WHITE PAPER

Stop Buying More Software: Building Custom Apps on the Frappe Low-Code Platform

ERPNext is itself an app on the Frappe Framework — a full low-code platform. Before you buy yet another SaaS, you can model the app you actually need to sit right beside your ERP.

For CTOs, founders & digital leaders · **9 min read**

Acube Innovations · Official ERPNext Partner · acubeinnovations.com · +91 62358 66111

EXECUTIVE SUMMARY

Most growing companies don't have a software shortage — they have software sprawl. Every gap gets filled with another SaaS subscription, and soon a dozen tools each hold a slice of the business, none of them talk to each other, and the ERP is just one more island. This paper makes a case most software vendors won't: you often don't need to buy the next tool at all. ERPNext isn't a closed product — it's an application built on the Frappe Framework, a genuine low-code platform. That means the same platform running your ERP can run the custom app you actually need. We explain what a Frappe app really is — DocTypes that generate their own forms, list views, reports and REST API; role-based permissions; visual workflows; and room to script when you need it — and how a new app installed beside ERPNext shares its database, its users and its data. Then we draw the line honestly: when building is the right call, when buying still wins, and where an experienced partner earns their keep.

The problem isn't too little software — it's too much

Ask a growing company where its data lives and you'll rarely get one answer. Customers are in the ERP and the helpdesk tool. Projects are in a project app and three spreadsheets. A bespoke process — dealer onboarding, warranty claims, site inspections, grant tracking — runs in whatever SaaS someone signed up for last year, or in email. Each tool made sense on its own. Together they're a tax: subscriptions that renew, integrations that break, and a business whose truth is scattered across a dozen logins.

The reflex, when a new need appears, is to go shopping for tool number thirteen. But every new SaaS you buy is another data silo, another vendor relationship, another export-and-reconcile job, and another place your customer or item master has to be kept in sync by hand. The question worth asking before you buy isn't 'which tool is best?' — it's 'do we need to buy a tool at all, or do we need to model our process where our core data already lives?'

For anyone running ERPNext, that second option is real, and it's closer than most people realise — because ERPNext is not a monolith. It's an app on a platform, and that platform is open to you.

- SaaS sprawl — a dozen tools each holding a slice of the business, none of them a single source of truth.
- Integration debt — every new tool needs syncing to the others, and those syncs are what break.
- Duplicated masters — the same customers, items and employees re-entered and drifting out of step across systems.
- The reflex to buy — filling each new gap with another subscription instead of asking whether it could be modelled on what you already run.

ERPNext is an app on Frappe — not a closed box

The single most useful thing to understand about ERPNext is that it is not the platform — it's an application that runs on one. That platform is the Frappe Framework: a full-stack, open-source, metadata-driven web framework (Python and JavaScript, backed by MariaDB). ERPNext is simply the

accounting, inventory, manufacturing and CRM app built on top of it. In Frappe terms, 'erpnext' is one app installed on a site; the framework layer, 'frappe', is what it stands on.

The defining idea of Frappe is that it treats an application's structure as data. Rather than hand-coding database tables, forms, list screens and APIs for every entity, you describe the entity once — its fields, links and behaviour — and the framework generates the rest. This is what makes it a low-code platform: the repetitive scaffolding of a business app is produced for you, so the effort goes into your logic and your process, not boilerplate.

Crucially, the platform is open to anyone, not just to Frappe's own developers. The same mechanism ERPNext uses to define an Invoice or a Work Order is the mechanism you use to define your own business objects. You're not extending a product from the outside through a limited plugin API — you're building on the identical foundation, as a first-class app alongside ERPNext.

- Frappe Framework — the open-source, metadata-driven platform (Python / JavaScript / MariaDB) underneath ERPNext.
- ERPNext — one application installed on that platform, not the platform itself.
- Metadata as data — describe an entity once and the framework generates its table, UI and API.
- Open to you — your app is built on the same foundation ERPNext is, not bolted on through a limited plugin slot.

