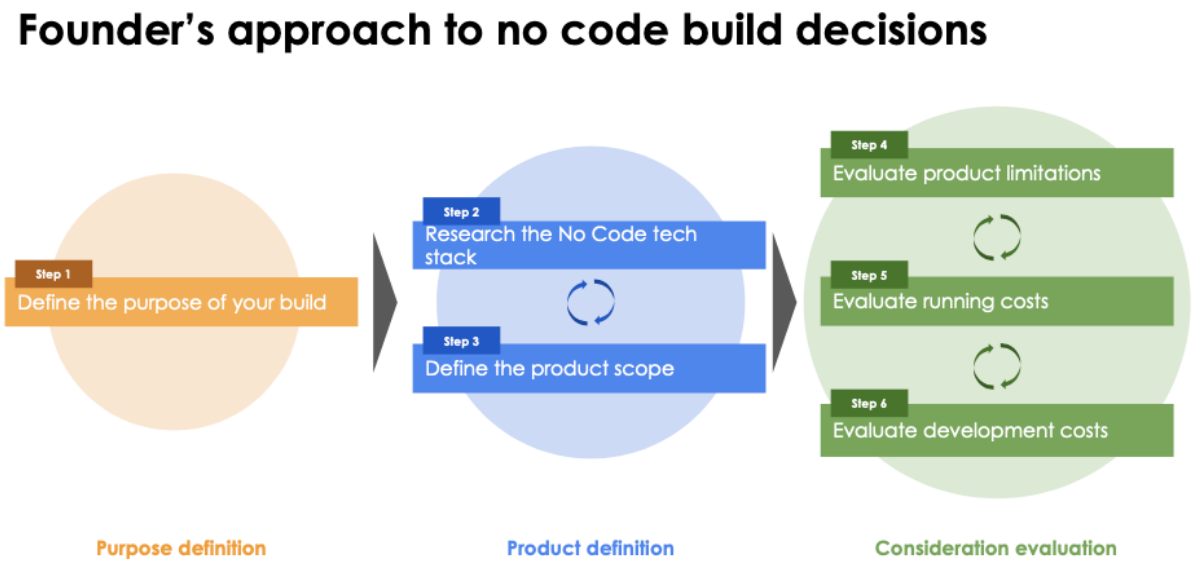
# A Founder’s Guide to No Code

Author: Annais J. Paetsch

The no code and low code industry has been developing quickly in the last 3-4 years with no code covering more and more use cases. Gartner projects a 23% increase for the global market and predicts 65% of overall application development activity to happen in low-code by 2024[1]. What does this accelerated maturing of the no code/low code industry mean for entrepreneurship? What is the potential of no code you can leverage as founder or early-stage team member? What can or can’t I build in no code? What should I consider when building in no code? And finally, how do I evaluate and decide, if it’s right for me?

If you are a founder or early-stage team member facing the question of whether to build in no code/low code or not, this guide is meant for you. For this guide, I analyzed 100 no code startups and interviewed 10 no code founders, no code CTOs or no code developers to provide guidance to decipher their evaluation process into a practical, actionable guide. In the following, “no code” refers to both no code and low code.

**A quick overview of this guide**




## Step 1: Define the purpose of your build

Before you start your No Code build journey, be clear on what purpose you are building for. This will greatly influence the scope of your build and your evaluation process thereof. From the founders I interviewed I heard of no code tools being used to achieve one out of three objectives:

- **Build to validate your idea and raise pre-seed funding:** For this objective you are trying to build a first version of your product to visualize it and get first user feedback. For this, you may want to build in No Code already and have users test basic functionality. Your no code prototype will typically focus on experimental questions pure design mockups cannot answer. Be sure to determine for yourself, a mockup to show users for feedback cannot "get the job done" at this stage.
- **Build to validate product-market fit and raise seed funding:** For this objective you are trying to build a product users actually *need*, gather in-depth feedback on feature requests, shift direction and adapt. Quick development, feedback and iteration are key here to get to build something your users want, leave you with time to focus on distribution and build a solid initial user base. This is also where no code tools currently excel (though the industry is developing towards more scalability). A risk here is overbuilding, so scope down as much as possible and list must-have

features only. This approach competes with building a fully-fledged raw code prototype from the beginning.

