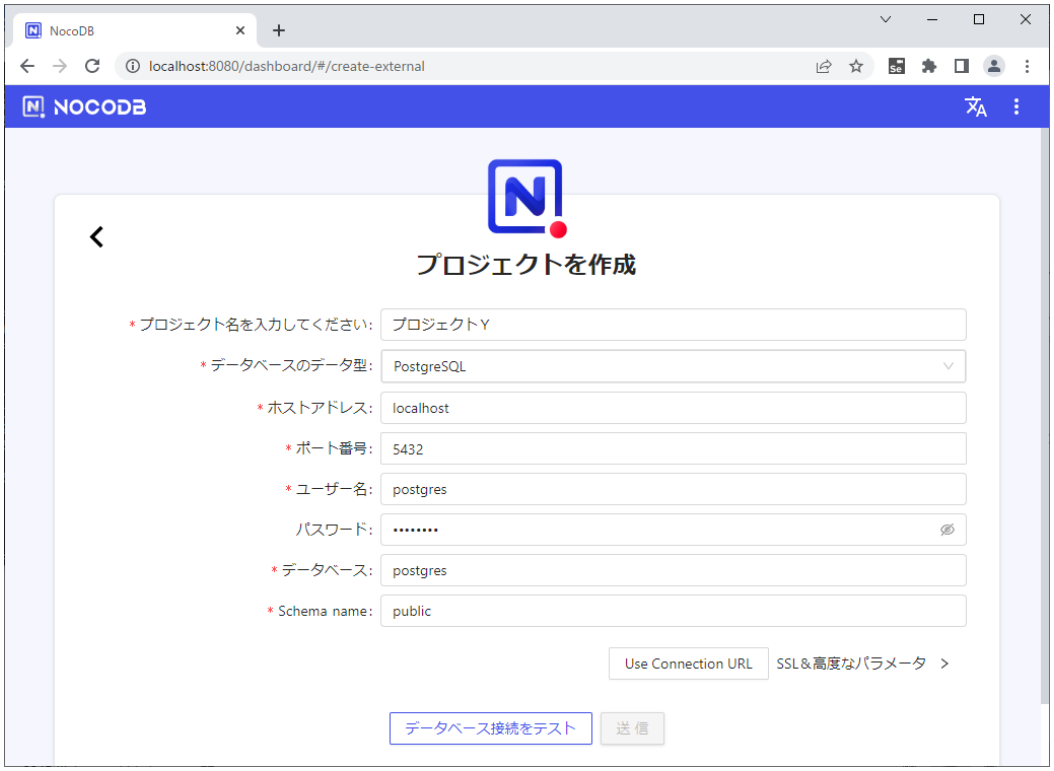


nocodb で追加した①テーブルはデフォルトでプライマリキーが付き、更新が可能です。

3.2. PostgreSQL

PostgreSQL の真偽値や配列、JSON 等のデータを nocodb から登録、更新することができます。

（1）プロジェクト作成画面から入力するデータベース接続情報



（2）列の表示（リスト形式）

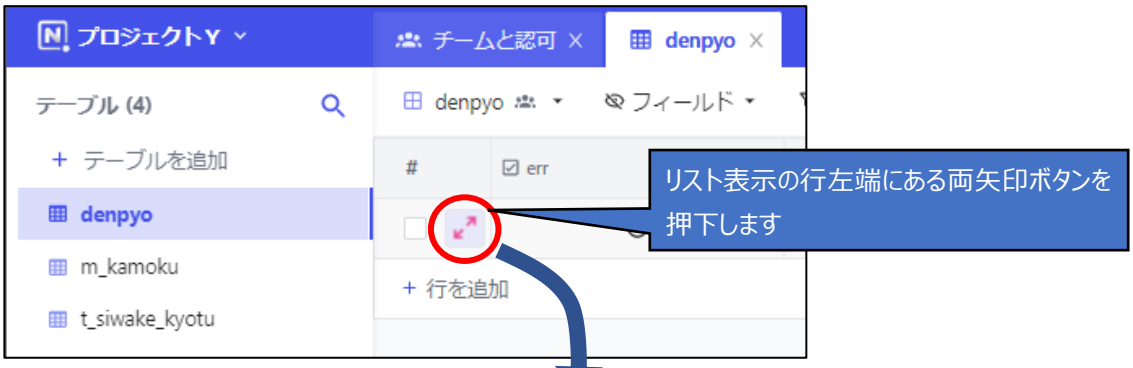
【表示しているテーブルの定義】

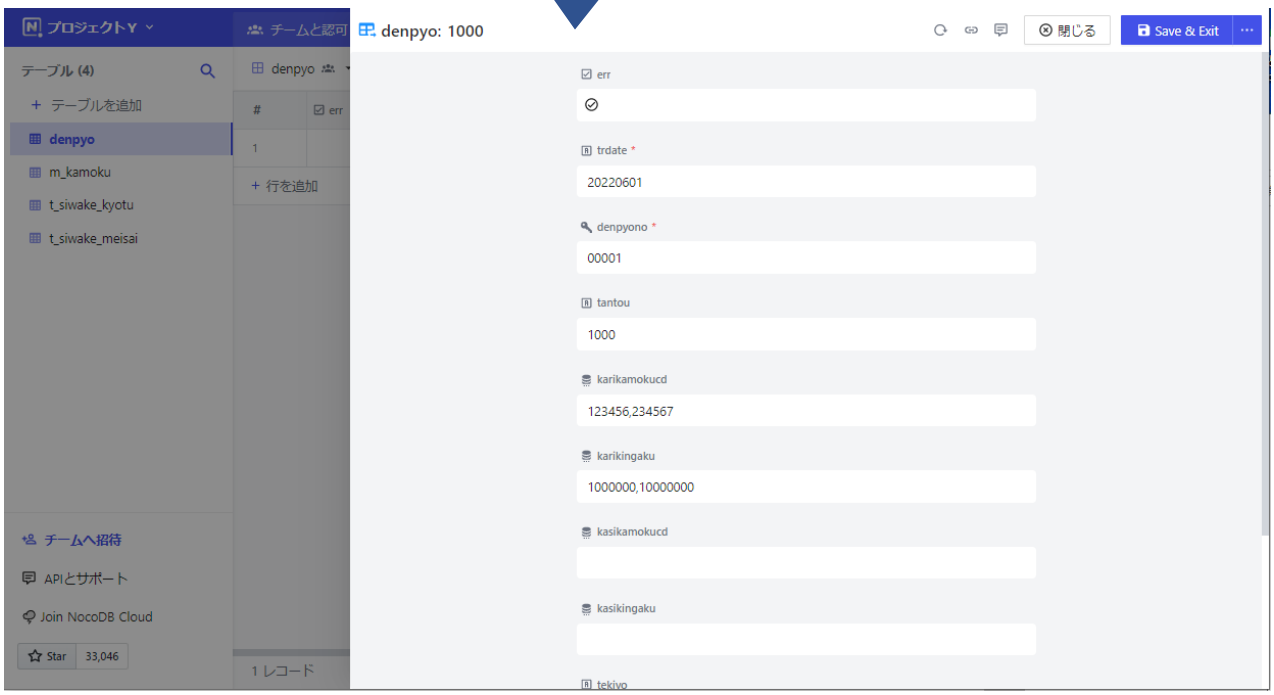
```
postgres=# ¥d denpyo
                    テーブル"public.denpyo"
     列     |          タイプ          | 照合順序 | Null 値を許容 | デフォルト
------------+--------------------------+----------+---------------+------------
 err        | boolean                  |          |               |
 trdate     | character(8)             |          | not null      |
 denpyono   | character(5)             |          | not null      |
 tantou     | character(4)             |          |               |
 karikamokucd | character(6)[]           |          |               |
 karikingaku | character varying(13)[]  |          |               |
 kasikamokucd | character(6)[]           |          |               |
 kasikingaku | character varying(13)[]  |          |               |
 tekiyo     | character varying(50)    |          |               |
 sysdate    | character(8)             |          | not null      |
 errmessage | character varying(100)   |          |               |
インデックス:
    "pk_denpyo" PRIMARY KEY, btree (denpyono)
```