What a Frappe app actually gives you

A Frappe app is a Python package that runs on the framework, and its core building block is the DocType. A DocType is the model, the view and the controller for one kind of business object rolled into a single definition — a Purchase Order, a Warranty Claim, a Site Inspection, whatever your process needs. You define its fields and their types (data, link, select, date, currency, a child table for line items, and so on), and from that single definition Frappe generates a great deal for free.

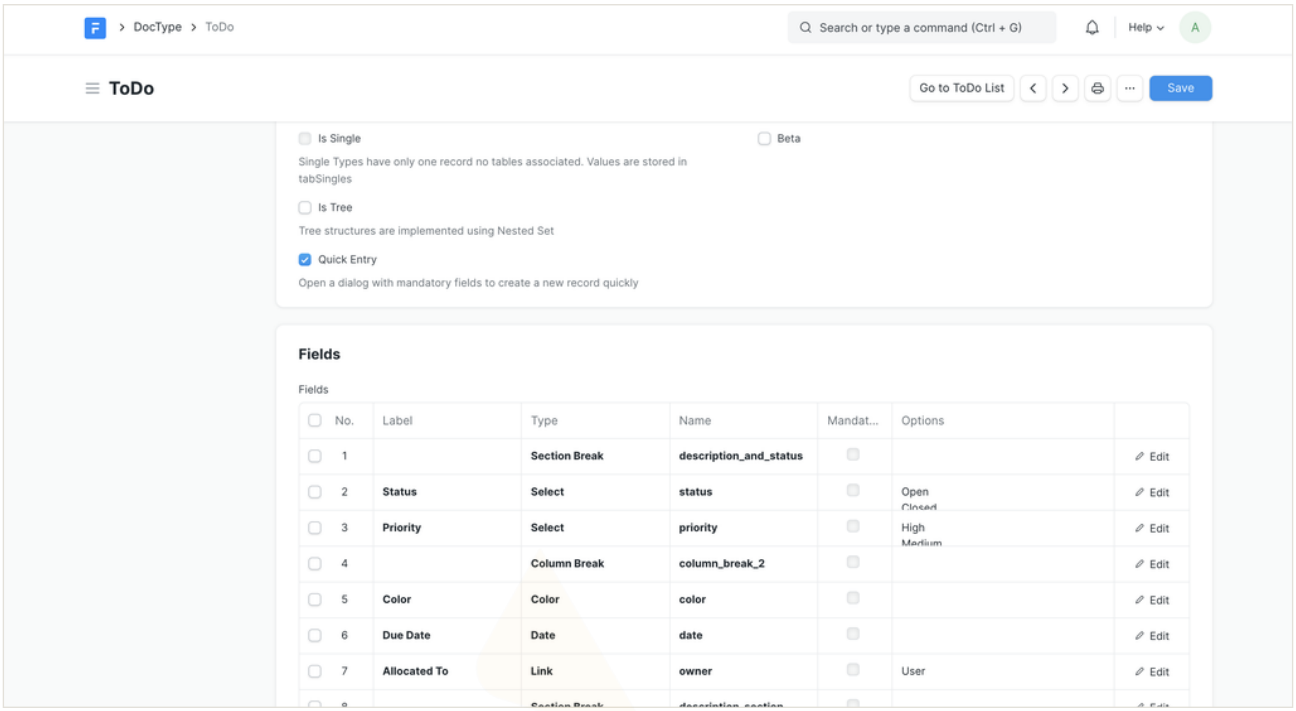
Define a DocType once and you get its database table, a form to create and edit records, a filterable list view, and the report and print machinery — without hand-building any of it. On top of that structure sit the capabilities that turn a data model into a real application: role-based permissions decide who can read, write, create, delete, submit or export each record; visual workflows move a document through states (say Draft to Approved to Closed) with the right people gating each step; and an automatic REST API exposes every DocType for create-read-update-delete over standard HTTP, so other systems can integrate without bespoke endpoints. When declarative configuration isn't enough, you drop into code — server-side controller methods in Python (validate, on-submit and other document hooks), client scripts in JavaScript for form behaviour, and whitelisted Python functions to publish your own API endpoints. That's the low-code promise kept honestly: configure what's routine, script what's genuinely yours.