- **Build to get to series A and/or keep core (parts of) product in No Code:** For this objective you are trying to build core parts or your full product in no code. Thus, building a product that can scale with your projected needs for your defined time horizon (1-2 years is a reasonable timeframe) and can extend to foreseeable features on your roadmap are relevant to the no code go or no-go decision at this stage. Running costs (e.g. maintenance) also carries more weight at this stage. This approach, too, competes with building a fully-fledged raw code prototype from the beginning.

## Step 2: Research the No Code tech stack for your product

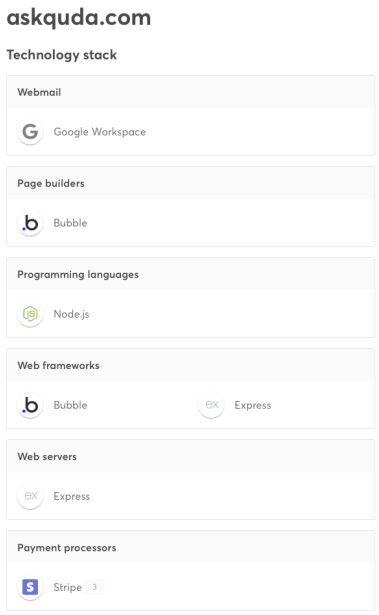
- **Find similar products to yours built in no code and uncover the tech stack (bottom-up tool search):** To quickly get to the tools that may be most relevant for the tech stack of your products, finding similar products built in no code can be helpful. To find these products, you can search no code aggregator websites and find similar no code products and features to yours. Typical no code products and features today include: eCommerce sites, appointment scheduling, marketplaces, listing directories, workflow automation, dashboards, etc. If you are interested in research on typical no code products and features, refer to my research of 100 no code startups here.

  Another way to find similar products is to browse communities for no code founders, build-in-public entrepreneurs, no code tools' internal communities (e.g. Bubble community) for products similar to yours. This can be products in completely different domains. You will want to abstract from content and look for products with similar functionality. Helpful communities may include: No Code Founders (founders here list the tech stack of their startups), Indie Hackers, Twitter (#nocode, #buildinpublic, #odnc - referring to OnDeck No Code cohort, #odnc2, etc.), Product Hunt (to find more new no code tools), Bubble's forum (and other no code tools' internal communities), etc.

  If you find a similar product, you can partly uncover the tech stack by using tools like builtwith.com and Wappalyzer (chrome extension), which lay out the tech stack of a given website such as the page builder, analytics, web framework and more. A

drawback is that these tools recognize the frontend tech stack well, but less so the backend. This means that no code databases (e.g. Airtable, Supabase) or no code workflow automation engines (e.g. Zapier, Integromat) are usually not recognized by these tools.

Lastly, try and get in touch with the founders. The no code community in particular is very happy to jump on a call, answer specific questions and share learnings.




Example: Wappalyzer result for askquda.com

- **Find the best-in class tools to build your no code product (top-down tool search):** Beyond getting inspiration from other products, checking for most popular tools in main no code product categories gives a quick lay of the land to expand the long-list of potential alternative tools and to get familiar with what the no code tools cover in terms of use case.