4. ビュー／フォーム

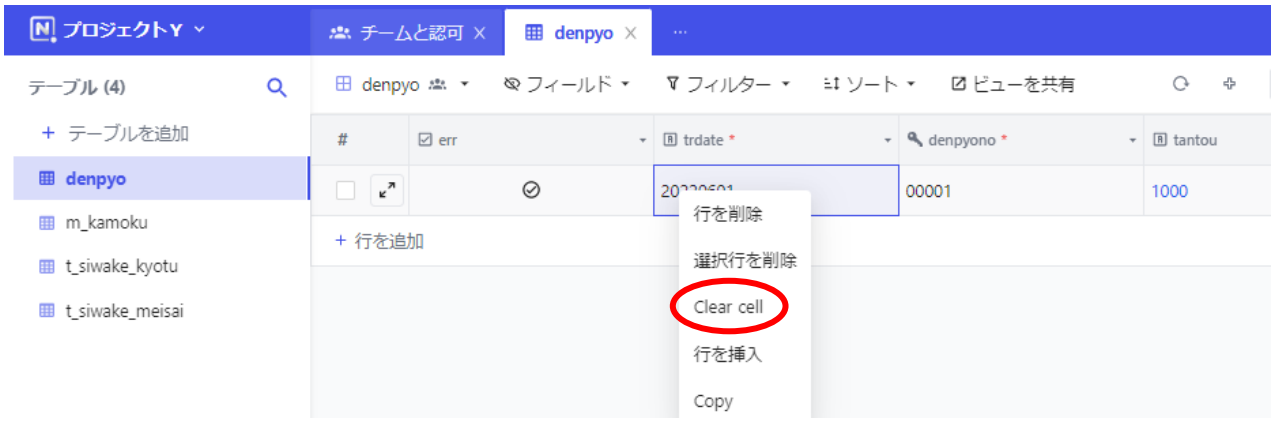
リスト形式の表示でデータの更新ができますが、フォーム形式に表示して更新することもできます。

