

kintone の利便性と弱点

著者：関 勝寿 公開日：2026 年 5 月 25 日 ソース：<https://sekika.github.io/2026/05/25/kintone/>

大学や自治体、企業の現場で急速に広がったローコード基盤 kintone の強みと限界について、ChatGPT との会話ログをそのまま公開します。

みたいな、「業務の流れはあるが、専用システムを作るほどではない」ものを高速に作れます。



1.user:

kintone はそんなに便利？

2.gpt-5:

kintone は、かなり「刺さる組織には刺さる」タイプの業務アプリ基盤です。特に、

- ・Excel + メール + 手作業 + 属人化
 - ・「ちょっとした業務システム」が大量にある
 - ・情シスが弱い
 - ・でも現場は改善したい
- という組織では強いです。

一方で、「何でもできる魔法の道具」ではなく、向き不向きはかなりのります。

3. 何が便利なのか

kintone は雑に言うと、ためのローコード基盤です。

例えば：

- ・研究費申請管理
- ・備品管理
- ・問い合わせ管理
- ・進捗管理
- ・学会参加申請
- ・原稿管理
- ・学生指導記録
- ・実験予約

4. 強いところ

4.1.1. 非エンジニアでも作れる

フォームを GUI で作って、

- ・テーブル
- ・ステータス
- ・承認フロー
- ・通知
- ・権限

などを設定できます。

つまり、
のが大きい。

4.2.2. 「とりあえず動く」が異様に速い

普通の業務システム開発だと：

- ・要件定義
- ・DB 設計
- ・フロントエンド
- ・権限
- ・認証
- ・デプロイ

が必要ですが、kintone は最初からかなり揃っています。

なので、

が本当にできる。

4.3.3. Excel 地獄をかなり減らせる

多くの組織では、

- ・Excel が最新版がわからない
 - ・誰が更新したかわからない
 - ・メール添付で壊れる
 - ・マクロがブラックボックス
- という問題があります。

kintone はデータベース型なので、

- ・同時編集
 - ・履歴
 - ・権限
 - ・一元管理
- が比較的まともです。

5. 逆に弱いところ

5.1.1. 本格システムには厳しい

例えば：

- ・大規模 ERP
- ・高速トランザクション処理
- ・複雑な会計
- ・超高度なワークフロー
- ・巨大データ

みたいなのは苦手。
「軽～中量級業務システム」が主戦場です。

5.2.2. 作る人のセンスに依存する

Excel と同じで、

- ・命名規則が崩壊
- ・アプリ乱立
- ・権限ぐちゃぐちゃ
- ・フィールド定義が不統一

になりがちです。
なので、
が大量発生することがあります。

5.3.3. 結局 JavaScript が欲しくなる

最初はノーコードで始めても、

- ・ちょっと複雑な UI
- ・自動処理
- ・外部連携

をやり始めると、かなり JavaScript を書きたくなります。
すると突然、
状態になる。

6. 日本企業で強い理由

kintone は、日本企業の

- ・稟議
- ・ハンコ文化
- ・細かい承認
- ・Excel 運用
- ・部署ごとの独自業務

との相性が良いです。
海外 SaaS は、
という思想が強いですが、kintone はむしろ
寄り。
ここが日本で普及した大きな理由です。

7. 大学・研究室との相性

実はかなり良いです。
大学や研究室って、

- ・例外処理が多い
- ・ローカルルールが多い
- ・人の入れ替わりが多い
- ・でも専用システムを作る予算はない

ので、kintone 的な「半手作りシステム」が合うことが多い。
特に：

- ・研究室運営
- ・共同研究管理
- ・機器予約
- ・学生進捗
- ・科研費事務

あたりは相性が良いです。

8. ただし「銀の弾丸」ではない

失敗例も多くて、

- ・「全部 kintone でやる」
- ・「設計なしで増殖」
- ・「担当者しかわからない」
- ・「Excel より複雑」

になることもあります。
なので本質的には、
として使うと強い、という感じです。

8.1.user:

典型的なワークフローは？

8.2.gpt-5:

典型的には、
を、Excel とメールの代わりに一つの Web アプリで回す感じです。
kintone の基本単位は「アプリ」で、各アプリが

- ・一覧表
- ・入力フォーム
- ・データベース
- ・ワークフロー

を兼ねています。
例えば、かなり典型的な「出張申請」を例にすると：

9.1. フォーム作成

管理者が GUI で項目を作る。

- ・氏名
- ・所属
- ・出張先
- ・日程
- ・予算
- ・理由
- ・添付ファイル

など。
ここはほぼ Google Form 的です。

10.2. レコード入力

利用者が申請。
すると 1 件が「レコード」として保存される。
Excel 的には「1 行」。

11.3. ステータス管理

ここが kintone 的な肝です。
例えば：

下書き
↓
申請中
↓
指導教員承認
↓

事務承認
↓
完了

みたいな状態遷移を設定できる。

12.4. 権限設定

状態ごとに：

- ・誰が見られるか
 - ・誰が編集できるか
 - ・誰が承認ボタンを押せるか
- を変えられる。

例えば：

- ・学生：自分の申請だけ編集可能
 - ・教員：研究室メンバーを承認可能
 - ・事務：全件閲覧可能
- など。

13.5. 通知

状態が変わると：

- ・メール
 - ・kintone 通知
- が飛ぶ。

例えば：

「あなた宛の承認依頼があります」

14.6. 一覧・検索

全部 DB 化されているので、

- ・「未承認だけ表示」
- ・「2025 年度だけ」
- ・「科研費 A だけ」

みたいな検索ができる。

ここで Excel 地獄からかなり解放される。

15.7. 集計・グラフ

簡易的な：

- ・件数集計
 - ・円グラフ
 - ・月別推移
- も作れる。

16.8. 外部連携（欲しくなる）

そのうち：

- ・Slack 通知
- ・Teams
- ・Google Calendar
- ・会計システム
- ・SSO
- ・API 連携

をやりたいくなり、JavaScript や REST API を触り始める。

ここから「軽い業務改善」から「ちゃんとしたシステム開発」へ寄っていく。

17. 実際の現場で多い構成

実運用では、

マスタ管理アプリ
↓
申請アプリ
↓
承認アプリ
↓
集計アプリ

みたいに複数アプリを関連付けます。

つまり：

- ・教員マスタ
- ・予算マスタ
- ・学生名簿

などを別アプリ化する。

すると、だんだん「簡易 ERP」っぽくなっていく。

18. なぜ現場にウケるのか

重要なのは、

ことです。

普通の業務システムだと、

現場 → 情シス → SIer → 修正

ですが、kintone だと：

現場担当がその場で項目追加

できる。

この速度感がかなり大きい。

19. 逆にハマる罠

典型的には：

19.1. 最初

便利！

↓

19.2. 半年後

アプリが 80 個ある

↓

19.3.1 年後

誰も全体構造を理解していない

↓

19.4.2 年後

この JavaScript 誰が書いた？

になります。
つまり、
のが強みでもあり弱みでもあります。

19.5.user:

微妙に昔の MS Access っぽい雰囲気が

19.6.gpt-5:

かなり近いです。実際、多くの人が
みたいな印象を持っています。
ただし、完全に同じではなく、
・ Access 的な部分
・ Access では弱かった部分
・ 逆に Access より弱い部分
があります。

20. かなり Access 的なところ

20.1.1. 「業務 DB を現場が作る」

これが本質的に同じです。
例えば：
・ 顧客管理
・ 在庫管理
・ 申請管理
・ 実験記録
・ 問い合わせ台帳
みたいな、
領域。
これは昔の Microsoft Access が得意だった世界そのものです。