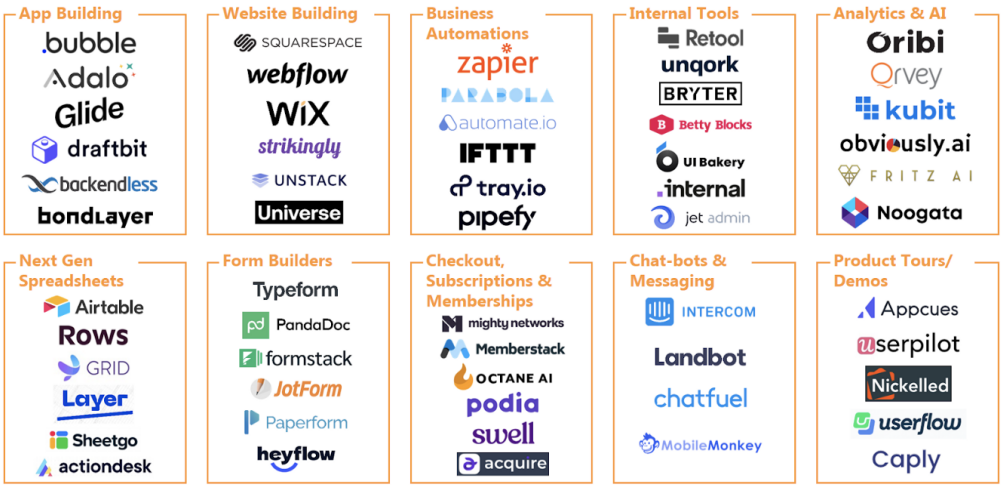



Image source: @mtommasi94 via Medium[2]

## Step 3: Define the product scope

As mentioned, a big risk when validating your idea is overbuilding. If you are in an early stage for your product, scope down your MVP in several iterations to make sure to get to the bare bones without fancy functionality. You will want to build an MVP to test your core proposition with as much focus as possible. After scoping down your product idea, break down your product into overall user experience of screens as wireframes and workflow. Process-heavy products will have less emphasis on screens and more on workflow steps for the automation. You can do this for low-fidelity prototypes (pencil and paper prototypes, as seen below) or high-fidelity prototypes (digital design of screens). If you have little or no experience with creating prototypes a low-fidelity prototype with pencil and paper is a great start to orient yourself and start breaking down your vision. Remember, if you haven't validated these screens of your key features in user interviews yet, it may be more resourceful to validate your product with screen mockups before building in no code.

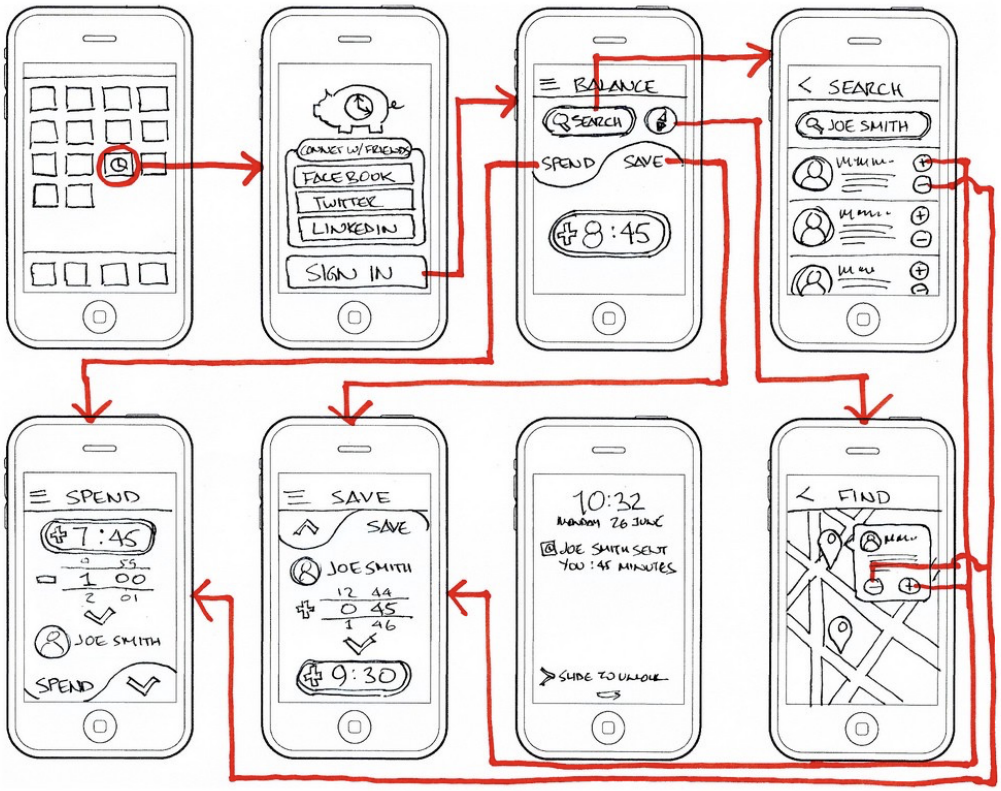



Image source: Nazil Kaya via Medium[3]

