
# 春のにゃん祭り2025

にゃん正式お披露目の会！

にゃんくるプロジェクト
羽生 章洋

# 本日の内容

- にゃんくるプロジェクトについて
- 各プロダクトの紹介
- 事例紹介
- 今後のロードマップなど

# にゃんくるプロジェクトについて

# にゃんくるプロジェクトとは

- 公式サイトできました！

  **https://nyanql.org/**

  - スターターキットがダウンロードできます！
    - 今はそれしかないｗ
  - **NyanPUIで動いてます！**

- プロジェクトのメンバーは

  - 羽生 章洋：リードビジョナリー
    - にゃんくるプロジェクト総責任者
    - プロダクト仕様の最終決定者
  - 宮川 貴子：エースプログラマー
    - すべてのコードの責任者

  の２人です。しばらくはこの体制で行きます

# 問題意識と見据えるビジョン

- 少子高齢社会に立ち向かう！
  - 人材不足・人手不足
  - 需要／市場の縮小
- とりあえずターゲットのイメージは、ここに情報収集に行くようなところ

  https://www.japan-it.jp/spring/ja-jp.html#/
- この需要に応えるために、
  易しくて小回りの効くシステム開発を実現することを目指す
  - 最先端でも最高でもなく、最適を提供する基盤として
  - 部門レベル、現場レベルの中小規模のシステム
  - 中小～中堅企業の基幹系くらいまでのイメージ
- 高校における「情報Ｉ」の必修化
  - 2025年3月卒業のこの春から社会人になった高卒の子たちから必修化された
  - 今年大学に入った子たちが社会人になる4年後には
    新卒はみんな情報Ｉを学んでる状態である

# ソリューションとしての、にゃんくる各プロダクト

- **生成AI**が存在する時代という前提！
  - コードは生成AIに出力させる
  - そのインプットとしての要件定義からの**コードの生成しやすさ**
- **開発生産性から、安心して使い続けられる継続性へのシフト**
  - 生成されたコードを使い続けるための保守性が重視される時代と考える
- **枯れた技術の水平思考**
  - 最先端ではなく、最初からヘルシーなレガシーとしてリリースする
- **情報Ｉのリテラシーの範疇で！**
  - 授業で知るのは、**SQL、HTML&CSS、Javascript** ！

# 各プロダクト紹介

# 共通のコアコンセプト

- **APIファースト！**
  - api.json で、APIを定義する
  - APIに対して、コードを記述したファイルをマッピングする
- Web時代のコマンド＆パイプ ＝ JSON on HTTP
  - UNIX的な発想で
  - 小さなブロックを組み合わせて、大きく育てていく
- **API一覧出ます！**
  - /nyan で出ます
- 環境構築に悩まない
  - 基本はバイナリーを置いて起動するだけ
  - config.json で起動時の設定を定義する
  - コードもファイルを置くだけ
- **コードこそ仕様！**

# NyanQL

# NyanQL（にゃんくる）

- JSON↔SQLマッパー
- **APIを呼び出すと、**
  **紐づけられたSQLが実行されて、実行結果をJSONで出力します**

- **SQLファースト！**
  - まず単体でちゃんと動作するSQLを書く
  - api.json にAPI定義を書く
  - API定義で、APIとSQLファイルを紐づける

**api.json**

```
{
   "list": {
      "sql": ["./sql/sqlite/list.sql"],
      "description": "当月の一覧を表示します。"
   },
}
```

# NyanQL（にゃんくる）

- **2way-SQL**
  - SQL内のコメントにパラメータを書ける（S2Daoリスペクト）
  - APIのリクエストパラメータを自動的にSQLのパラメータにマッピングする
  - このパラメータを拾って /nyan によるAPI情報にパラメータ情報が出ます
- 選択リストのAS句が出力するJSONの項目名になります

**SQLサンプル**

```
SELECT
   count(id) AS this_days_count
FROM
   stamps
WHERE
   date = /*date*/'2025-02-15';
```

# NyanQL（にゃんくる）

- パラメータ
  - リクエストパラメータのSQLへのマッピングは prepareStatementで行うため、SQLインジェクションは起こらない。
- 複雑なパラメータチェック
  - APIに対するリクエストパラメータのバリデーションチェックをJSで書ける
  - api.json の check 定義で、JSファイルを割り当てる
  - **受信したリクエストパラメータは、nyanAllParams に自動的に格納される**
    - **これは、NyanPUI, Nyan8 も同じ**
  - 受け付けるパラメータを nyanAcceptParams で定義することもできる

**JSサンプル**

```
// "date" が存在しない場合はエラーとする
  if (!nyanAllParams.date) {
    return JSON.stringify(
      { success: false, status: 400, error: { message: "date パラメータが必要です" } });
  }
  // 上記チェックが成功した場合は成功レスポンスを返す
  return JSON.stringify({ success: true, status: 200 });
```

# 時代はDXだ！

- DXというキーワードは、ざっくり2つに分類される
  - データ活用系＝DSS：データ検索系
    - データ分析とかデータサイエンスとかデータ駆動経営とかBIとか…
  - 業務支援系＝OLTP：データ発生／登録系
    - モバイルで現場とか、IoTとか、ネットショップとか、基幹系の入力とか…
- **データ活用といえばExcel！**
  - ということで、Excelをテーブルの代わりにしてSELECTできるようにする
  - **DuckDB**を対応DBのひとつとする

**DuckDB経由でExcelから**
**データを取得するSQLのサンプル**

```
SELECT
 *
FROM
 st_read('./excel/test.xlsx',layer = 'Sheet1',
                 open_options = ['HEADERS=FORCE', 'FIELD_TYPES=AUTO']
         )
;
```

但し、現時点では初回に
INSTALL spatial;
LOAD spatial;
を行う必要があるため、
利用時にインターネットに繋がっている必要がある

# トランザクション

- 業務支援系＝OLTP
- OLTPといえばトランザクション

- **複数のテーブルを同時に操作する**
  - 例えば、入庫テーブルに追加 & 在庫テーブルを更新 のような場合

  - api.json の sql 定義の配列内で、複数のSQLファイルを指定する
  - 全部のSQLをまとめて、ひとつのトランザクションとして処理する

**api.json**

```
{
   "addItem": {
      "sql": [ "./sql/insertNyuko.sql", "./sql/updateZaiko.sql" ],
      "description": "商品を1件入庫します"
   },
}
```

# トランザクションスクリプト

- 複雑な一連の処理をひとつのトランザクションとしてまとめる
  - 例えば、注文伝票を1件追加
    …注文ヘッダに1件insert & 注文明細に複数件insert
  - JSONで1件の注文伝票として受け取って、明細部をループで分解したい

- JSファイルでトランザクション処理を書く
  - チェックも使える
  - JSファイル内では、nyanRunSQL でSQLファイルを呼び出す
    - あくまでもJSはSQLファイルを呼び出すだけ
    - SQL自体は、SQLファイルを書く