（1）フォーム形式の表示



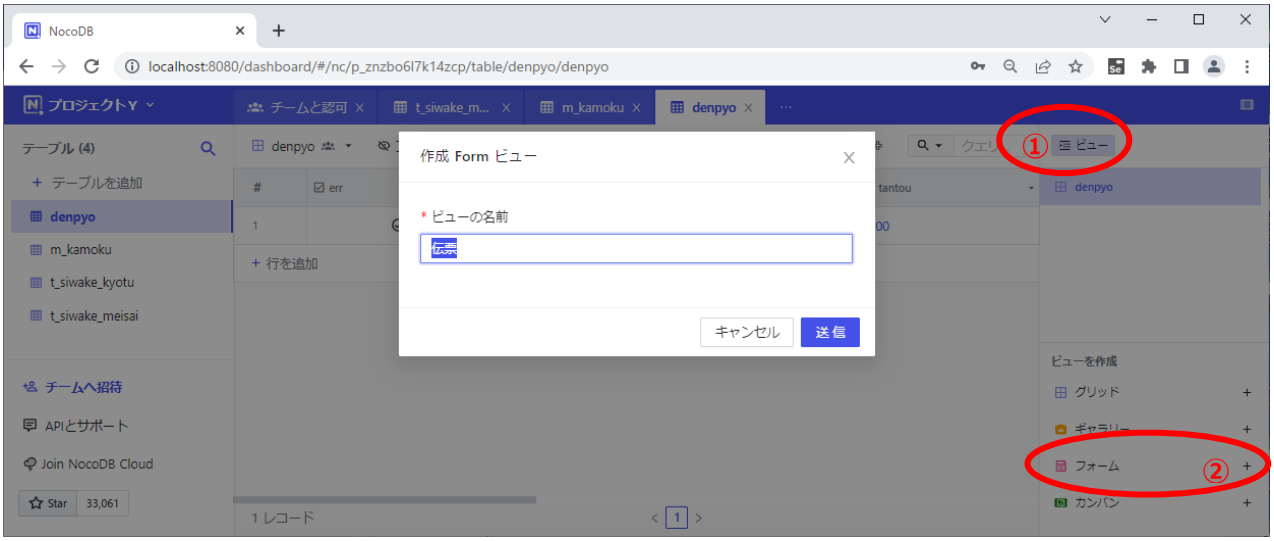


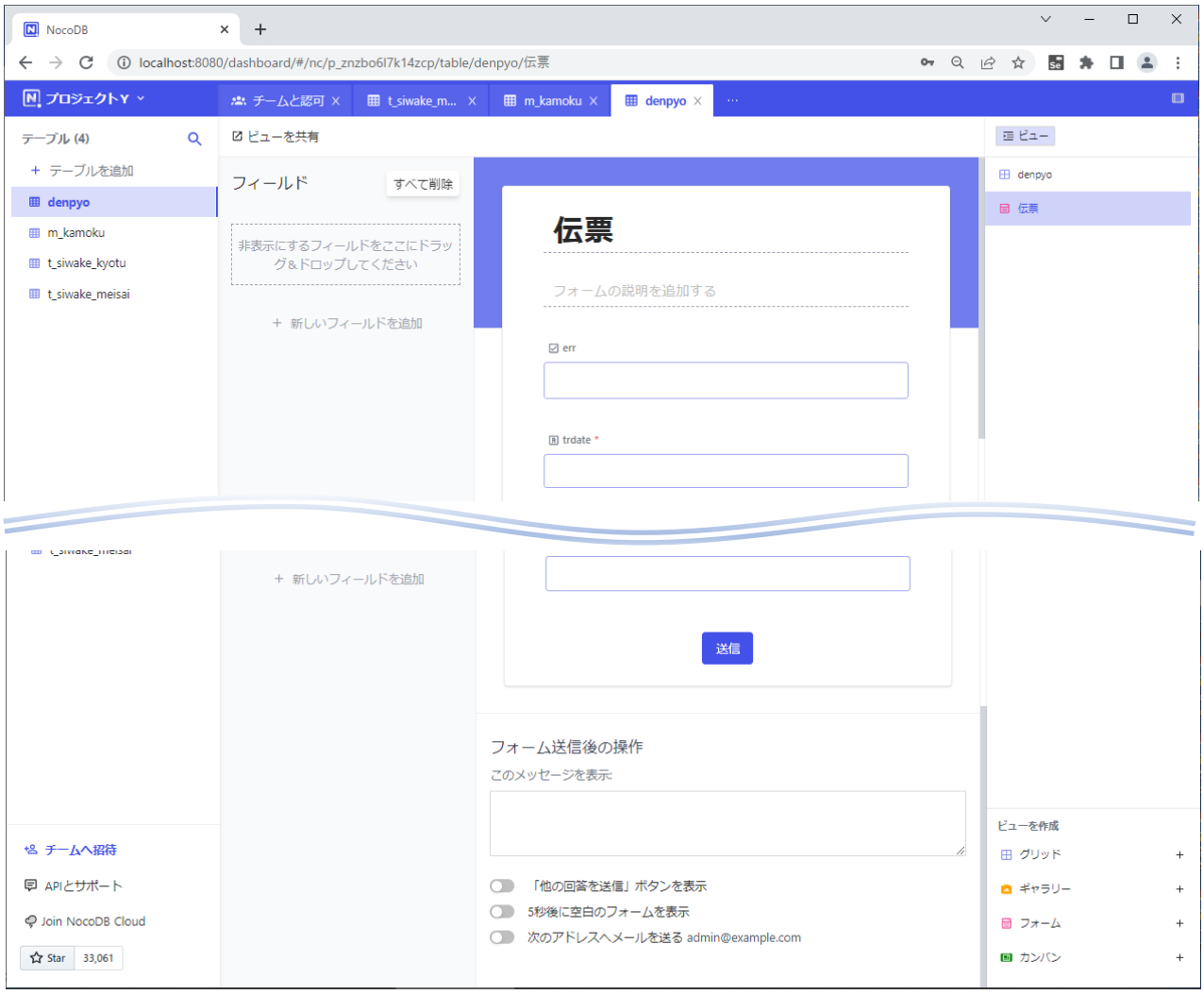
※（nocodb-v0.100.2 の場合）フォーム形式では列を null にすることはできません。null にするにはリスト形式で Clear cell を使います



4.1. 登録フォームの作成

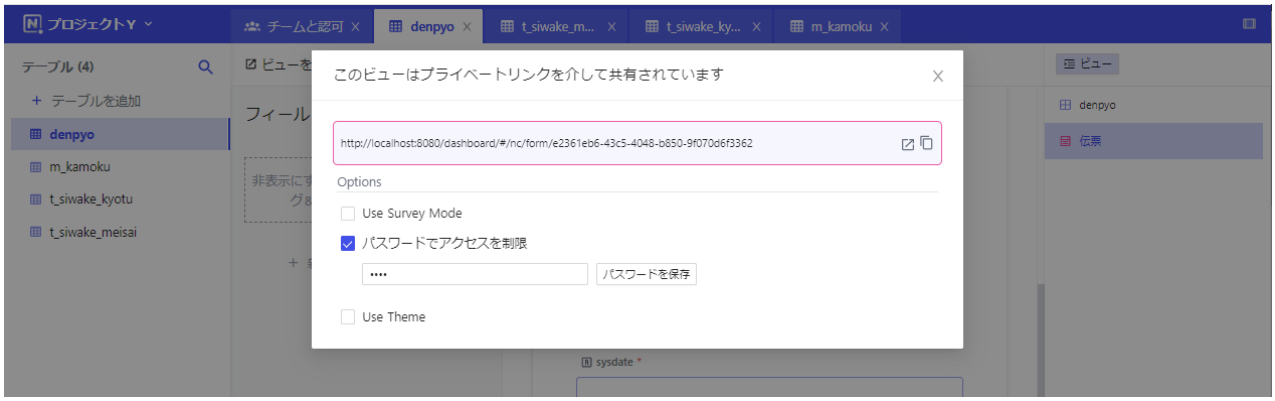
テーブルに関係づけたデータ登録用のフォームが①ビュー、②フォームのリンクから簡単に作ることができます。このフォームではコメントや登録後の動作を付けたり、フォームを表示するためのリンクを送ることができる等グループウェア的な動作を加えることができます。