A few things to note as you draw your wireframes:

- **Highlight interfaces with other tools:** As you draw the wireframes, highlight key features and start referencing what no code tools could build given features or not. Importantly, start annotating where tools interface. Databases and workflow automation stand out especially here. Highlighting where the tools interface, will help you understand how your tools need to integrate to check this later. Additionally, you will want to note which no code tools could accomplish building the respective features on the screens.
- **For workflows, start manually, automate later:** If you are struggling to draw out the workflows (especially if your tool is workflow-heavy), start by doing what your product would do manually. For example, one founder I interviewed built a startup that requests tech companies and companies scraping personal information to delete one's this private information (e.g. email). He started by manually doing this service and then began automating the workflow steps with no code tools.

- **Get external help for wireframes, if needed:** Lastly, if wireframes are completely new territory and you feel intimidated, you may consider getting a consultant to create your wireframes via platforms such as Fiverr or Upwork (approximately $50-70/hr). Typically for an MVP 2-3h should suffice to create wireframes to breakdown your product, give you clear guidance on what needs to be built and (as will be discussed in final chapter) request quotes from developers. Some founders stated that for them the money spent on wireframes was very well invested money to reduce risk and create more certainty.

# Step 4: Evaluate product limitations

## Assess No Code tool feasibility to build your product

- **Compare to common no code product types and reverse engineer similar products:** Leverage your research from step 2 to review similar no code products and gain insight into what one can potentially build in no code. As mentioned, don't shy away from messaging founders for calls to learn more. Refer to "Step 2: Find similar products to yours built in no code and uncover the tech stack (bottom-up tool search)" for more detail.
- **Ask for advise from experts:** If you still feel uncertain about your whether and how certain features can be built, ask for help. Almost all founders I interviewed, mentioned asking for help as the main way they learned and could get "unstuck" in the building process. One person found a mentor through a no code tool he chose. As they want people to use their products, they are often motivated and incentivized to pair you up with mentors. This is how one founder in Europe got a mentor in Singapore to learn how to build in Adalo (mobile app builder). Forums and communities are another great way to ask about very specific questions to get certainty. Most tools have their own forum, so ask away there and join Slack or Discord communities to get the help you need (see research section for examples communities).
- **"Crash test" by building the most complex feature or the feature you have most uncertainty about first:** Lastly, to test whether you can build your product, go ahead and test it. You can be strategic about this by building the features with

lowest certainty (e.g. have not seen specific feature in other products built with tool X) or by building the most complex feature first.

One CTO I interviewed, knew that some of the mathematical calculations in their table would be most complex, so their team built this first and found that they in fact were missing the “log” calculation ability in App Sheet.

- **Don’t dismiss no code tools for missing a feature immediately:** The CTO could have dismissed App Sheet as a tool to use realizing it was missing a relevant feature. However, instead they requested the App Sheet team to incorporate this feature, which App Sheet did. Many no code tools are pretty responsive to user feedback and the younger they are the more they are trying to build their user base, so requesting a feature can be a very viable approach to not dismiss a tool for one shortcoming.

## Evaluate scale & performance considerations

There are three types of ways your products can be limited when scaling:

1. Hard caps: Scaling works until a fixed, defined point
2. Costly scaling: Scaling a specific dimension is subject to tiered-pricing leadings to steep increases in running cost
3. Performance-reducing scaling: Scaling hurts performance and cannot be upgraded through higher-tiered plans

### Identify hard caps

Hard caps on scale are tool specific. Typically these can be found on the pricing website of the tool. A typical example of a hard cap is a limited data storage capacity. Airtable, for example, has a maximum of 50,000 rows per table even for the most advanced pro plan (before it is the enterprise plan with custom prices). Webflow has a maximum on their content-management-system of 10,000 CMS items for the most advanced business plan. This can be a limit to your scaling potential within a tool. Zapier has a dimension called update time or “delays”, which determine in what time intervals a workflow can get triggered. Even for the most advanced pricing the update time is at

least 1min. Thus, if you are building a tool for which real-time action is critical or becomes more critical with scale, this would potentially be a reason to not use Zapier.