**api.json**

```
{
   "scriptTest": {
      "check": "./js/check.js",
      "script": "./js/script.js",
      "description": "トランザクションスクリプトのサンプルです"
   }
}
```

# NyanPUI

# NyanPUI（にゃんぷい）

- お手軽HTMLサーバ

- **APIを呼び出すと、**
  **紐づけられたHTMLを出力します**

- **HTMLファースト！**

  - まず単体でちゃんと動作するHTMLを書く
  - api.json にAPI定義を書く
  - API定義で、APIとHTMLファイルを紐づける

**api.json**

```
{
   "test": {
      "html": "./html/test.html",
      "description": "テストページ" }
}
```

# NyanPUI（にゃんぷい）

- 動的HTMLを生成する（SSR：サーバサイドレンダリング）

- **APIを呼び出すと、**
  **紐づけられたJSを実行して、実行結果をHTMLを出力します**

**api.json**

```
{
   "test": {
      "script": "./js/test.js",
      "html": "./html/test.html",
      "description": "テストページ" }
}
```

# サーバサイドJavascriptによる動的HTMLの実現

- **書けるJavascriptは、ECMAscript5.1相当**
  - **NyanQL, Nyan8 も同じ**
  - 利用しているGojaの仕様
  - 枯れていること、安心して使えること
- **nyanHtmlCode**
  - これはNyanPUIのみ
  - api.json で定義したHTMLファイルは、変数nyanHtmlCode に格納される
  - コード内で、nyanHtmlCode を自由に加工できる

# 他のサーバからデータを取得する

- 例えば、NyanQLの実行結果のJSONを取得して
  HTMLに埋め込むなど

- **nyanGetAPI** または **nyanJsonAPI**
  - ES5.1でAjaxの記述をするのは面倒くさいので、独自で用意してある
  - サンプルは、readme.md を参照のこと
  - **NyanQL, Nyan8 のJSでも使える**

# テンプレートエンジンにゃんぷれ（NyanPlate）

- 取得したJSONをHTMLにマッピングするのは面倒くさい
- そこで、にゃんぷれ（NyanPlate）
  - **動的文字列置換**：要素内のテキスト（またはコメント内のテキスト）を **data-nyanString="key"** 属性に基づいて JSON データの該当キーの値に置き換えます。
  - **ループ処理**：**data-nyanLoop="key"** 属性を持つ要素内のテンプレート部分を、JSON 配列の各アイテムごとに展開します。 各ループ項目では、個別のデータコンテキストに基づいて文字列、クラス、スタイルの置換が行われます。
  - **クラス・スタイルの動的変更**：**data-nyanClass="key"** および data-nyanStyle="key" 属性を使い、要素の class や style 属性を JSON データの値で置換します。 ループ内のテンプレートとグローバルな置換処理の干渉を防ぐため、ループ展開後に不要な属性を削除する工夫を施しています。
- **これでOK！**

  **const マッピングされたHTML = nyanPlate(JSONデータ);**

# Nyan8

# Nyan8（にゃんぱち）

- JSON↔JSマッパー
- **APIを呼び出すと、**
  **紐づけられたJSが実行されて、実行結果をJSONで出力します**
- **JSファースト！**
  - api.json にAPI定義を書く
  - API定義で、APIとJSファイルを紐づける
- **M2M（マシンtoマシン）のAPIジャンクションサーバ**
  - 他のにゃんとのコネクション
  - 生成AIとのコネクション
  - IoT／エッジ（センサー／アクチュエータ）とのコネクション
  - 四方八方の意味で「８（はち）」

# 何でもAPI化しちゃうぞ！

- APIの入出力定義：パラメータと出力項目の情報定義
  - **nyanAcceptParams** で受け取るべきパラメータの定義を記述する
  - **nyanOutputColumns** で出力するべき項目の定義を記述する
  - この定義が、/nyan によるAPI情報に出力されます
  - 今の時点では、API情報のためだけ
  - **この2つは、NyanQLのJSでも使えます。**
  - nyanAcceptParams はNyanPUIでも使えるようになる予定です（近日）
  - 先々は、これらを使った事前条件／事後条件のチェックも書けるようにするつもり（ちょっと先かも）

# ChatGPTのAPIを呼び出す例（古いけど）

```
function main() {
    let text = nyanAllParams.text

    // コマンドラインで、ChatGPTを呼び出すシェルスクリプトを実行する
    result = nyanHostExec("./js/chatgpt_api.sh " + text)

    // nyanHostExecの戻り値の中から標準出力の部分を抜き出す＝ChatGPTのレスポンス
    response = JSON.parse(result.Stdout)

    // ChatGPTのレスポンスの中から、回答のコンテキストを抜き出す
    msg = response.choices[0].message.content

    // 結果を出力する
    return JSON.stringify({
        "success": true,
        "status": 200,
        "data": msg
    });

}
main();
```

**nyanHostExec も、NyanQL, NyanPUI で使える**

# ChatGPTのAPIを呼び出す例（古いけど）

```
function main() {

    // パラメーターからユーザからのプロンプトを取得する
    let text = nyanAllParams.text

    // 環境変数などからAPIキーを取得することを推奨
    let chatGptApiKey = "Bearer " + 実際のキー

    // OpenAIのエンドポイントURLを定義
    var endPoint =
"https://api.openai.com/v1/chat/completions";
    var modelName = "gpt-4o"; // 最新のモデル

    // チャットのメッセージを定義
    var messages = [
        {
            role: "user",
            content: text
        }
    ];

    // ヘッダー情報をJSON文字列で渡す例
    let headers = JSON.stringify({
        "Authorization": chatGptApiKey
    });

// リクエストデータの作成
    var requestData = JSON.stringify({
        model: modelName,
        messages: messages,
        max_tokens: 700 // レスポンスのトークン最大数
    });

    // リクエスト送信
    let response = JSON.parse( nyanJsonAPI(endPoint, requestData, "", "",
headers) );

    // レスポンス処理
    let msg = response.choices[0].message.content

    // 結果を出力する
    return JSON.stringify({
        "success": true,
        "status": 200,
        "data": msg
    });
}
main();
```

# ビジネスルールを統一する

- そもそも３にゃんとも、共通処理を集めたJSファイルを作れる
  - config.json に…

```
"javascript_include": [
                        "./javascript/base.js",
                        "./javascript/lib/nyanPlate.js"
    ],
```

- 但し、これだと他のホストと共有できない

- **外部APIの呼び出しは、**
  **nyanGetAPI** または **nyanJsonAPI**
  - **NyanQL, Nyan8 のJSでも使える**

