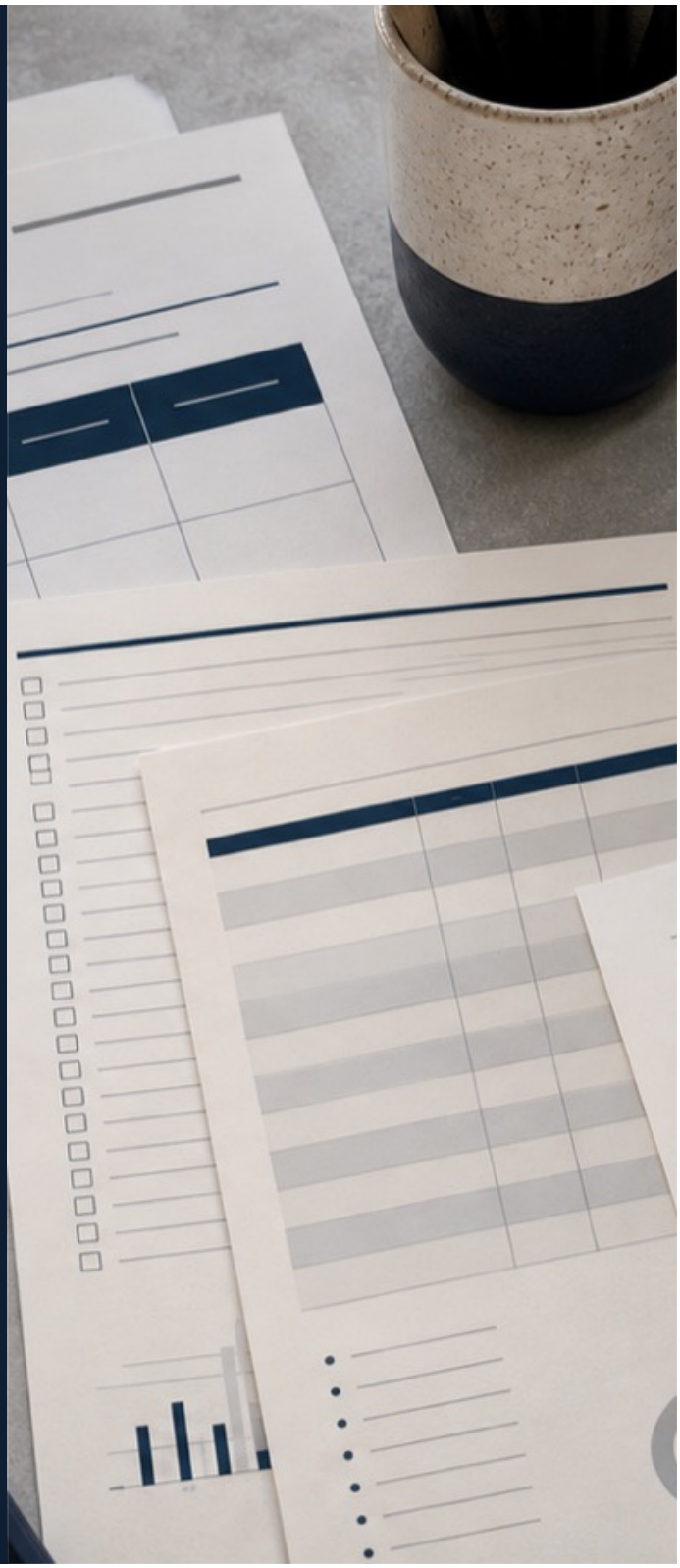
Rangefront Labs

Worksheet · Free resource

Systems Integration Requirements Worksheet

Most integration projects don't fail on the code. They fail because nobody agreed which system is the source of truth, or noticed the legacy app has no API. This worksheet surfaces those answers before the build, when...



Updated May 2026

https://rangefrontlabs.com.au/resources/systems-integration-requirements-worksheet/

Built in Toowoomba. Working across Australia and internationally.

“Can’t you just connect them?” is the most expensive sentence in business software. Two systems that both store customer records sound easy to join up, right up until someone asks which one is correct when they disagree, what happens when a sync fails at 2am, and whether the ten-year-old finance system even has a way in.

Those questions don’t go away if you skip them. They just turn up mid-build as surprises and change requests. This worksheet drags them into the open early, while they’re still cheap to answer. Work through it before you commission an integration, and you’ll get a far more accurate scope and far fewer nasty discoveries.

The worksheet

Fill in what you know and flag what you don’t. The gaps are as useful as the answers: a blank “does it have an API?” is exactly the thing to chase before you commit.

1. The goal

[In one sentence: what should be true once these systems talk to each other? e.g. “A sale in the online store automatically appears in our accounting system and updates stock.”]

2. Systems inventory

For each system involved:

System	Vendor / version	What it holds	Has an API?	Who owns it internally
[CRM]	[e.g. HubSpot]	[contacts, deals]	[Yes / No / Unsure]	[name]
[Accounting]	[e.g. Xero]	[invoices, payments]	[Yes / No / Unsure]	[name]
[Other]				

3. The data that must flow

For each piece of data moving between systems:

Data	From → To	Direction	When it moves	How often
[Customer record]	[Store → CRM]	[one-way / two-way]	[on new order]	[real-time / hourly / nightly]
[Invoice]	[CRM → Accounting]			

4. The source of truth

[For each shared piece of data, which system wins if they disagree? This is the question that quietly sinks integrations. Decide it now. e.g. “Accounting is the source of truth for payment status; the CRM never overwrites it.”]

5. Volume and timing

[Roughly how many records move per day? Does anything need to be instant, or is nightly fine? Are there peaks (end of month, sale events) that change the answer?]

6. Authentication and access

[How do you authenticate to each system (API key, OAuth, a username and password)? Who holds the credentials? Are there licence tiers or limits that affect API access?]

7. When something goes wrong

[If a sync fails, what should happen: retry, queue, alert someone? Who gets told? How will you know a record *didn’t* make it across? Silent failures are the worst kind.]

8. Constraints and known traps

[Legacy systems with no API, rate limits, data that must stay onshore, compliance obligations, anything held together with a manual export today. Name the awkward bits here.]

9. Success and ownership

[How will you know it’s working? Who owns the integration once it’s live and watches that it keeps working?]

How to use it

1. **Fill in the systems inventory first.** The “Has an API?” column alone will tell you whether this is a tidy job or an adventure. “Unsure” means go and find out before you budget anything.
2. **Settle the source of truth for every shared field.** If two systems can both edit a customer’s address, one of them has to lose. Decide which, in writing, now.
3. **Think hard about failure, not just success.** The happy path is the easy 10%. Most of the real engineering is in what happens when a record is malformed, a system is down, or two updates collide.
4. **Flag the manual workaround you’re replacing.** That nightly CSV export someone does by hand is a clue about data shape, volume and edge cases: write it down.

Common mistakes this catches

- **No agreed source of truth.** Two systems silently overwriting each other is the classic cause of “the data’s wrong again”.
- **Assuming an API exists.** Plenty of business systems, especially older or on-premise ones, have no clean way in, which completely changes the approach and cost.
- **Ignoring error handling until production.** “What happens when it fails?” is a design question, not a bug to fix later.
- **No owner after launch.** Integrations need watching. Without an owner, the first silent failure goes unnoticed until it’s a mess.

Get this worksheet filled in and you’ll have the answers a developer actually needs to quote the job properly. When you’re ready, send it to us. Connecting business systems cleanly, without the headache, is exactly what our integration work is for.

Need this adapted to your organisation, systems or data? Book a discovery call: https://rangefrontlabs.com.au/contact/