- DocType — one definition that is the data model, the form and the controller for a business object.
- Generated UI — form view, list view, filters, reports and print formats, produced from the DocType, not hand-coded.
- Role-based permissions — control read, write, create, delete, submit, export and print per role, per DocType.

- Workflows — move a document through defined states with role-gated transitions and approvals.
- Automatic REST API — every DocType is exposed for CRUD over HTTP, so integrations don't need bespoke endpoints.
- Scripting when needed — Python controller methods and document hooks, JavaScript client scripts, and your own whitelisted API functions.

What one DocType definition gives you

1. **Database table**
   a table is created and maintained automatically from the DocType's fields; no schema by hand.
2. **Form view**
   a create/edit screen for each record, laid out from the fields you defined.
3. **List view**
   a filterable, sortable list of all records, with search and saved filters.
4. **Reports & print**
   report-builder and print-format machinery over the same data, code-free.
5. **REST API**
   automatic create-read-update-delete endpoints over HTTP for the DocType.
6. **Permissions & workflow**
   role-based access and state transitions layered on top of the same definition.



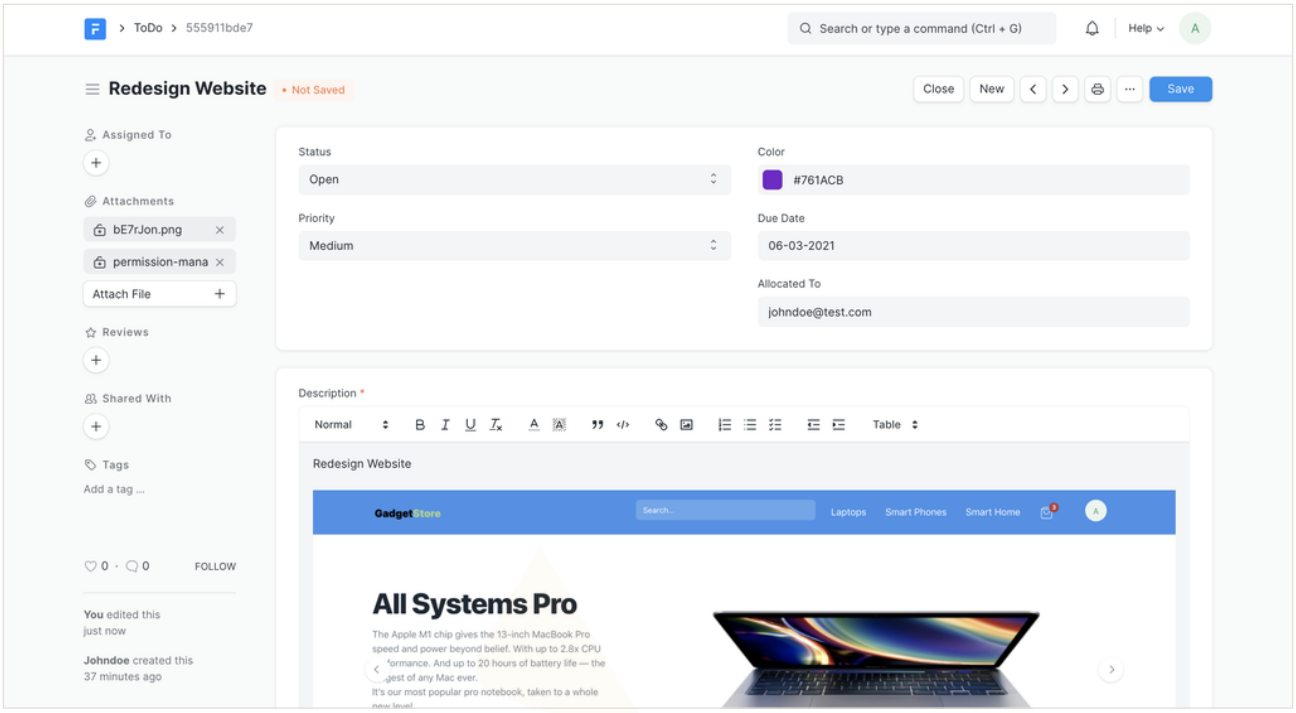
Defining a DocType in Frappe — you describe the fields (label, type, name, options) once, and the platform builds the table, form and API around them.