- **別に外部でなくても良い＝同一ホストのAPIも呼び出せる！**

# 共通機能

# すべてのにゃんのJSで使える便利グッズ

- パラメータ関係
  - nyanAllParams
  - nyanAcceptParams
  - nyanOutputColumns (NyanPUI除く)
- 外部API呼び出し
  - nyanGetAPI
  - nyanJsonAPI
- Cookie関係
  - nyanSetCookie
  - nyanGetCookie
- localStorage系
  - nyanSetItem
  - nyanGetItem
- cosole.log
- OS系
  - nyanGetFile
  - nyanHostExec

# 水平分散×疎結合を目指して

- **WebSocketによるサーバプッシュの実現**
  - すでに使えます！
- **JSON-RPC2.0によるAPI呼び出しのサポート**
  - こちらも、すでに使えます
  - 一応、MCPへの準備のつもり
  - ただまぁ、様子を見ながら徐々に
    - 冒頭のターゲットの話につながる…生成AIの導入以前の話だったりするので

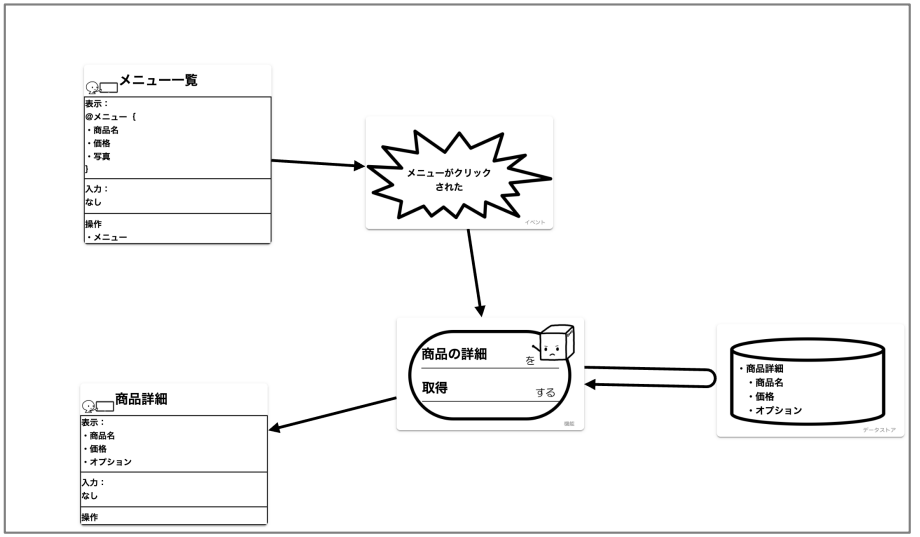
# 事例紹介

# 事例といっても

- デジマジなんですけどね
  - React + Nyan8 + NyanQL で開発中
  - サーバサイドをFirebaseから置き換え
  - RDBMSはPostgreSQL
  - エンティティはちゃんと正規化してる
  - が！　項目は可変するので、
    基本的にJSONB型の列にJSONをそのままぶち込んでいる
- とりあえずAPIファーストでDBはJSONストアとして開発を進めて、落ち着いてからAPIを維持したまま、DB設計のリファクタリングをするのは全然アリだと感じている
  - 多分、システム開発のボトルネックが、DBの項目決定だと思うので

# デジマジで書いた1枚のIFDAM図から…

- この画像をo3に放り込んで、
  コード生成までやらせたんだけど…
- o3すげぇ！
- Nyanの仕様を理解させたら、
  ちゃんとNyanらしいコードを生成した！
  - api.json まで吐いた！
- 修正は以下のみで動いた！
  - api.json に書いてるJSファイルのフォルダ名
    （俺が初期値から変えてたからｗ）
  - API呼び出しの認証User/Pass
  - ポート番号
- 引き継ぎ用の保守ドキュメントも作ってくれて
- 仕様変更にも対応してくれた！
- にゃんぷれのバグと仕様漏れに気づくきっかけも見つけてくれたｗ



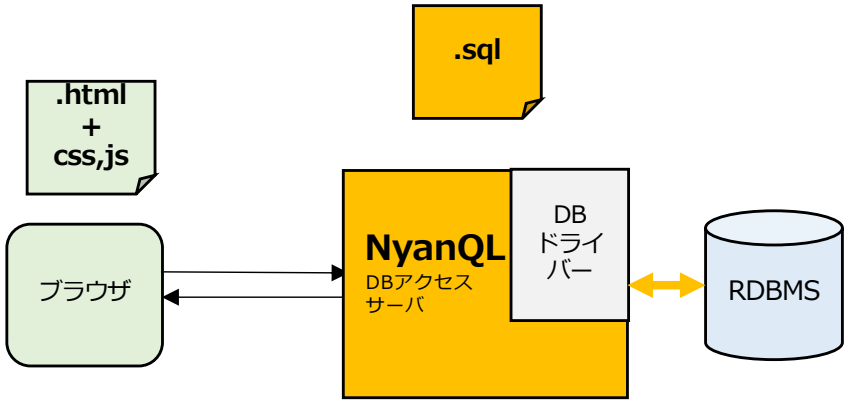

- **o3がコードを読んで理解してくれている！**
  - **どのにゃんもmain.goの１ファイルのみ**
  - **一番大きいNyanQLでも現時点で2kライン**
- **にゃんの利用者が少なくても問題ない！**
- **サンプルとコーディングガイドを充実させて
  それも合わせて理解させれば、かなり良い感じに生成するんじゃないかな**

