Key Things to Remember When Picking the Tech Stack

In this lesson, I'll share a few key things to keep in mind when researching a technology stack for our application.

We'll cover the following

- Be thorough with the requirements
- See if what we already know fits the requirements
- Does the tech we have picked has an active community? How is the documentation and the support?
- Is the tech being used by big guns in production?
- Check the license. Is it open source?
- Availability Of skilled resources on the tech

I've written numerous projects from the ground up and spent countless hours scavenging the web and browsing through technologies and frameworks to pick the right tech that would align with my requirements.

I have wasted days, if not months, trying to implement stuff with a not so fitting technology. I have implemented tech stuff having no support and community and have later cried for days into a pillow.

Picking the right tech stack is crucial for our business's success. There is no way around it, and I think we all understand this fact pretty well. Once we pick the tech and get down to coding, there should be no looking back. We naturally can't afford it.

The guidelines listed below are the gist of my experience, and they are the most important factors in the process of picking the right technology stack.

So, without any further ado, let's get started.

Be thorough with the requirements#

We should be crystal clear on what we are going to build. Things should not be hazy. We cannot pick the right tech if we are unclear about the requirements. When we go hunting, we should be clear about what we are hunting for.

For instance, when looking for a database, we should be clear about if we are going to store relational data or if it will be document-oriented, semi-structured, or with no structure at all.

Also considering, are we handling quite a large amount of data that is expected to grow exponentially? Or, the data is expected to grow at a manageable pace up to a certain limit?

Will a *monolithic architecture* serve our requirements well or do we need to split our app into several different modules?

Splitting the app into several modules, using heterogeneous tech in services, helps us bail out on a particular technology in case things don't work out.

See if what we already know fits the requirements#

It's easier to build new applications with tech we already know. We don't have to go through the steep learning curve that comes along with the new tech.

Things are also comparatively clearer when using tech we are well familiar with. Being aware of the nitty-gritty, having familiarity with the errors, exceptions, and the knowledge to fix them helps us release the features at a quick pace.

Avoid running for shiny new toys until you really need them. Do not fall for the hype.

Imagine an exception thrown by a new tech that you haven't seen before ever in your life and also cannot find the solution online. You are stranded. There is no one to help you, and all you hear is crickets.

I've been there, done that. It's frustrating, clicking through all the search result pages of Google and finding nothing.

Does the tech we have picked has an active community? How is the documentation and the

support?#

The technology we pick ought to have an active community. Check the involvement of the community on *GitHub*, *StackOverflow*, etc. The documentation should be smooth and easy to comprehend.

The larger the community, the better. Having an active community means updated tools, libraries, frameworks, and whatnot.

If there is official support available for the tech, there should be some rescue available if we get stranded down the road, right?

Is the tech being used by big guns in production?#

If the tech we are picking is being used by big guns in the industry, this confirms it is battle-tested and it can be used in production without any worries.

We can be certain that down the line we won't face any inherent scalability, security, or other design-related issues with the technology. Since, the codebase is continually patched with new updates, bugs, and design fixes.

We can go through the engineering blogs of the companies to get more information about how they have implemented the tech.

Check the license. Is it open source?#

Picking an *open-source* technology helps us write our own custom features in case the original solution does not have it. We do not have to rely on the tech's creator for new features and stuff.

Also, in terms of money, we don't have to pay anyone any sort of fee to use the product. *Open-source* tech also has a larger community since the code is open to all. This way anyone can fork it, start writing new features or fix the existing known bugs.

Availability Of skilled resources on the tech#

Once our business starts gaining traction, we would need a hand to move at a quick pace and roll out new features within a stipulated time. It's important that there are enough skilled resources available in the industry on the technology we pick.

For instance, it's always easy to find a MySQL administrator or a Java developer as opposed to looking for a resource skilled on comparatively newer technology.

Well, this concludes the lesson. Let's move onto the next.

← Back

How to Pick the Right Server-Side Tec...

Next →

A Web-Based Mapping Service Like G...

Mark as Completed

Report an Issue