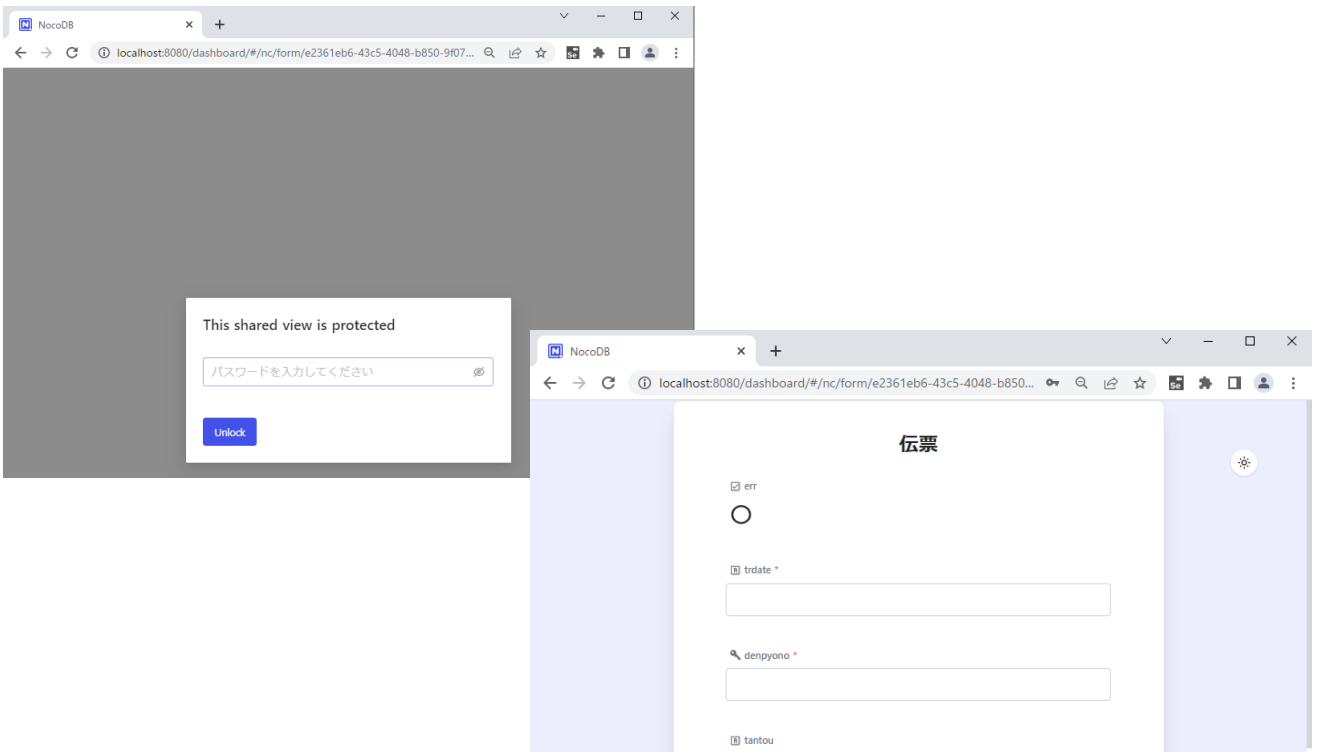
4.2.　登録フォームの送付

フォームの[ビューを共有]を使うとフォームへのリンクが作られ、一時的な利用者に提供できます。



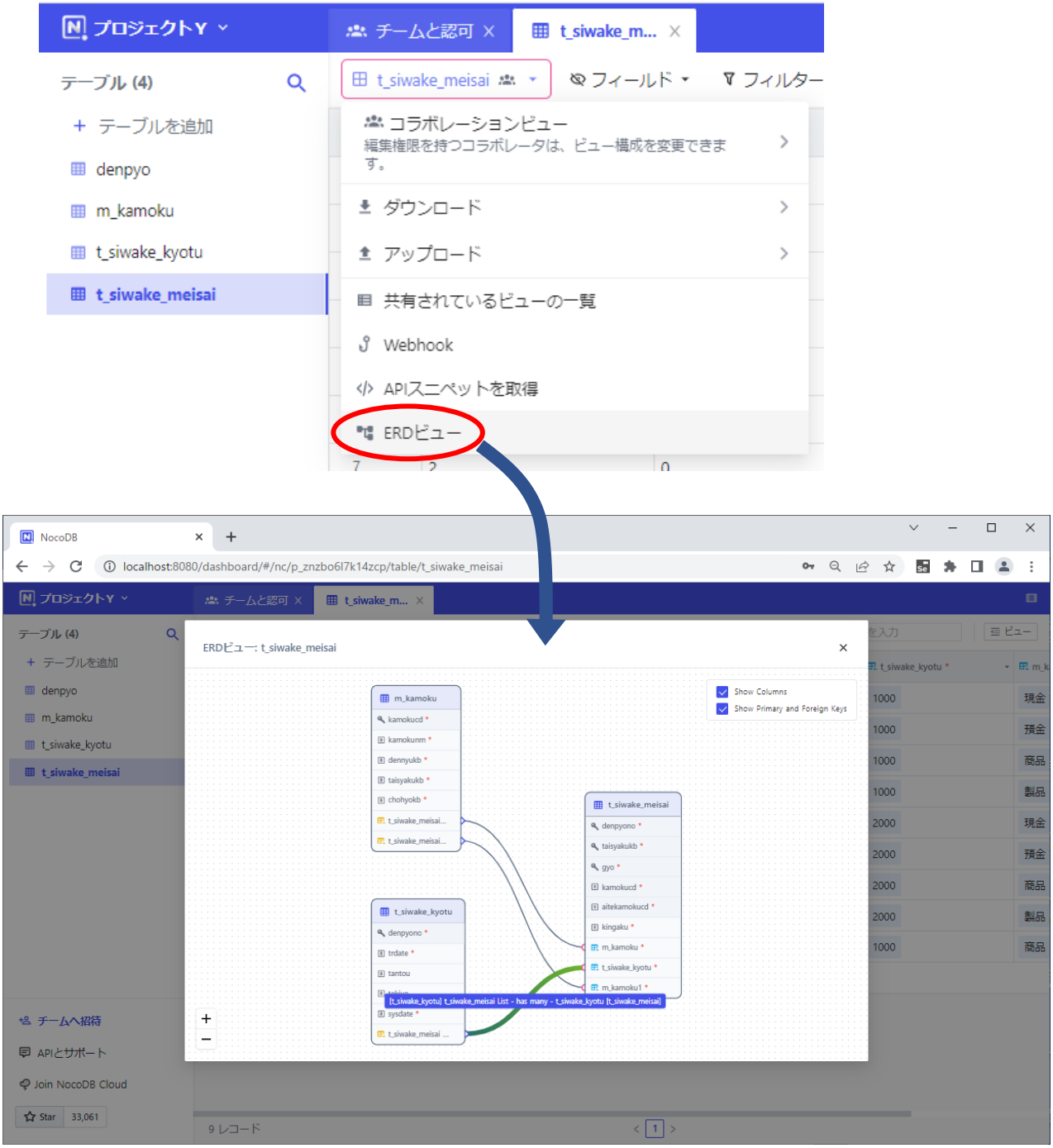
※ モーダル画面に生成されたリンクが表示されるので、これを送ります

●このリンクにアクセスすると、フォームが表示できます（下図はパスワードを設定した例）



4.3. ER 図ビュー

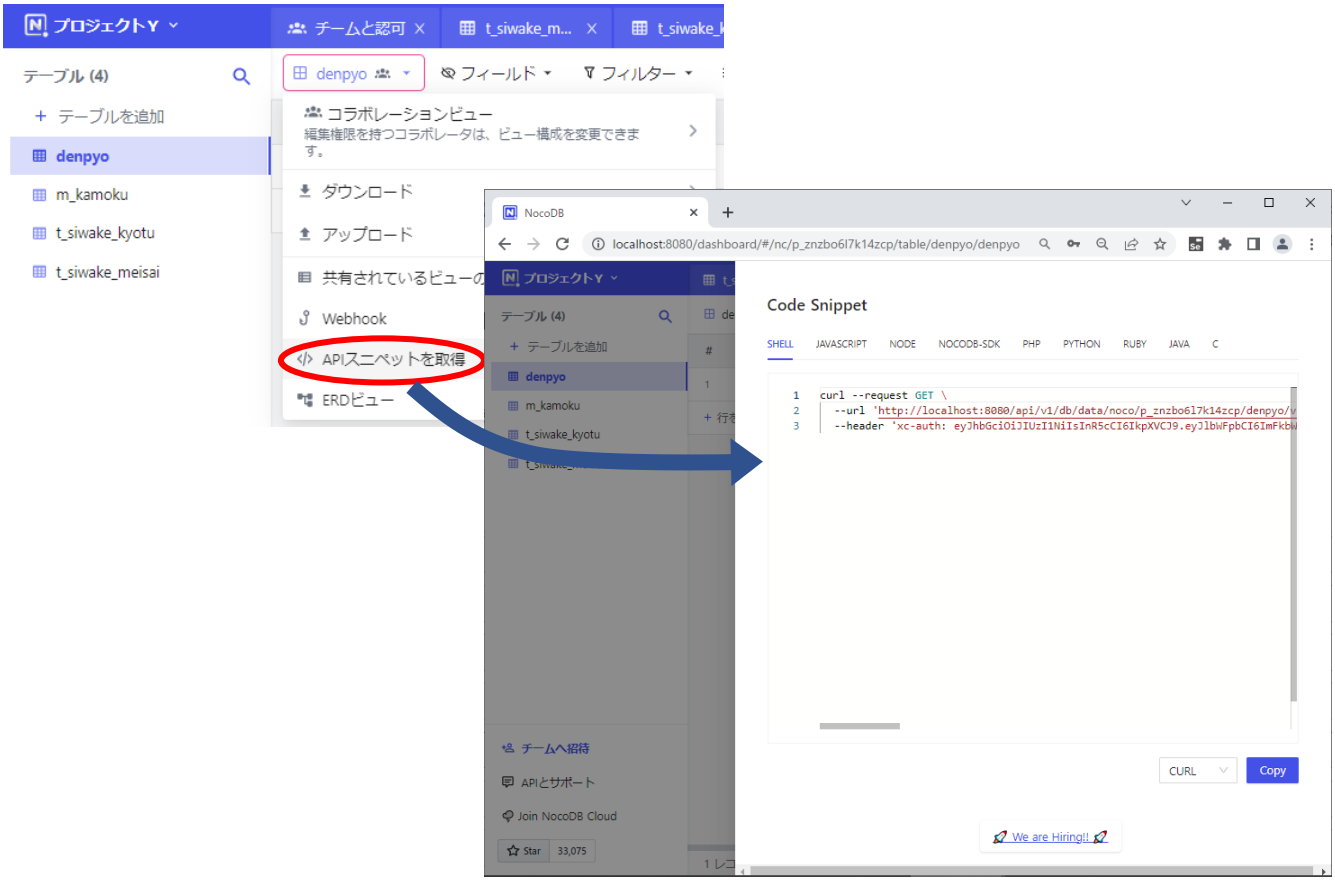
テーブルの論理構造を各テーブルの ERD ビューを選択してみることができます。



※ このＥＲ図では関係線を水色や黄色の列の間に結んでいますが、これは関係を表すために使われる列種類=LinkToAnotherRecord と呼ぶ仮想の列で、テーブルの列として定義されたものではありません

5. REST-API による利用

nocodb には http リクエスト（REST-API）を使ってアクセスできます。REST-API を使う簡単な方法はテーブルアクセスしたいテーブルを選択して[API スニペットを取得]を実行することです。



●Windows のコマンドは、API スニペットの SHELL で表示される内容に以下の変更が必要です

・改行の行端をバックスラッシュ（”¥”）⇒キャレット（“＾”）に変更

・文字列の囲みは、シングルクォーテーション（‘）⇒ダブルクォーテーション（”）に変更

＜コマンド実行例＞ --include オプションはレスポンスのヘッダを表示します（指定は任意）

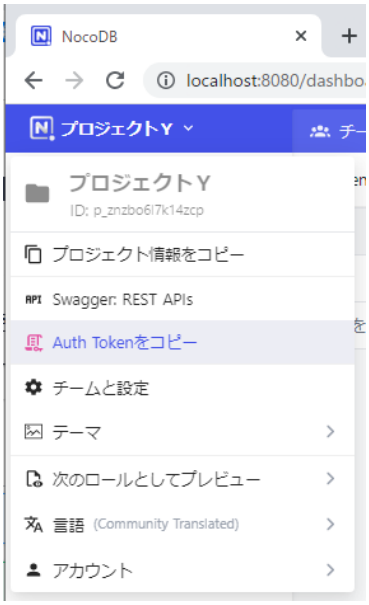
```
curl --request GET --include ^
  --url "http://localhost:8080/api/v1/db/data/noco/p_znzbo6l7k14zcp/denpyo/views/denpyo?offset=0&limit=25&where=" ^
  --header "xc-auth: eyJhbGciOiJIUzI1NiIsInR5cCI6I…途中略…Xl-sR7gIiFSpaGGP4GR_zRE4N_ij-E"

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 335
ETag: W/"14f-HSDLPLeOm5UOzoiLJuaSTkKt9mE"
Date: Wed, 21 Dec 2022 05:27:34 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"list":[{"err":true,"trdate":"20220601","denpyono":"00001","tantou":"1000","karikamokucd":["123456","234
567"],"karikingaku":["1000000","10000000"],"kasikamokucd":null,"kasikingaku":null,"tekiyo":null,"sysdate"
:"20210631","errmessage":"err
test"}],"pageInfo":{"totalRows":1,"page":1,"pageSize":25,"isFirstPage":true,"isLastPage":true}}
```