21.2. 「Excel の次」

Access って昔、

Excel では限界
↓
でも本格 DB は重い
↓
じゃあ Access

というポジションでした。
kintone もかなり近い。

22.3. 「現場主導」

Access 文化って、
が大量に生まれる世界でした。
kintone もまさにそうです。

23. ただし、Access と違う重要点

23.1.1. クラウド前提

Access は：

- ・ ローカル MDB
 - ・ ファイル共有
 - ・ ネットワークドライブ
- 文化でした。

なので：

- ・ 壊れる
- ・ 同時編集事故
- ・ バージョン衝突

が多かった。

kintone は最初から Web + クラウド DB なので、そのへんがかなり改善されています。

24.2. ワークフローが強い

Access は「DB アプリ作成ツール」寄りでしたが、kintone は：

- ・ 承認
- ・ 通知
- ・ 権限
- ・ ステータス管理

が最初から強い。

つまり、

をかなり意識している。

これは日本企業向け SaaS っぽい。

25.3. マルチユーザー前提

Access は実は：

がかなり苦しかった。

kintone は最初から：

- ・ ブラウザ
- ・ 権限
- ・ 同時利用
- ・ モバイル

前提。

26. 逆に Accessの方が強かったところ

面白いのですが、昔の Access は意外と「開発自由度」が高かったです。

例えば：

- ・ SQL 直書き
- ・ VBA
- ・ フォーム制御
- ・ レポート印刷
- ・ ローカル処理

など。

下手すると「小規模業務システム開発環境」でした。

kintone は逆に、
よう制限されています。

なので：

- ・ 超複雑 UI
- ・ 高自由度画面

- ・特殊帳票
はむしろ苦手。

27. 歴史的に見ると

実は系譜としてかなり近いです。

Excel
↓
Access
↓
グループウェア
↓
クラウド SaaS
↓
kintone

みたいな流れ。
特に日本企業では、
・「現場が自分で作る」
・「情シス不足」
・「業務が特殊」
という文化が強いので、Access 的需要在ずっと存在している。
kintone はそのクラウド時代版と言える面があります。
なので、「なんか Access っぽい」はかなり本質を突いています。

27.1.user:

逆に kintone で運用上不自由するところは？

27.2.gpt-5:

かなりあります。kintone は「軽量業務アプリ基盤」としては強いのですが、使い込むほど、
という限界が見えてきます。
典型的には次のあたりです。

28.1. リレーショナル DB としては弱い

ここは Access とも共通です。
例えば：
・複雑 JOIN
・正規化
・多対多関係
・高度な集計
・SQL 自由記述
がかなり弱い。

28.1. 典型的につらくなる例

研究費
↔ 教員
↔ 学生
↔ プロジェクト
↔ 出張
↔ 成果物

みたいな関係が増えると、
になりやすい。

29.2. UI 自由度が低い

kintone の UI はかなり「決まった形」です。
なので：
・複雑ダッシュボード
・高度なインタラクション
・特殊入力 UI
・リアルタイム画面更新
などは苦しい。

29.1. よく起きること

最初：

標準 UI で十分

半年後：

ここだけちょっとカスタムしたい

1 年後：

JavaScript で画面改造

2 年後：

もはや何のフレームワークかわからない

30.3. 帳票が弱い

日本業務で超重要なのに、ここが意外と弱い。
例えば：
・請求書
・稟議書
・見積書
・学内様式
・ハンコ位置固定 PDF
など。
kintone 標準では：
への対応が弱め。
なので結局、
・プラグイン
・外部サービス
・PDF 生成カスタム
に頼りがち。