# 構成パターンあれこれ

昨年の、にゃんのから騒ぎ2024 からの再掲

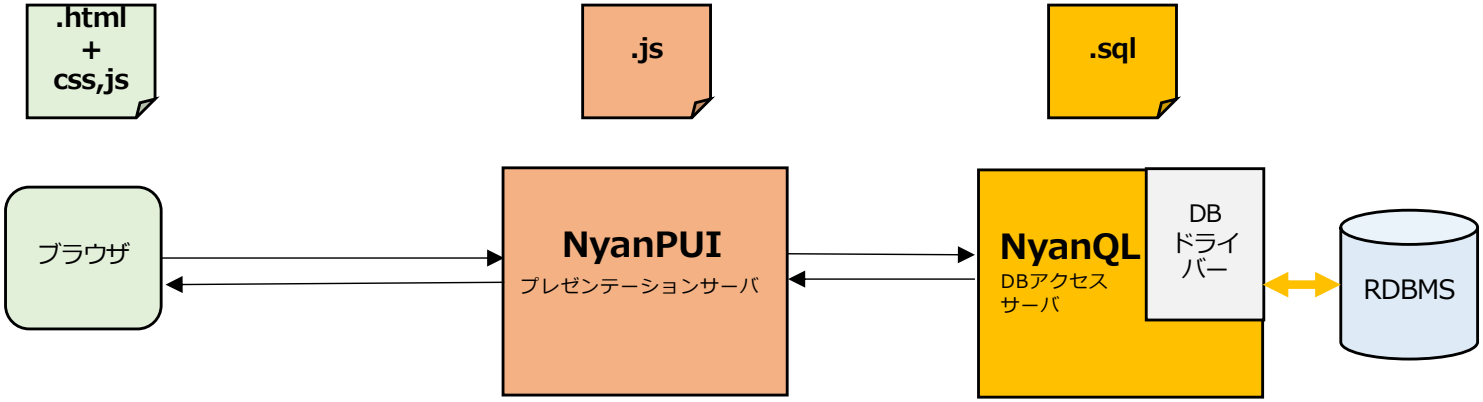
# いろいろな構成パターン

■ **これぞ王道！　いいじゃんこれで！なHTML＋DBのみパターン**
…jQueryとかでいいじゃん

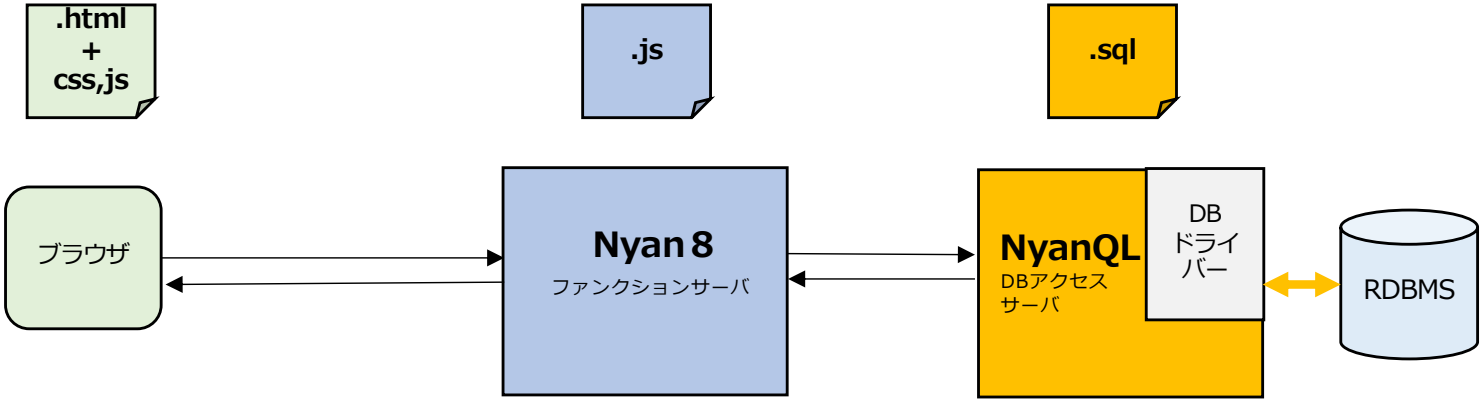


# いろいろな構成パターン

## ■NyanPUIをPHP的に使うSSRなパターン

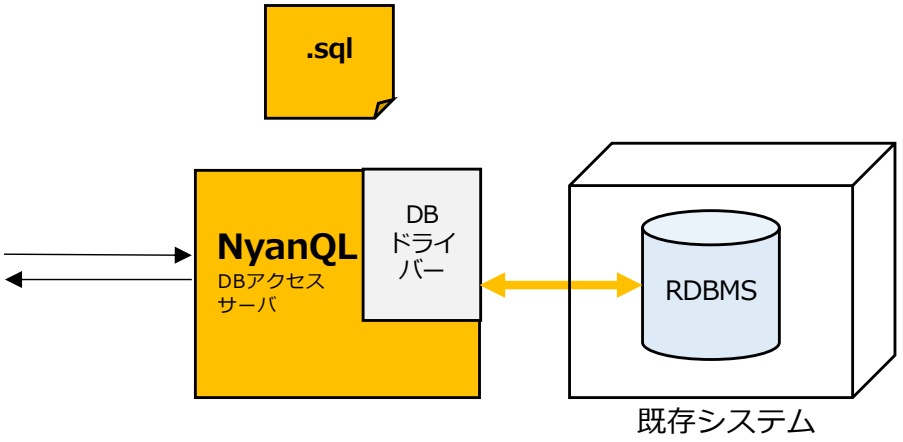



## ■SPA（Reactとか）とJSONなAPIサーバのパターン

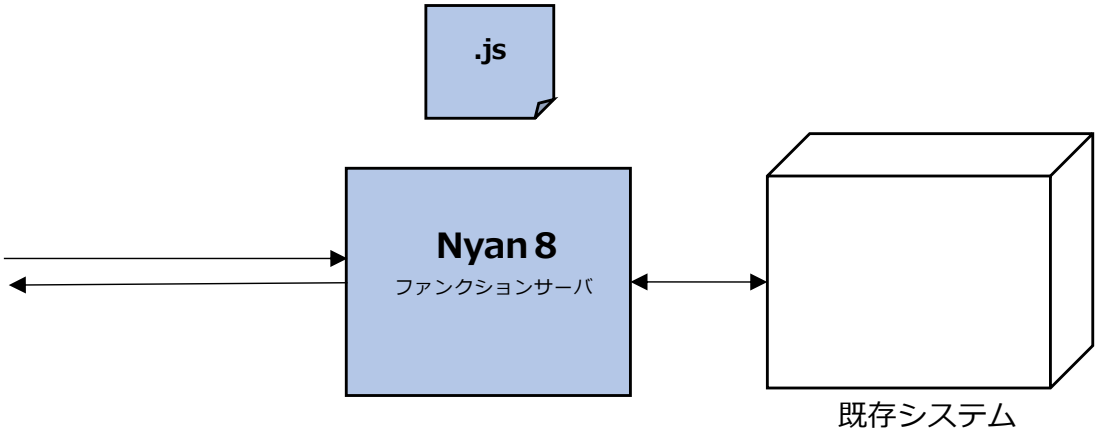


# いろいろな構成パターン

## ■既存システムのDBをAPIサービス化して使いやすくするパターン

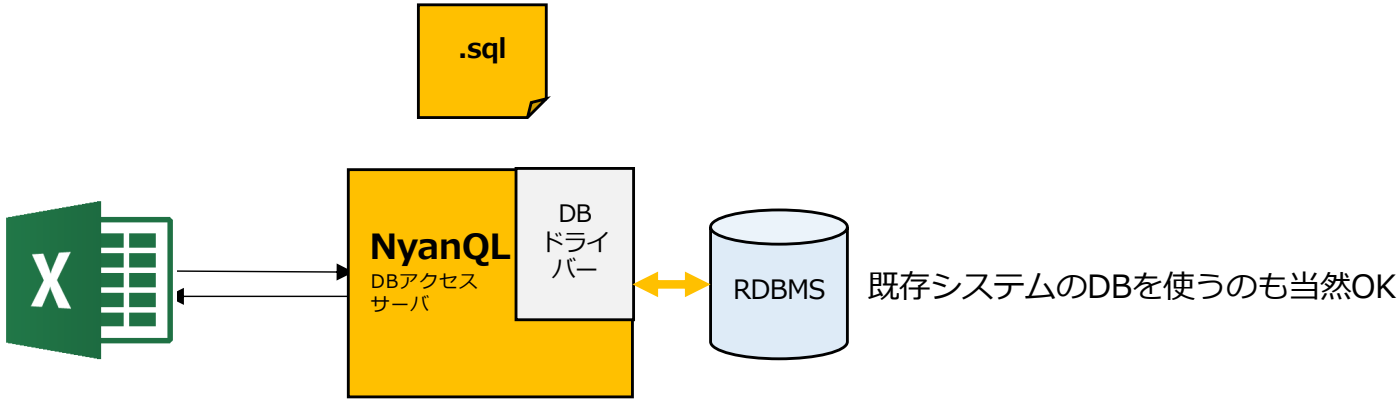



## ■既存システムをどうにかAPIサービス化して使いやすくするパターン

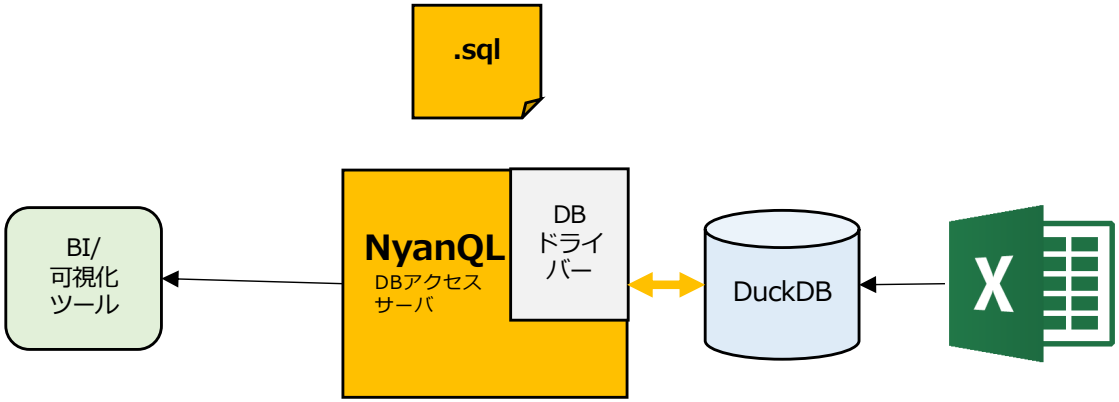


# いろいろな構成パターン

## ■Excelでデータ活用してDXだ！パターン

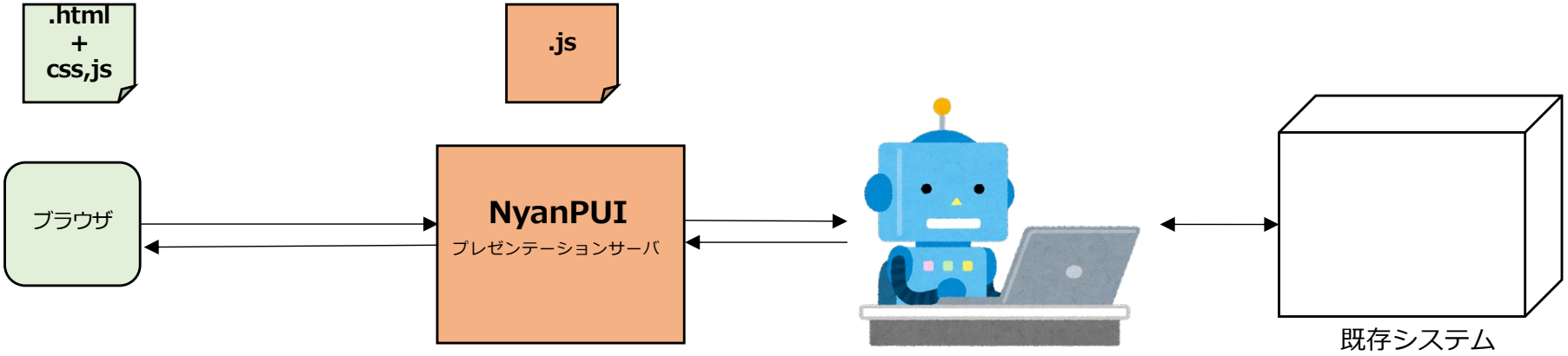