5.1. API の認証

API を使うには認証が必要で、http ヘッダにトークンを格納して実行します。トークンには以下の 2 種類あります。

① 認証トークン：利用者がログインしたときに生成され、デフォルトで生成後 10 時間有効です

② API トークン：トークンを手動で生成して API 毎に束特しておきます



コマンド実行例のオプション --header xc-auth で指定しているのが認証用の文字列で、プロジェクトの「Auth Token をコピー」でトークンだけ取り出すこともできます

5.2. リソース／API 毎のトークン登録

トークンを生成して Swagger で API に登録します。

（1）トークンの生成

以下の手順でトークンを生成します。

① プロジェクトのメニューから[チームと設定]を選択

② [チームと認可] ⇒ [API トークン管理]タブを選択

③ [トークンを追加]を押下 … トークンが生成され、右側の Copy ボタンで内容を取得できます

（２）トークンの登録

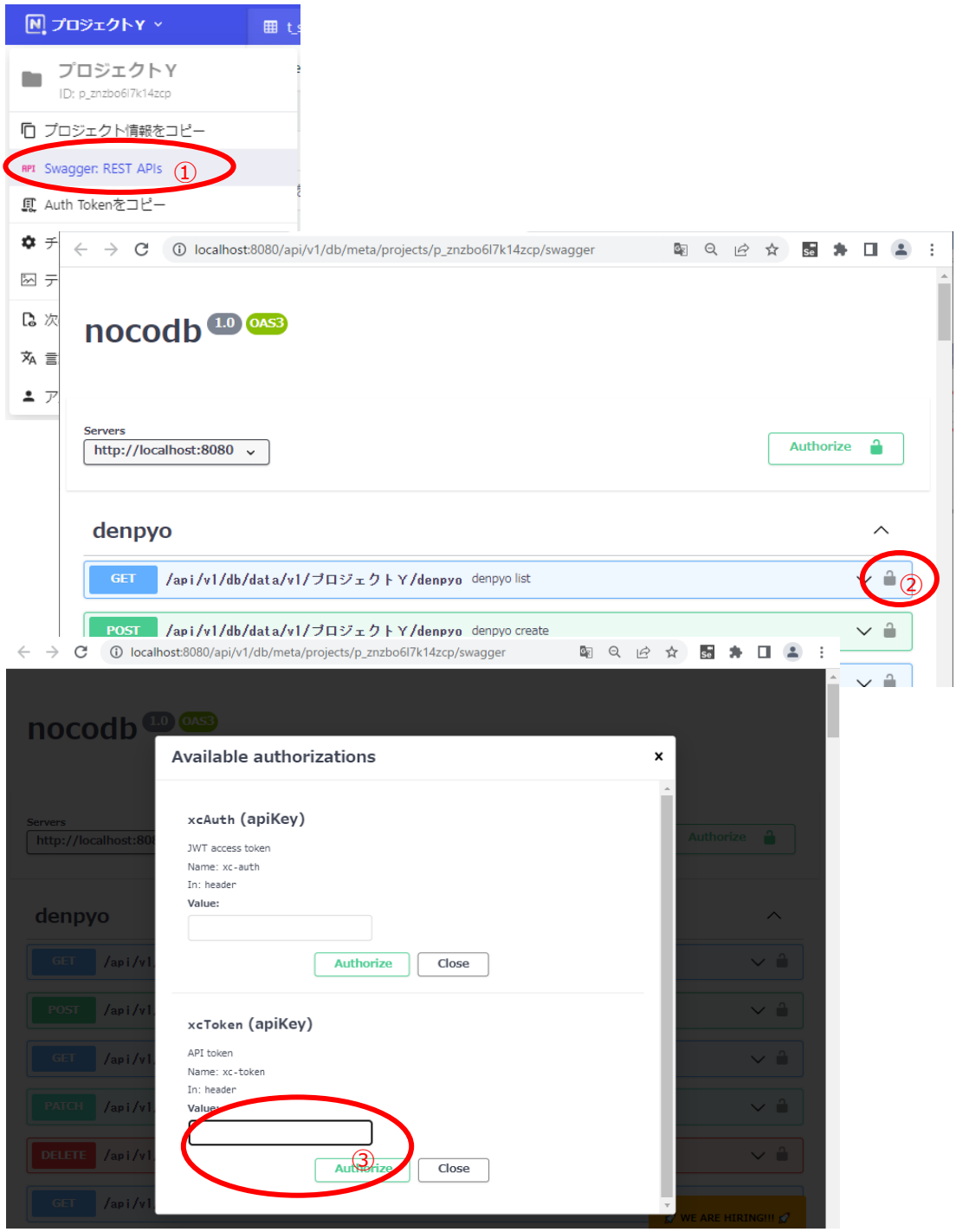
以下の手順でトークンを登録します。

① プロジェクトのメニューから[Swagger: REST APIs]を選択

② 表示された一覧から登録先の API を選び、右端の鍵マークをクリック

③ xcToken(apiKey)の Value:に前に作ったトークンを貼り付け、[Authrize]を押下

… --header xc-token で同トークンが使えるようになります




```
curl --request GET --include ^
  --url "http://localhost:8080/api/v1/db/data/noco/p_znzbo6l7k14zcp/denpyo/views/denpyo?offset=0&limit=25&where=" ^
  --header "xc-token: ASOUrU38kOB5RMN3S81OYTwut8qCzPQzkp7RfIBb"
```

③

5.3. サインインの API（認証トークンの取得）

API を使ってデータを操作するにはトークンが必要ですが、認証トークンは時間制限があり、そもそも同一のトークンを使い続けるのはセキュリティ上のリスクがあります。アプリケーションからデータ操作を行うときは最初に登録済の利用者サインインの API を使って認証トークンを取得し、それを使ってデータ操作の API を呼出します。