To get certainty on whether or not this is a problem for you, first refer to the pricing website of the respective tool to identify the hard cap. Link these hard caps back to the purpose of your build: How long is your product supposed to last for? An MVP build to prove product-market-fit for the next year or to build part of your core product for the next two years?

In the example of maximum data storage capacity, search the maximum capacity listed for the no code tools you are evaluating (typically in price plan) and estimate whether you will hit that limit. For example, if you are building your core product for the next two years and you expect to grow to 5,000 users. An example calculation for Airtable (maximum of 50,000 rows per base) goes as follows: If one row represents one user, you are far from the maximum. However, if one row represents a session per user and you expect users to have on average 3 sessions per month you hit the maximum very quickly (50,000 rows < 5,000 users x 3 session x 12 months x 2 years). Hence, try to breakdown the estimated capacity by referring back to your timeline and expected users, content items posted, etc.

**Calculate the cost of scaling**

Some products can keep scaling easily but rapidly increase in price and, thus, running cost. An example of a steep increase in running cost from scaling that was mentioned in several interviews, was scaling access to Airtable, for example. While it is possible to increase the number of people accessing Airtable, the running cost increases *steeply* as it is priced per person per month.

Another example is Zapier. Zapier becomes more expensive the more “tasks” you execute per month. “Tasks” are the steps per workflow. Thus, total “tasks” per month is equal to # of workflows x # average number of steps x # average number of executions. As pricing is based (amongst other things) on the number of tasks, this can become very expensive very quickly if you keep expanding the number of processes of your product.

Thus, you will want to again refer to the pricing site of the tools, browse the features per plan, what the tool is priced on (e.g. per user per month vs. per month vs. other). Another useful action is to browse tool-specific forums to understand what scaling issues people have come across. Finally, similarly as for hard caps, calculate the running cost with your expected growth for the relevant dimensions starting from the

defined purpose of your build (as for hard caps). For more guidance on calculating running cost, please refer to Step 5.

**Evaluate when performance slows down**

Lastly, some tools start slowing down in performance and cannot be upgraded with higher-tiered price plans. These are often performance issues due to you reaching a scale that the given no code tool has not optimized for (though many no code tools are focusing on covering people at increasingly larger scales). These scale issues won't be "advertised" or shown on pricing sites, as these are not intentional limits. Thus, the best way to search for these types of performance issues is by combing through blogs, tool-specific forums or by asking no code developers working with the respective tool.

Generally, different tools will slow down at different scales and for different reasons. Though no code tools for similar categories (e.g. app builder vs. database) will slow down in performance due to similar factors (e.g. number of simultaneous users as slowdown factor for web apps). Summarized below is an overview of the most popular no code tool categories with an example tool being analyzed with respect to unintentional performance slow downs. As mentioned, while these are tool-specific, they can to some degree be generalized for the respective no code tool category.

For the example analysis of Bubble, the performance slowdown factors boil down to amount of traffic (e.g. simultaneous users) and complexity of site (e.g. database calls, API calls, data loaded on page load, etc.). More specific reasons for slowdowns in Bubble are customized data filters (for loading data on site), queries that are dependent on one another ("chained", query calls next query etc.), modifying the data on page load and lastly expensive calculations required for the site on-demand (as opposed to data being stored readily)[4].

Interestingly according to my interviews, for business automation tools like Zapier and Integromat (now called: Make), performance limits are more subject to hard caps (update time at least 1min) or tiered-pricing scaling ("tasks" per month) than unintentional performance slowdown. Thus, for business automation tools Zapier and Integromat scalability comes down to an upfront calculation of cost and/or pre-determined caps (update time speed) as opposed to potential user growth slowing down performance and consequently user experience.