Building alongside ERPNext — and sharing its data

The reason this matters so much for an ERPNext user is what happens when your app is installed on the same site. Apps that share a site share one database and one set of users. Your new app isn't a separate system to be integrated with the ERP — it lives in the same place, so a login, a role, a customer, an item or an employee is the same record whether ERPNext or your app is looking at it. There is no export, no nightly sync, no reconciliation job, because there is nothing to keep in sync: it's one source of truth.

That also means your app can build directly on ERPNext's own objects. A DocType in your app can link to an ERPNext Customer, Item, Supplier or Sales Order, so your bespoke process is anchored to the master data the business already trusts. And Frappe gives apps clean ways to extend each other rather than fork them — through the app's hooks (the same mechanism ERPNext uses to attach scripts, override methods and react to events on documents), you can add behaviour to ERPNext's flows and react to its events without editing ERPNext's code, which keeps you upgrade-safe. In practice, this is the difference between 'we bought another tool and now we maintain the bridge between it and our ERP forever' and 'we modelled the process where our data already is.'

- One site, one database, one user list — ERPNext and your app share them, so there's nothing to sync.
- Shared masters — your DocTypes link straight to ERPNext's Customer, Item, Supplier and other core records.
- Extend, don't fork — hooks let you add behaviour and react to ERPNext's document events without editing its code.
- Upgrade-safe — because your app is separate code that hooks in, ERPNext can be upgraded without unwinding your work.



The form users actually work in — generated straight from the DocType definition, with fields, attachments, assignments and a rich-text section, no UI hand-coded.

When to build — and when to just buy

This paper is a case for building where it makes sense, not a case against ever buying software. Buying still wins in clear situations, and pretending otherwise helps no one. If a need is generic and a mature product already solves it well — email, video calls, payroll statutory filing, a specialist tax engine — buy it; you'll never out-build a focused vendor at their own game, and you shouldn't try. If a domain carries deep regulatory or security burden that a certified product already shoulders, that's usually theirs to carry.

Building on Frappe earns its place when the process is genuinely yours — a workflow, a data model or a way of working that's specific to how your business runs and that no off-the-shelf tool fits without contortions. It's compelling when the app's whole value is being joined to your ERP data: a process that constantly reads and writes customers, items, stock or orders is far better modelled where that data lives than bridged to it from outside. And it's the right call when you're otherwise about to buy your third or fourth overlapping subscription to cover variations of one internal process — consolidating that onto the platform you already run removes cost and silos at the same time. The honest test is simple: is this a solved, generic problem, or is it your problem? Buy the first. Build the second.

- Buy when it's generic and well-solved — email, payroll filing, video, specialist certified tools. Don't out-build a focused vendor.
- Build when the process is uniquely yours — a workflow or data model no off-the-shelf tool fits cleanly.
- Build when the value is the ERP link — the app lives and dies on reading and writing your core ERPNext data.
- Build to consolidate — when the alternative is yet another overlapping subscription for a variation of one internal process.

What it takes to build well

Low-code lowers the floor; it doesn't remove the craft. A Frappe app can be stood up quickly, but a good one still rests on decisions that are cheap to make well at the start and expensive to unpick later. The data model is the foundation: getting your DocTypes, their fields and — above all — the links between them and ERPNext's masters right is what makes the app feel native rather than bolted on. Permissions and workflow should mirror how the business actually approves and controls things, not a guess, because that's what makes people trust the app with real work.