類似の API にサインアップがありますが、そちらは利用者の新規登録を行います。

＜API の呼出し例：Windows コマンドプロンプト様式＞

```
curl --request POST --include --url "http://localhost:8080/api/v1/auth/user/signin" ^
  --header "Content-Type: application/json" ^
  --data ^
"{ ^
  ¥"email¥":¥"admin@example.com¥" ^
, ¥"password¥":¥"Password¥" ^
 }"
```

※ password の値は平文でセットします。専用のアカウントを使いアクセス権の制限が必要です

●上記実行例は以下の点が Windows コマンドプロンプト専用になっています

・コマンドオプションを複数行に分けて記述するための改行コードエスケープにキャレット(^)
（キャレットの後ろに改行コード以外の文字=空白等が残っていると機能しません）

・コマンドプロンプトでは文字列の括りに使えるのがダブルコーテーション(“)だけのため、JSON 文字列の一部として渡す(“)を￥記号でエスケープしています

※ 上記の 2 点はコマンドプロンプト以外のツールを使う場合やプログラム化で変更が必要です

＜サインイン API のレスポンス＞

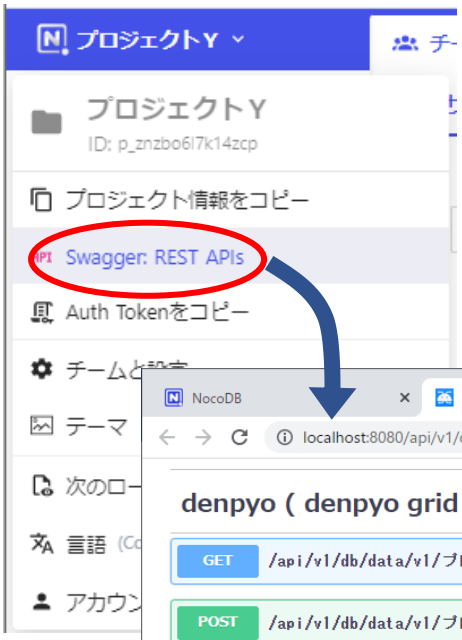
```
HTTP/1.1 200 OK
X-Powered-By: Express
Set-Cookie:
refresh_token=a64140e722229fc84302f2426ff8de412510917b26fc16e1fec2c7c1f01092fefa87f339e5
74f99f; Path=/; Expires=Fri, 30 Dec 2022 02:18:38 GMT; HttpOnly
Content-Type: application/json; charset=utf-8
Content-Length: 432
ETag: W/"1b0-X4YOIObR6lv1nE3cRT/bMGiPLQI"
Date: Fri, 23 Dec 2022 02:18:38 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImFkbWluQGV4YW1wbGUuY29tIiwiZ
mlyc3RuYW1lIjpudWxsLCJsYXN0bmFtZSI6bnVsbCwiaWQiOiJ1c193cnlvb2h3aDBmOHY0MCIsInJvbGVzIjoib
3JnLWxldmVsLWNyZWF0b3Isc3VwZXIiLCJ0b2tlbl92ZXJzaW9uIjoiNjMzNmJjMmIzMWY4MDI3NjZlY2JkYWQzY
jcxMmU1MjU1MjgyYjYxNzMyYTgyYzQ0NDI2N2VkOGU5MTQ5MDNhNjQ4NzMzZDNhMzg5ZGZjNTMiLCJpYXQiOjE2N
zE3NjE5MTgsImV4cCI6MTY3MTc5NzkxOH0.WmUcyKPuscXk7KIIlyzQZKBakZ3mUC01Z93wi-RYNuc"}
```

＜nocodb のアクセスログ出力例＞

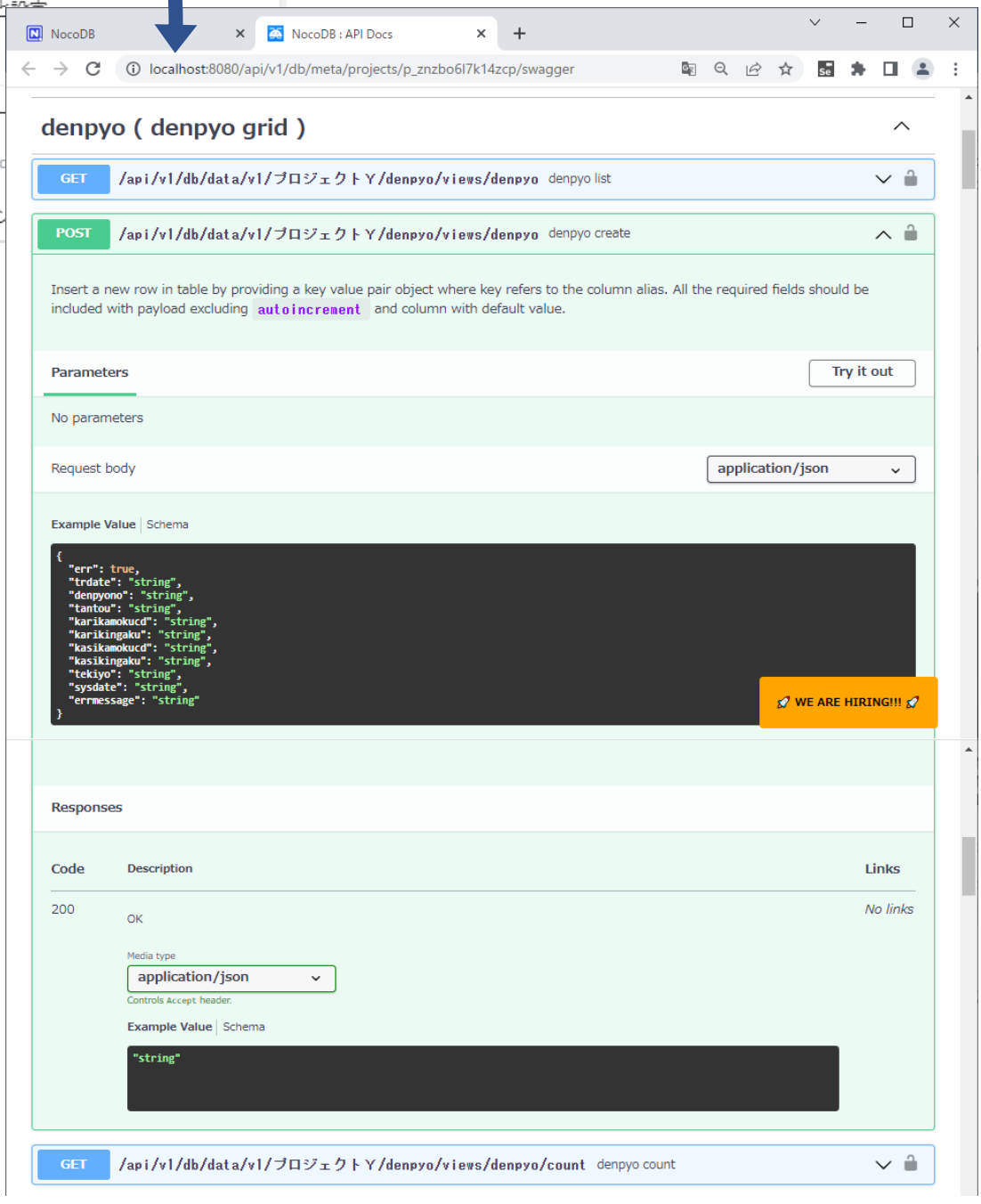
```
POST /api/v1/auth/user/signin 200 432 - 394.502 ms
```