## ■Excel"の"データを活用してDXだ！パターン

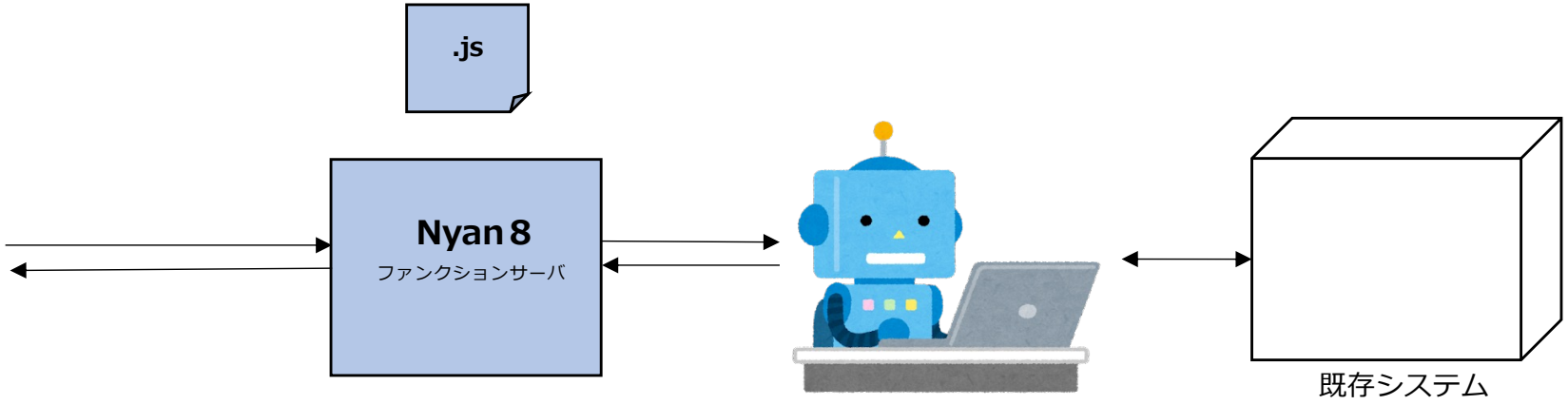


# いろいろな構成パターン

## ■既存システムの画面操作RPAをブラウザ越し利用にサービス化するパターン

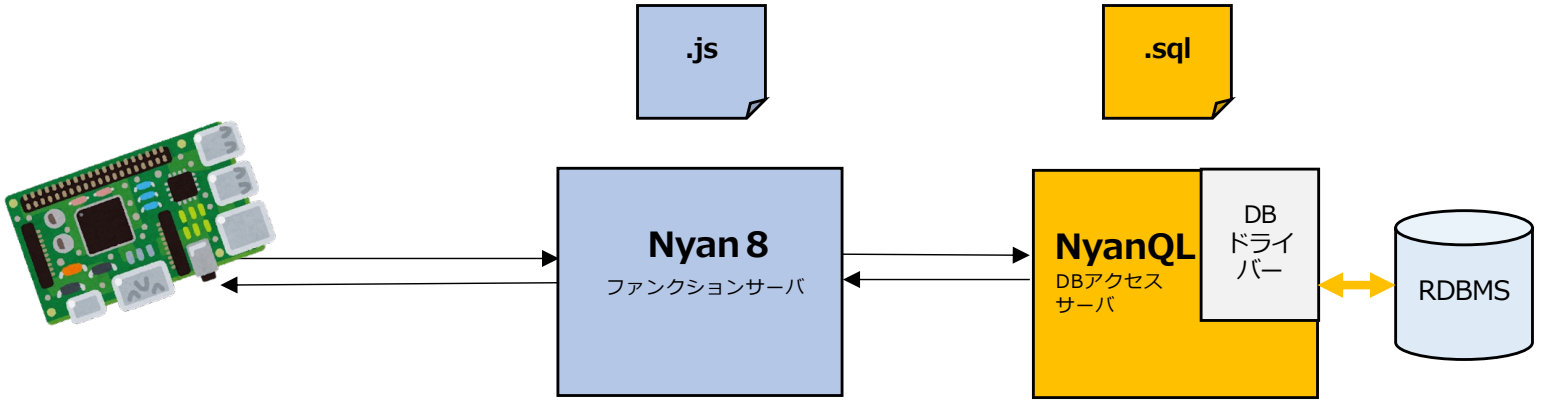



## ■既存システムの画面操作RPAをJSONでAPIサービス化するパターン

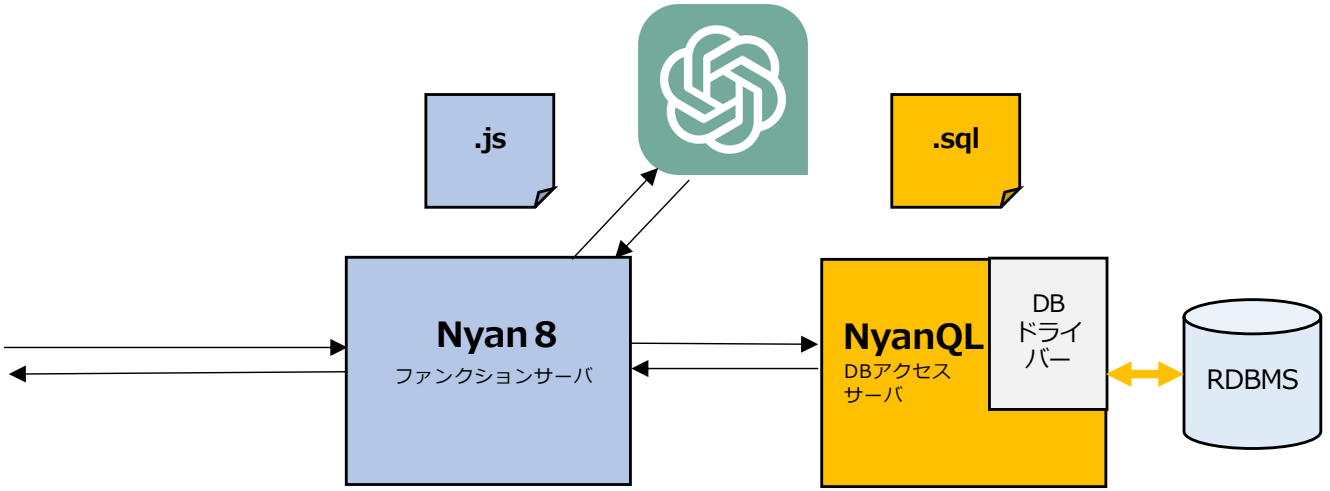


# いろいろな構成パターン

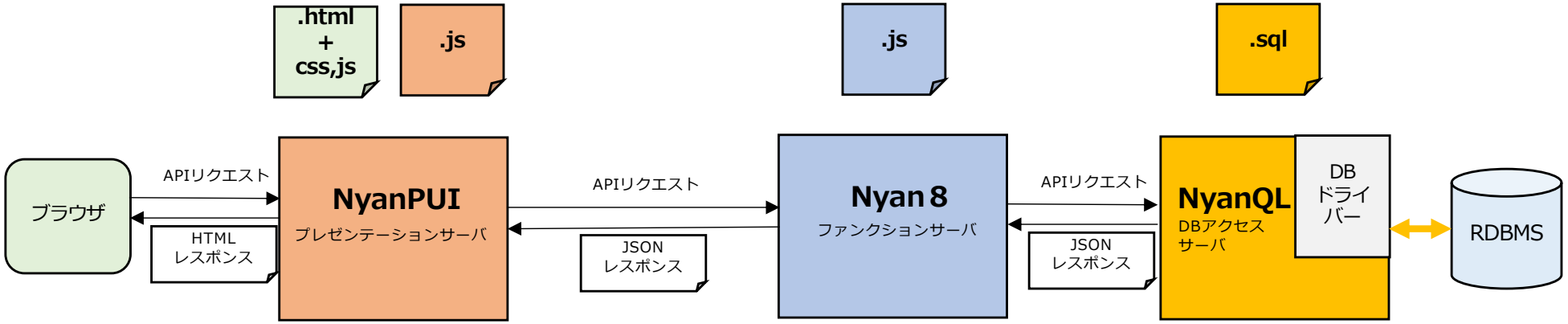
## ■センサーだ！エッジコンピューティングだ！IoTだ！パターン



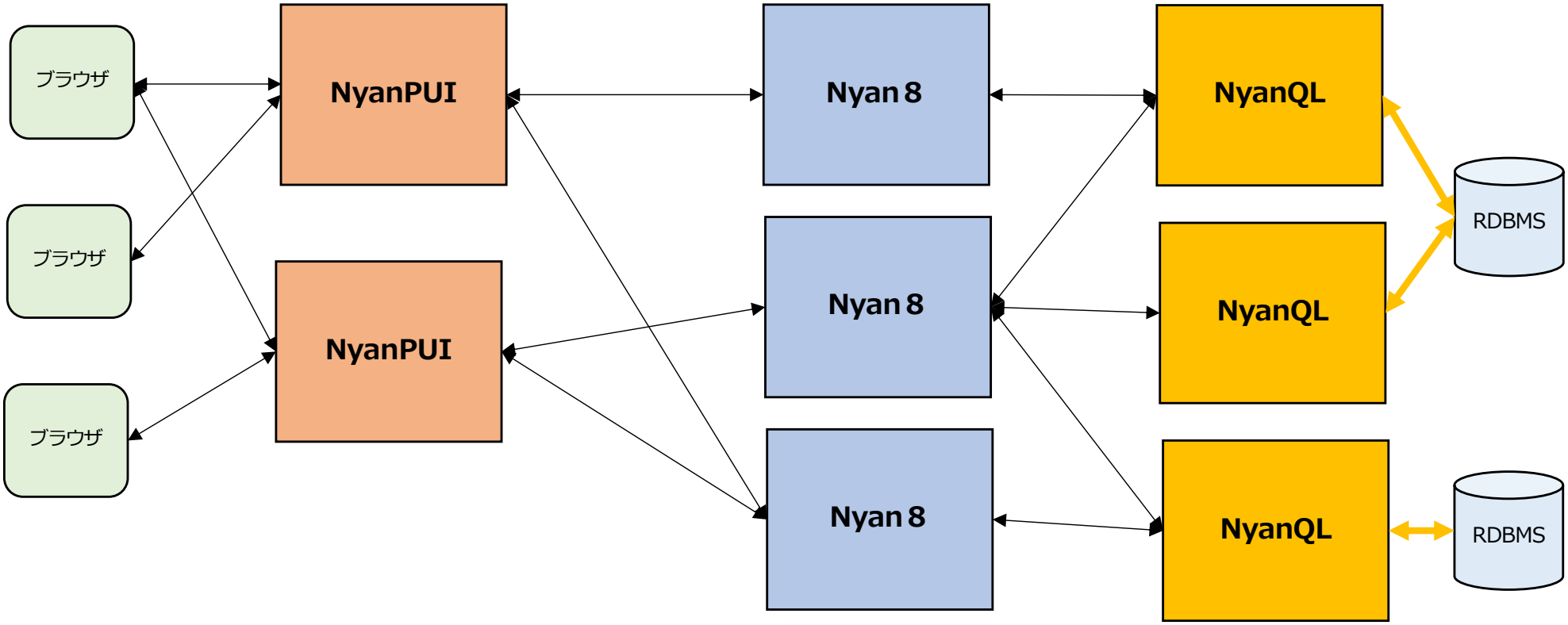

## ■生成AIを呼び出すぜ！パターン（外部サービスや他システムも当然OK）



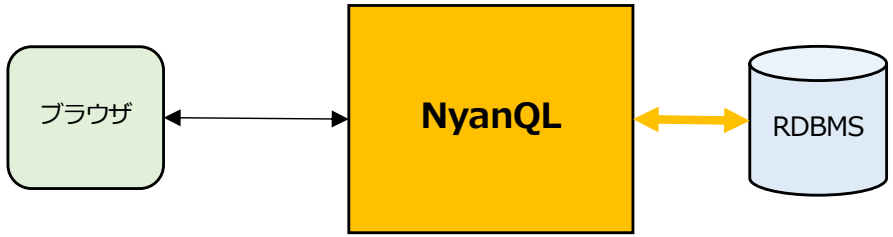
# その上で、基幹系システムをやるにはやっぱりこの構成！



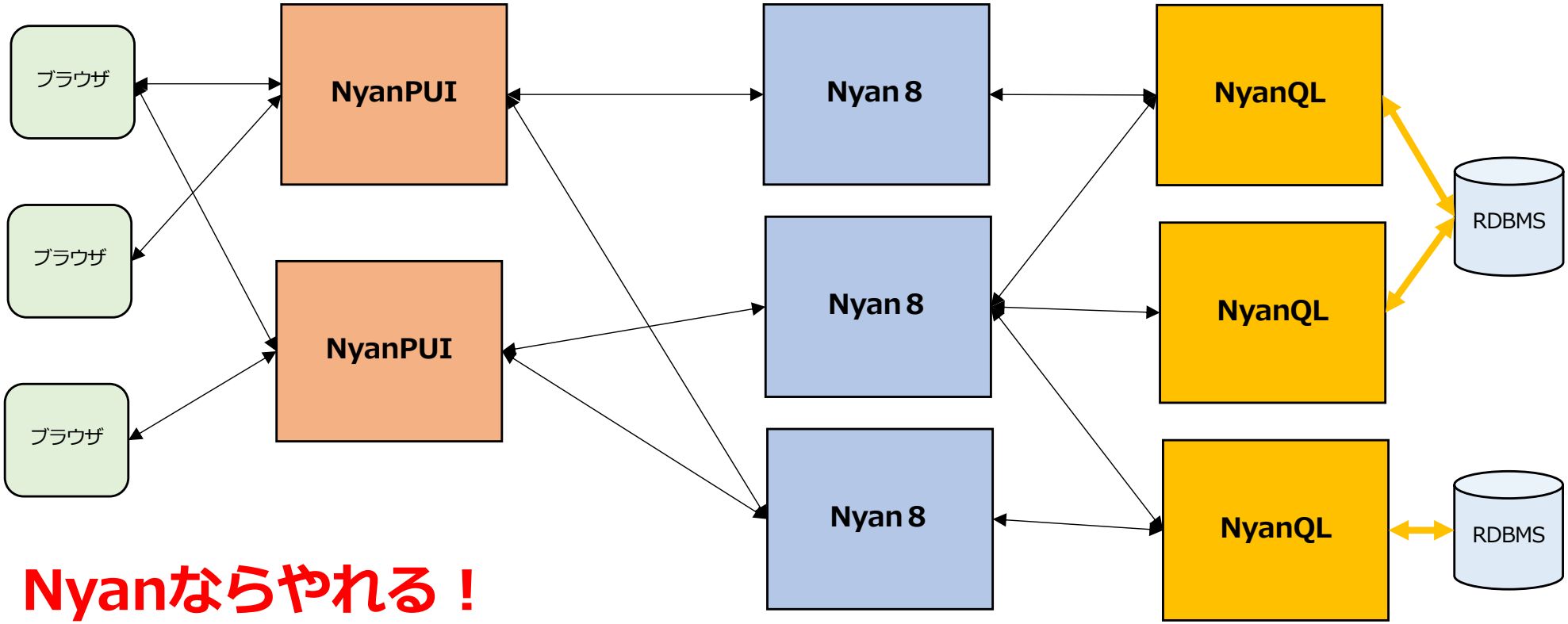
