# Software Development RFP Template

## How to use this template

This template helps you write a request for proposal (RFP) for a custom software project — a web platform, an internal tool, a chatbot, a mobile app. Replace each bracketed prompt with your own answer. The “why this matters” notes explain what vendors do with each section: read them, then delete them before sending. Gaps are fine — “we don’t know yet” is a legitimate answer that a good vendor will treat as a discovery task, not a defect. One rule above all: a short, honest RFP gets better proposals than a long, vague one.

## 1. Company & context

*[Company name, website, industry, and size — headcount or a revenue band.]*

*[Two or three sentences: what the company does, and for whom.]*

*[Who is sponsoring this project internally, and why it has become a priority now.]*

**Why this matters:** vendors calibrate everything to context. A 20-person logistics company and a 500-person bank need different architectures, different processes, and different price points. Without context, every vendor guesses which version of the project you need — and each guesses differently, so the proposals you receive cannot be compared.

## 2. The problem — not the solution

*[Describe what hurts today, in plain language. What takes too long, breaks too often, costs too much, or blocks growth?]*

*[What does the problem cost you — in hours, money, errors, or lost customers? Rough estimates are fine.]*

*[What happens if nothing changes for another year?]*

**Why this matters:** this is the highest-leverage section in the document. If you prescribe a solution (“we need a React app with a microservices backend”), you will receive quotes for your guess — including from vendors who can see a better or cheaper route but quote what was asked, because that is what wins bids. Describe the problem instead, and each proposal shows you how that vendor thinks. If you have already chosen a solution direction, say so — but state the problem first, and invite vendors to challenge the direction.

## 3. Current process & systems

*[Walk through how the work is done today, step by step — including the manual steps, spreadsheets, and workarounds nobody is proud of.]*

*[List the systems involved (CRM, ERP, accounting, e-commerce platform…) and which of them must stay.]*

**Why this matters:** software projects rarely fail on code; they fail in the gap between how a process is described and how it actually runs. The vendor will discover the real process eventually. If they discover it during estimation, it is priced in. If they discover it mid-project, it becomes change requests.

## 4. Scope: must-have vs nice-to-have

*[Must-have: the capabilities without which the project has failed. Keep this list brutally short.]*

*[Nice-to-have: everything you would like eventually — candidates for a second phase.]*

*[Explicitly out of scope: anything a vendor might reasonably assume is included, but is not.]*

**Why this matters:** an undifferentiated list of forty features forces vendors to price all forty as mandatory — you pay must-have prices for nice-to-haves. The split also gives every vendor the same starting point for proposing phases, which makes their timelines and prices comparable with each other.

## 5. Users & volumes

*[Who will use the system: each role, roughly how many people, how often.]*

*[Volumes: orders, documents, messages, or transactions per day or month — today, and expected in two years.]*

*[Peaks and seasonality, if any.]*

**Why this matters:** “about 50 staff use it daily” and “100,000 customers with seasonal spikes” are different systems at very different prices. When volumes are missing, careful vendors pad the estimate to cover the unknown, and careless ones underprice a system that will fall over. Rough numbers beat no numbers every time.

## 6. Integrations & constraints

*[For each system the software must talk to: its name, what data flows in which direction, whether it has an API, and who on your side owns it.]*

*[Regulatory and data constraints: GDPR, industry rules, data residency, accessibility requirements.]*

*[Technical constraints or preferences, if you genuinely have them: hosting, existing infrastructure, languages the interface must support.]*

**Why this matters:** integrations are the classic quiet budget-mover — each external system brings its own interface, its own failure modes, and its own owner to coordinate with. A named list with

API status turns the vaguest part of any estimate into one of the most concrete.

## 7. Success criteria & metrics

*[Six months after launch, what has measurably changed? Pick two or three numbers: processing time, error rate, conversion, support load, revenue.]*

*[The current baseline for each of those numbers, even roughly.]*

**Why this matters:** success criteria give vendors a target to design toward rather than a feature list to burn down — and they move the acceptance conversation to now, when it costs nothing, instead of to the final invoice, when it costs the relationship.

## 8. Timeline & budget frame

*[Desired launch date, and whether it is a preference or a hard deadline. If hard, what makes it hard — a season, a contract, a regulation?]*

*[Budget range: a lower and an upper bound you are prepared to invest.]*

**Why this matters — an honest note on budget:** companies hide the budget out of fear that vendors will price up to it. In practice, vendors scope to it. Almost any software problem has a $30,000 version and a $300,000 version; without a range, you receive proposals scattered across that whole spectrum, none comparable to another, most solving a different-sized problem than yours. A stated range filters out mismatched vendors before anyone wastes a week — including yours.

## 9. Team & decision process

*[Who is the product owner on your side — the person who answers questions, attends demos, and makes day-to-day calls? How much of their week is realistically available?]*

*[Who signs off on the final decision, and what does approval involve — board, procurement, legal?]*

**Why this matters:** past the first weeks, the biggest schedule risk in most software projects is not engineering speed — it is decision speed on the client side. A named product owner with real availability changes how a serious vendor plans your project; the absence of one tells them exactly what the delays will look like.

## 10. Evaluation criteria for proposals

*[How you will weigh proposals — for example: relevant experience, quality of approach, team seniority, price, communication, each with a percentage.]*

Then ask every vendor these questions — the answers separate strong vendors from polished sales decks:

- **Who exactly will do the work?** Names, roles, and seniority — and whether the people in the sales conversations will be the people on the project.
- **What happens after launch?** Who fixes bugs, what support costs, what response times are guaranteed — in the contract, not in the pitch.
- **How are changes priced mid-project?** Scope will shift; it always does. A vendor without a clear change process is quoting a fixed price for an unfixed project.
- **What does a normal week look like?** How often you see working software, through which channel you communicate, and who your single point of contact is.
- **Tell us about a similar project where something went wrong.** Every experienced vendor has one. A vendor with no such story is not a vendor without failures — it is a vendor without honesty about them.

**Why this matters:** proposals differ less in price than in assumptions. These questions force the assumptions into the open — and a vendor who cannot name the team, price a change, or describe life after launch is showing you the project's future before it starts.

## 11. Submission logistics

*[Deadline for proposals, and the date you expect to decide.]*

*[Where to send proposals, and in what form. A short document that answers your questions beats a 60-page deck — say so.]*

*[A contact for questions, and a Q&A window before the deadline.]*

*[Whether shortlisted vendors get a call or a demo before the final decision.]*

**Why this matters:** leave room for questions, and note who asks them. The questions a vendor asks before proposing are a free preview of how they think — and a vendor who asks nothing at all is planning to price your project blind.

Prepared by Momentum Squads — momentumsquads.com