5.4. 行追加の API

追加を含めた API の使い方はプロジェクトのメニューから[Swagger: REST APIs]で確認できます。



●API の受け口になる URL やパラメータはテーブル／機能毎に表示され、Try it out で試行することもできます。
●URL に指定するプロジェクトは名前の下にある ID: で表示されている文字列を使います。



Windows のコマンドプロンプトから実行する場合は以下のようになります。

（xc-auth で指定している値はサインイン API で取得したトークンです）

＜実行例＞

```
C:¥Users¥User>curl --request POST --include --url
"http://localhost:8080/api/v1/db/data/noco/p_znzbo6l7k14zcp/denpyo/" ^
More?   --header "Content-Type: application/json" ^
More?   --header "xc-auth:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImFkbWluQGV4YW1wbGUuY29tIiwiZmlyc3RuYW1lIjpud
WxsLCJsYXN0bmFtZSI6bnVsbCwiaWQiOiJ1c193cnlvb2h3aDBmOHY0MCIsInJvbGVzIjoib3JnLWxldmVsLWNyZWF0b3I
sc3VwZXIiLCJ0b2tlbl92ZXJzaW9uIjoiNjMzNmJjMmIzMWY4MDI3NjZlY2JkYWQzYjcxMmU1MjU1MjgyYjYxNzMyYTgyY
zQ0NDI2N2VkOGU5MTQ5MDNhNjQ4NzMzZDNhMzg5ZGZjNTMiLCJpYXQiOjE2NzE3NjE5MTgsImV4cCI6MTY3MTc5NzkxOH0
.WmUcyKPuscXk7KIIlyzQZKBakZ3mUC01Z93wi-RYNuc" ^
More?   --data ^
More? "{ ^
More?  ¥"err¥":null ^
More? ,¥"trdate¥":¥"20221201¥"^
More? ,¥"denpyono¥":¥"00099¥"^
More? ,¥"tantou¥":¥"1000¥"^
More? ,¥"karikamokucd¥":[¥"123456¥",¥"234567¥"]^
More? ,¥"karikingaku¥":[¥"1000000¥",¥"10000000¥"]^
More? ,¥"kasikamokucd¥":[¥"654321¥",¥"543210¥"]^
More? ,¥"kasikingaku¥":[¥"1000000¥",¥"10000000¥"]^
More? ,¥"tekiyo¥":null^
More? ,¥"sysdate¥":¥"20221223¥"^
More? ,¥"errmessage¥":null^
More? }"
```

＜正常処理のレスポンス＞

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 20
ETag: W/"14-GIQfdR1R2usNIp7L/RcUcNObGsQ"
Date: Fri, 23 Dec 2022 05:06:47 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"denpyono":"00099"}
```

――――　以上　――――

＜エラーがある場合＞

```
HTTP/1.1 400 Bad Request
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 442
ETag: W/"1ba-ncmydlMQsuy1kyuBtfK1JAqpMMw"
Date: Mon, 26 Dec 2022 01:34:21 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"msg":"insert into ¥"denpyo¥" (¥"denpyono¥", ¥"err¥", ¥"errmessage¥", ¥"karikamokucd¥", ¥"karikingaku¥",
¥"kasikamokucd¥", ¥"kasikingaku¥", ¥"sysdate¥", ¥"tantou¥", ¥"tekiyo¥", ¥"trdate¥") values ('00100',
NULL, NULL, '{¥"123456¥",¥"234567¥"}', '{¥"1000000¥",¥"10000000¥"}', NULL, NULL, '20221223', '1000',
NULL, '20221201') returning ¥"denpyono¥" as ¥"denpyono¥" - 重複したキー値は一意性制約¥"pk_denpyo¥"違反と
なります"}
```


6. nocodb（LowCode、NoCode ツール）の用途

nocodb は「フォームのカスタマイズをできるようにする」、「モバイルに対応する」といった開発計画がが出ていますが、今回のバージョンでは業務システム向けに使うには以下の機能が足りません。

① 入力データの検証機能が RDB のテーブルに設定した制約条件のみ

② 自動生成されたフォームは DB の列と 1 ： 1 で項目の合成や加工ができず、項目にコメントを付ける等の装飾もできない

以上のように DB の更新（CRUD）に関しては汎用的な RDB 編集ツールと機能的には大差ありませんが、REST API が使えるのでフォームやデータバリデーションは別のアプリ（React や Vue、Excel/VBA 等）を使ったリッチクライアント方式も可能です（但し、React 等を使うのであれば自身で DB アクセスが可能なので情報共有が必要なければ nocodb も必要ありません）。

入力データのチェックに関しては、例えば注文データに対する在庫確認のような複雑なチェックは複数テーブル／複数項目の操作や計算用のコードが必要になり、ツールだけで実現するのは困難ですが、逆に言えば複雑でないチェックはツールで十分です（1 画面内、1 テーブル内の項目間チェックは RDB の制約条件で事足ります）。また、複雑なチェックは頻発するものでもありません。

いずれにしろコードを書くということはその開発から保守のライフサイクル全体に亘って世話をしていくという手間がかかります（ビジネスに繋がるとも言えますが）。LowCode/NoCode がオープンソースの世界で盛り上がっていてその先の AI を使った設計という話さえ聞こえている現在、少なくとも 30 台以前の人はコードを書くのと同程度に書かないで済ませる研究もしておくべきです（年金が必要になる頃に支給額も減ってくるでしょうし...）。

以上