**Overview of performance considerations by No Code tool type**

| No code tool category | Example | Performance slowdown factors | Rough heuristic of scalability (for selected example) |
|---|---|---|---|
| App Building (web & mobile) | Bubble | # of database calls, # of database queries, # of non-native integrations, # of simultaneous users, # of workflows run, Data amount on page load, Dedicated vs. non-dedicated hosting, Frequency of API calls | 100s-1000s of users simultaneously on app (depending on complexity of app) |
| Website Building | Webflow | # of database calls, # of database queries, # of non-native integrations, # of simultaneous users, # of workflows run, Data amount on page load, Dedicated vs. non-dedicated hosting, Frequency of API calls | n/a |
| Next Gen Spreadsheets / Databases | Airtable | # of database calls, # of non-native integrations, # of rows, Frequency of API calls | Slowdown from 20k rows onward (dependent on width of table too though) |

Examples of no code tool categories that have too heterogenous performance slowdown factors are “internal tools” or “AI & analytics”. Examples of no code tool categories where scalability generally does not affect performance much in terms of slowdown are  “checkout, subscription & memberships”, “form builders” and “product tours”.

A special note on server capacity: A frequent question no code freelance developers get from their clients is “Will this scale and will this hurt performance?”. The short answer is, if you can get to a scale where so many users ping your server and drive down

performance, you are probably in a position where you can afford hosting on a dedicated instance so that you can handle more users or where you can re-build in raw code with funding (”it’s a good problem to have, if you can even achieve that!”). In other words, too high traffic is usually not the bottleneck and the point where no code fails. Should you, yet, want to increase capacity of the server, hosting a dedicated server instance can increase performance, if setup near a target geography or for running heavy data operations. Founders I interviewed estimated that hosting on a private cloud can provide significant performance boosts at 2x cost.

## Gauge your design flexibility needed

Design flexibility is another limitation some founders and no code developers mentioned. Generally design requirements are rising continuously due to consumers getting used to rising standards. Specifically, if high control over the UI is important, it’s important to test if the design flexibility is given in your tool of choice. Some tools allow for more flexibility than others. For example, Glide (app builder) allows for only basic colors and selected icons. Workarounds like adding a photo of an icon can work, but can lead to worse user experience. Bubble is relatively more customizable and is also expanding customization options. Yet, many of these tools have a characteristic “touch and feel” that can be recognized.

A way to test whether design capabilities of the no code tool meet your expectations is to build a feature that you have most “design uncertainty” over (e.g. intricate table of tables) and determining whether this is satisfactory for your product.

This should typically not be a problem for validating an idea (pre-seed) or product-market fit (seed), but if you are building your core product in no code and high UI control matters a lot this could be an issue.

An example for this was a CTO I interviewed in East Africa, whose company had very specific UI requirements. Their product had to look familiar to non-smartphone interfaces that people already know. Thus, they built their UI in raw code but everything else in no code.

## Step 5: Evaluate the running cost

- **Evaluate the cost of your tools “What is the total cost of ownership for one month/year?”:** When considering the tools based on cost, it can get complicated as tools price on different variables. For example, Airtable prices per user per month, Webflow prices per month for given bandwidth of the CDN (content distribution network) and fixed # of guest editors.  Zapier prices (mainly) on tasks executed per month (=# of workflow x # of avg. tasks / workflow x # of avg. executions per workflow per month).

  Thus, again it is good to think back to the purpose of your build and given timeline. Are you trying to build to validate an idea (few months), find product-market-fit (few months) or build your core product (1-2 year horizon)? Estimate relevant parameters for that given timeframe: This can include # of users (to estimate storage capacity needed), your team size, amount of content posted (to estimate CMS storage capacity, for example), etc. Number of team members growing can be the biggest driver of your the cost of your no code costs scaling more than anticipated.

  Estimating this for your given timeframe, can help you calculate the “total cost of ownership” either for a month (earlier stage) or year (later stage/longer term) for your different options of tech stack you are considering. The longer-term you are thinking, the more this price comparison is going to matter to you in your decision making around tech stack.

- **Evaluate maintenance cost:** A rather hidden fact of no code is that maintenance cost of a no code tech stack can be literally zero. A CTO I interviewed initially built the company’s product with raw code and no code first and then completely rebuilt it fully in no code (besides UI) when it had 100 of users.