31.4. バージョン管理が弱い

これはエンジニアが嫌がる点。
例えば：
・誰が設定変えた？
・いつ権限変えた？
・JavaScript 差分どこ？
・本番反映どう管理する？
が弱い。

31.1. 結果

本番で直接編集

文化になりやすい。

Access 的。

32.5. 「アプリ乱立問題」

これは本当に多い。
kintone は作るのが簡単なので、

「とりあえず作る」

が加速する。
すると：
・同じデータが 5 箇所
・命名不統一
・マスタ不整合
・似たアプリ大量発生
になる。

33.6. 性能限界

小～中規模向けなので、
・巨大データ
・超大量レコード
・複雑検索
・高頻度 API
・重い一覧
は徐々につらくなる。
特に：

「全部 kintone に入れる」

をやると危険。

34.7. JavaScript 地獄

kintone の怖いところは、
こと。
つまり：
・jQuery 断片
・野良コード
・CDN 依存
・属人 plugin
が増殖しやすい。

34.1. そして典型的に

退職者しかわからない

状態になる。
昔の Access VBA にかなり近い。

35.8. ガバナンスが難しい

現場主導が強みですが、逆に：
・情シス統制不能
・セキュリティばらばら

- ・権限設計崩壊
- ・データ定義不統一
が起きやすい。

36.9. 「あと少し」が永遠に続く

これはローコード全般の宿命。
最初：

80 点を 1 日で作れる

でも、

残り 20 点

が異様に重い。
すると：
・plugin 追加
・JS 追加
・API 追加
が積み上がり、
ゾーンに入ることがある。

37. 実は向いている用途はかなり限定的

強いのは：
・中規模以下
・業務フロー中心
・定型入力
・多少雑でも回る
・現場改善
です。

逆に弱いのは：

- ・厳密 DB 設計
- ・複雑 UI
- ・高性能要求
- ・長期大規模運用
- ・高度内部統制

つまり kintone は、
を埋めるのは非常に得意だけど、
にはなりきれない、
という理解がかなり正確です。

37.1.user:

ある程度、向いている人数の規模がありそうですね

37.2.gpt-5:

かなりあります。kintone は実際、
で限界が来るタイプです。
ただ、経験的にはかなり典型的なレンジがあります。

38. かなり向いている規模

38.1. 数人～数十人

ここは非常に強いです。

例えば：

- ・研究室
- ・小規模企業
- ・部署単位
- ・学科事務
- ・プロジェクトチーム
- など。

38.2. なぜ強いのか

この規模だと：

- ・「顔が見える」
 - ・暗黙知で回る
 - ・多少雑でも困らない
 - ・管理者 1 人でも把握可能
- だから。
つまり、
が成立する。

39. 一番ハマるのは「50 ～ 300 人くらいの部分最適」

実はここが kintone の黄金レンジです。

例えば：

- ・中小企業
- ・部署横断業務
- ・大学の特定部局
- ・病院の一部門
- ・工場の現場改善
- など。

40. この規模の特徴

ここでは：

- ・Excel 限界は来ている
 - ・でも SAP は重すぎる
 - ・情シス人数が少ない
 - ・現場改善ニーズは強い
- ので、
として非常に刺さる。

41. 逆に厳しくなる領域

41.1. 数百～数千人規模

ここから：

- ・権限管理
- ・データ整合性
- ・ガバナンス
- ・監査
- ・標準化

が急激に重要になります。

すると、

が逆に危険化する。

42. 典型的崩壊パターン

最初：

営業部で導入

↓

便利なので：

他部署も使う

↓

2 年後：

アプリ数 300

↓

3 年後：

誰も全体構造を理解していない

43. 大企業での実際の使われ方

面白いのですが、大企業でも：

ではなく、

として使われることが多いです。

つまり：

- ・SAP
 - ・Oracle
 - ・基幹システム
- は別にあり、
kintone は：
- ・現場改善
 - ・申請補助
 - ・一時 DB
 - ・部門独自運用
- を担当する。

44. 大学だと特にありがち

大学は：

- ・部局自治
- ・ローカルルール
- ・小規模組織の集合体

なので、kintone 的なものと相性が良い。

ただし全学展開すると：

- ・学部ごとに別文化
- ・権限複雑
- ・人事異動
- ・長期保守

で急に厳しくなる。

45. 実は「人数」より重要なもの

本当に重要なのは、

45.1.1. 業務が標準化されているか

標準化されているほど：

- ・ ERP
- ・ 専用 SaaS が強い。
- 逆に：
- ・ 例外多い
- ・ ローカル運用多い
- ほど kintone 向き。

45.2.2. データ整合性要求

例えば：

- ・ 会計
- ・ 在庫
- ・ 人事給与

みたいに、

絶対にズレてはいけない

領域は苦手。

45.3.3. 保守体制

kintone は：

- 思想なので、
- ・ 面倒を見る人
- ・ 改善を回す人

がいないと崩壊しやすい。

なので、かなり雑に言う：

規模	相性
1 ～ 30 人	非常に強い
30 ～ 300 人	黄金レンジ
300 ～ 1000 人	ガバナンス問題が出る
全社数千人	単独基盤は厳しい

くらいの感覚はあります。

45.4.user:

この記事では、この規模でうまくいっているように書いているけど、実際のところはどう？

[200 超アプリが全庁で稼働](#) 埼玉県庁はなぜ kintone を DX 基盤に選んだのか

45.5.gpt-5:

これは、かなり「本当にありえる成功例」だと思います。ただし重要なのは、

- ではなく、

という点です。

ここを誤解すると、「kintone で巨大組織全部いける」と見えてしまう。

46. 実は、この事例はかなり kintone 向き

- 記事を読むと、中心は：
- ・ 照会回答業務
 - ・ 進捗管理
 - ・ 部局横断の情報共有
 - ・ 承認フロー
 - ・ 集約
 - ・ 可視化

です。

これは kintone が最も得意な領域です。

47. 逆に、やっていないはずのもの

おそらくですが、埼玉県庁でも：

- ・ 人事給与
- ・ 財務会計
- ・ 住民 DB
- ・ 税
- ・ 基幹住民サービス
- ・ 大規模データ処理

を kintone に載せているわけではない。

そこは別の基幹系があるはずで。

つまり構造としては：

基幹システム
↑
kintone が周辺業務を接続

に近い。

48.13,000 人でも成立する理由

ここが面白いところです。

「13,000 人利用」と言っても、

13,000 人が同じ巨大アプリを使う

とは限りません。

実際には：

- ・ 小規模アプリが多数
 - ・ 各部局で局所利用
 - ・ 共通ワークフローだけ統一
- である可能性が高い。

49. むしろ「200 アプリ」は少ないくらい

実は、13,000 人規模で 200 アプリは、
印象です。

野放図運用だと：

2000 アプリ

くらいまで簡単に増えます。

なので、おそらく：

- ・ DX 推進部門
 - ・ 標準化ルール
 - ・ 命名規則
 - ・ 権限管理
 - ・ テンプレート化
- をかなりやっているはず。

50. 成功の鍵は「用途制限」

この記事で重要なのは、

を主ターゲットにしていることです。
これは行政組織で猛烈に多い。
例えば：

各課へ調査
↓
Excel 回収
↓
メール催促
↓
最新版不明
↓
集約地獄

という業務。
kintone はここに非常に強い。

51. 行政との相性も実は良い

行政には：

- ・定型フォーム
- ・承認フロー
- ・部局横断照会
- ・履歴保持
- ・アクセス権

が大量にある。
しかも：

- ・業務は多い
- ・専用システム化は困難
- ・Excel 依存が強い

ので、
としてはかなり合理的。

52. ただし、本当に難しいのはここから

記事を見る限り、今は：

業務の電子化・共通化

フェーズです。
問題は次の段階。

53. 危険領域 1：アプリ増殖

今後：

- ・各部局独自カスタム
- ・JS 改造
- ・plugin 依存
- ・独自運用

が増えると、
化するリスクはある。

54. 危険領域 2：人事異動

自治体で怖いのはこれです。
kintone の真価は：

現場主導改善

ですが、自治体は：

- ・2～3 年異動
- ・担当変更
- ・非 IT 職員

なので、

作った人がいない

問題が起きやすい。

55. 危険領域 3：統制 vs 現場自由

kintone は：

のが強み。
でも自治体規模では：

- ・セキュリティ
- ・個人情報
- ・監査
- ・権限
- ・保存年限

が重い。
つまり：

自由に作れる

と

統制したい

が衝突する。

56. なので、本当に成功かどうかは「5 年後」

ローコード基盤は、

56.1.1 年目

爆速で成果

↓

56.2.3 年目

乱立開始

↓

56.3.5 年目

保守体制勝負

になりやすい。

57. 逆に、この事例がかなり賢いと思う点

記事の範囲では、
という、

- ・横断性が高く

- ・ 定型化しやすく
- ・ Excel 依存が強く
- ・ 基幹系ではない

ところに集中している。

これは非常に筋が良い。

なので総合すると：

- ・ 「13,000 人で kintone は無理」という話ではない
- ・ ただし「全業務を kintone」は危険
- ・ この事例は比較的 kintone の得意領域を選んでいる
- ・ 真の勝負は長期保守とガバナンス
という感じです。