# だってやりたいのは「これ」でしょ！？



## ここから小さくはじめて…

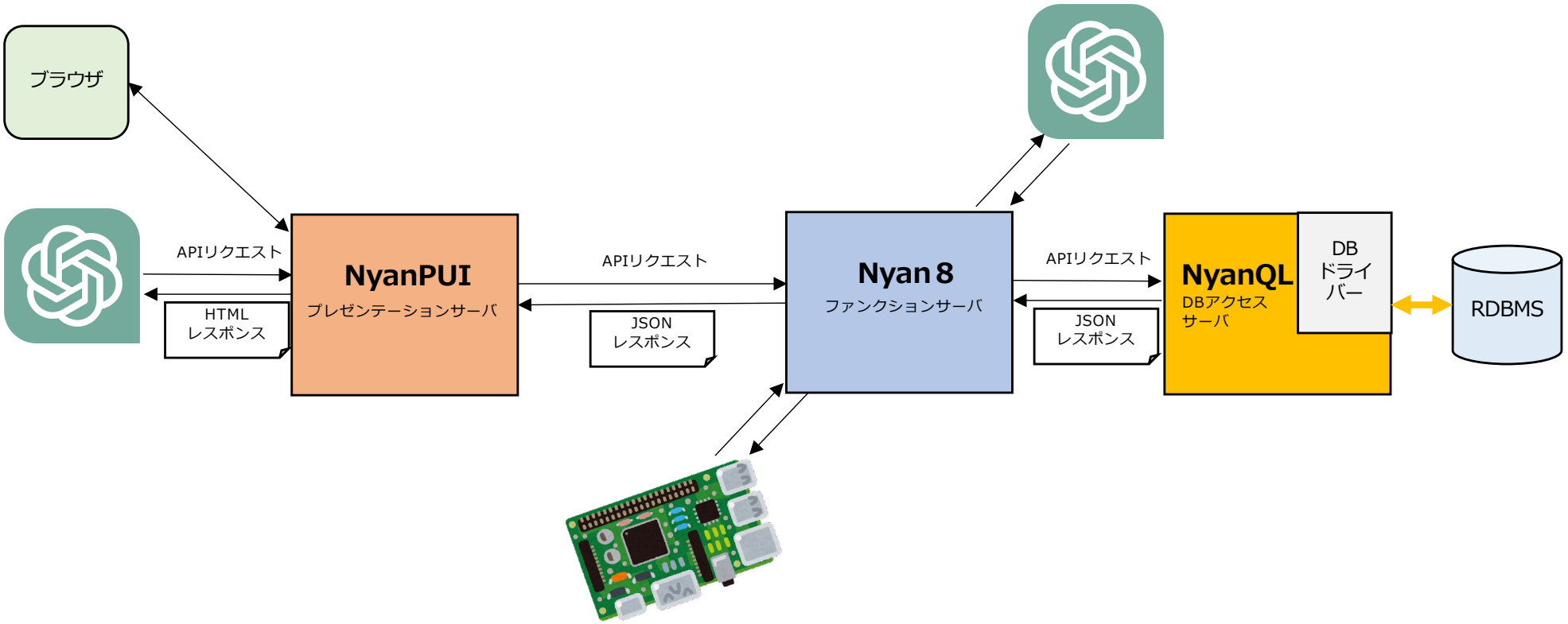



## ここまでシームレスにスケーリングしたいんでしょ？




**Nyanならやれる！**

# そして2025年の今、AIもHTMLを"見る"時代へ



# UIからでもDBからでも

- DOAだ！　データモデリングだ！　派も
- UI/UXが大事だ！　UIファーストだ！　派も
- サービス指向だ！　APIこそ重要だ！　派も

…どこから始めても大丈夫！　それがにゃん！

# 今後のロードマップなど

# 枯れされる！

- **バージョン番号の考え方**
  - ０．０．１１
  - メジャー、マイナー、連番
  - メジャー／マイナーに関わらず、連番が純粋なバージョン番号
  - マイナーは、バグ対応などでバージョンアップしてねという意思表明
  - メジャーは、仕様の互換性が損なわれるくらいの変更があったとき
- **現在のメジャーは、０（ゼロ）**
  - 仕様はもう少し揺れると思うので…
  - 皆さんのお力添えをいただきたく
  - 1.0.n にしたら、そこで仕様は一旦Fixです。

# 機能追加よりも、安心の向上を

- 基本的な機能は、概ね揃ってきてると思うので…
  - とはいいながら、積み残しもあるんですが…
    - api.json のホットリロードは個人的に欲しいかなぁ、とか