  This team’s no code experience was that making changes to the code (e.g. changing name of column in table), as your product becomes more complex, can start becoming slower in no code than in code. However, while changing something small in no code can be slower, there is less time that needs to be spent on environment maintenance (like staging, etc.) or complicated deployment infrastructure and processes (e.g. CI/CD).

  For example, changing the names of columns in one table that is used at different parts of the product, can take long to change in no code (a lot of clicking). But once

it has been changed, it has been changed at all parts of the product and has no risk of going into production unlike for raw code.

## Step 6: Evaluate development costs

The main question for development is do you want to build the product yourself or hire a no code developer? A few considerations to think about for either path you choose:

- **Hire no code developer:** Hiring a no code developer can be a really good option, if you feel your time is better spent focusing on user acquisition / community building, if you have the money and are willing to invest it or if you feel learning to build in no code does not seem worth the time.

  A good way to approach this can be to use your simple pencil and paper wireframes and request quotes from no code developers. Often no code developers do this upfront for free.

  What you may find is that no code developers charge higher hourly rates than raw code developers (obviously highly dependent on experience and quality). This may lead you to think you should hire raw code developers. However, do consider that no code developers can typically build faster (depending on product), meaning they need less hours, and can get and implement user feedback more quickly. So especially for early stages this can be a real advantage when building your product in no code. This does not mean no code is the right answer and raw code is not. But you should consider this advantage.

  Lastly, you should likely be comfortable to pay 1.5-2x the quote you were given. As you build changes happen, the requirements can change a lot. So be prepared for the final development cost to vary before deciding on this option.

- **Develop yourself:** Building yourself has the functional outcome of building your product. But you are also very likely to learn a lot about the basics of how the internet (e.g. to understand how websites are built and what their frameworks are) and databases (essentially spreadsheets) work.

  Secondly, if certain tools seem more complex and intimidate you, this can be a very valid factor to consider a more beginner friendly no code tool to build your MVP. Bubble, for instance, is far harder for people to build their first app in than Adalo.

> Lastly, almost all founders I interviewed said that it was surprisingly hard and sometimes frustrating to build in no code and that “it is also very easy” and “everyone can do it” within one interview. This may seem paradoxical, but I believe this is what it means: It takes grit to keep trying, ask in forums, find a mentor and the process of building can be far more messy than it may sound here. But ultimately if you are curious to try and explore, it can also be a very valuable long-term skill you develop to quickly prototype in the future.

Diving into no code and building your first own product as a one-person team can be very empowering. No code founders, moreso, than “status-quo” founders are often solo founders, as I found in my analysis of 100 no code startups. As you’ve seen building a first product involves many different aspects: Customer discovery, basic user interface/experience skills (as seen in section X), developer skills and sales/business development. No code can really help people, who are curious about a broad set of skills succeed, and lower that barrier to entry to building your product. For more mature products and teams, it can really help keep the team stay lean for longer and decrease the number of touch points (e.g. less meetings, less potential error in communication) needed for a team to stay aligned and quick. In either stage - prototype-build or product-build - no code is shaking up old assumptions and shaping how startups are being built.

**Sources**


1. *Gartner forecasts worldwide low-code development technologies market to grow 23% in 2021*. Gartner. (n.d.). Retrieved May 2, 2022, from https://www.gartner.com/en/newsroom/press-releases/2021-02-15-gartner-forecasts-worldwide-low-code-development-technologies-market-to-grow-23-percent-in-2021
2. Tommasi, M. (2021, August 8). *No-code: Democratizing software*. Medium. Retrieved May 3, 2022, from https://medium.com/@mtommasi94/no-code-democratizing-software-9421c2208cb
3. Kaya, N. (2018, January 24). *Why you shouldn't skip your wireframing*. Medium. Retrieved May 3, 2022, from https://blog.prototypr.io/why-you-shouldnt-skip-your-wireframing-1f7a70d5c125
4. *General Principles & Tips About Performance*. Performance & Scaling - Bubble Docs. (n.d.). Retrieved May 5, 2022, from https://manual.bubble.io/help-guides/optimizing-an-application/performance-and-scaling