The discipline that pays off most is restraint about code. Frappe rewards configuring what the platform already does and writing code only for the logic that's truly yours; over-scripting what a workflow, a permission rule or a report builder could have done is how a clean app slowly becomes a maintenance burden. And because your app shares a site with ERPNext, it has to be a good neighbour — extending through hooks and events rather than editing core, so ERPNext stays upgradeable. None of this is exotic. It's the difference between an app you're glad you built and one that quietly becomes the next thing nobody wants to touch.

- Model first — DocTypes, fields and links to ERPNext masters designed deliberately, not accreted.
- Permissions and workflow that match reality — mirror how the business really approves and controls, so people trust it.
- Configure before you code — script only the logic that's genuinely yours; let the platform do the rest.
- Be a good neighbour to ERPNext — extend via hooks and events, never edit core, so upgrades stay clean.

When to get help

Frappe is learnable, and plenty of in-house teams build capable apps on it. The place a partner earns their fee is at the decisions that are hard to reverse: the shape of the data model, how your app links into ERPNext's masters, and where the boundary between configuration and code should sit. Getting those right once, up front, is far cheaper than discovering a year in that the model can't represent a case the business now needs, or that a tangle of scripts has made every ERPNext upgrade a risk.

As an official ERPNext partner, we build custom Frappe apps that sit beside ERPNext — sharing its data, its users and its upgrade path — for exactly the processes that don't belong in yet another SaaS. If you're weighing whether to buy the next tool or model the process on the platform you already run, that's a conversation worth having before the subscription renews, not after. The goal isn't more software. It's fewer systems, one source of truth, and an app that fits how you actually work.

KEY TAKEAWAYS

1. ERPNext isn't a closed product — it's an app on the Frappe Framework, a full low-code platform that's open to you too.
2. In Frappe you define a business object once as a DocType and get its table, form, list view, reports and REST API generated for free.
3. On that structure sit role-based permissions, visual workflows and room to script in Python and JavaScript when logic is genuinely yours.
4. An app installed beside ERPNext shares one database and one user list, links to ERPNext's masters, and extends it via hooks — so there's nothing to sync and upgrades stay safe.
5. Buy the generic, solved problems; build the ones that are uniquely yours or that live and die on your ERP data — instead of adding yet another SaaS silo.

FAQ

Is ERPNext built on the Frappe Framework?
Yes. ERPNext is an application built on the Frappe Framework — an open-source, metadata-driven web platform (Python, JavaScript and MariaDB). 'frappe' is the platform layer and 'erpnext' is one app installed on top of it. The same platform can run other apps, including custom ones you build.

What is a DocType in Frappe?
A DocType is Frappe's core building block — a single definition that acts as the data model, the form and the controller for one kind of business object. From one DocType, Frappe automatically generates the database table, a form view, a filterable list view, reporting and print formats, and a REST API. On top of that you configure permissions and workflows and, where needed, add Python and JavaScript code.

Can a custom Frappe app share data with ERPNext?
Yes — that's the main advantage. An app installed on the same site as ERPNext shares one database and one set of users, so a customer, item or employee is the same record for both. Your app's DocTypes can link directly to ERPNext's Customer, Item, Supplier and other masters, and can extend ERPNext's flows through hooks without editing its code, so there's nothing to sync and ERPNext stays upgradeable.

Should we build a custom app or just buy another SaaS tool?
Buy when the need is generic and already well-solved — email, payroll statutory filing, video, specialist certified tools — because you won't out-build a focused vendor. Build on Frappe when the process is uniquely yours, when the app's value is being joined to your ERPNext data, or when the alternative is yet another overlapping subscription for a variation of one internal process. The test: is it a solved, generic problem, or is it your problem?

Talk to a real ERPNext expert.

Call or WhatsApp +91 62358 66111 · info@acube.co · acubeinnovations.com