    - 画像やファイルの簡単アップロード／ダウンロードはいるなぁ、とか
    - にゃんぷれの逆版（HTMLから送信用のJSONをつくる）とか
  - NyanQLは機能的には、ほぼ完成かなと考えてます。
  - NyanPUIはもう少し強化します。
  - Nyan8は今後の生成AIの状況にに沿って機能追加していきます。
- **非機能要件のほうをしっかりやっていく＝皆さんに叩いて欲しい！**
  - **品質**（にゃん本体のバグ／誤動作を無くす）
    - にゃんで作ったユーザーアプリケーションの品質＝テスト支援は今後頑張る
  - セキュリティ（設定は容易に、運用時は堅牢に）
  - 性能（単独でより高速に、より省メモリで、とはいえハードの性能は上がるんだし…）
  - 継続性（同じ仕様のまま、ずっと使い続けられる）
  - 保守性（APIと、SQL,JS,HTMLをずっと使い続けられる）
  - スケーラビリティ（どんどん適用範囲を広げられるように＝API周り）
- **公式サイトとドキュメント整備を頑張っていきます**

# サンプルも頑張っていこうというお気持ち

- これは、完全に今の羽生の思いでしかないですが…
- こういうサンプルをまずは揃えたい
  - Webっぽいもの：チャット、掲示板、Wiki
  - ITっぽいもの：ネットショップ、予約（＝カレンダーUI）
  - 業務っぽいもの：入力フォーム、マスターメンテナンス
- ここまで行けば、業務系のサンプルがやれる
  - デジマジ→生成AI→にゃんの流れで
  - まずは本丸の販売管理
  - ここまで出来れば、基幹系システムがやれる
- これらのために、ワイヤーフレーム風のシンプルなCSSのサンプル

# 今の羽生の課題意識の一部として

- 先々は、より現場系／フィールド系／設備系へ
  - IoT（センサー＆アクチュエータ）／エッジ／デバイスのような
    リアルワールドに対するインターフェイスをフロントエンドと考える
  - RX（ロボティックス・トランスフォーメーション）との連携
  - …を見据えて、デバイスとNyan8の連携を強化（MCPとかもこの延長で）
  - いずれは、デバイスへのにゃんの組み込みも（WASMとか軽量化とか）
- こういうイベントに情報収集に行くような
  ところの需要にも、にゃんで応えたい

  https://www.fiweek.jp/osaka/ja-jp/lp/visit/robo.html

# 秋のにゃん祭り2025　開催決定！


次回は是非、皆さんにも発表していただければ嬉しいです(^^)

小さなサンプルとかでも十分なので

**引き続き、にゃんくるプロジェクトをよろしくお願い申し上げます**