**How to Pick the Right Server-Side Technology?**

In this lesson, we'll learn how to pick the right server-side technology for our projects.

**We'll cover the following**

* [Real-time data interaction](https://www.educative.io/module/lesson/web-application-architecture-101/JYM0xJLNn5P#Real-time-data-interaction)
* [Peer-to-peer web application](https://www.educative.io/module/lesson/web-application-architecture-101/JYM0xJLNn5P#Peer-to-peer-web-application)
* [CRUD-based regular application](https://www.educative.io/module/lesson/web-application-architecture-101/JYM0xJLNn5P#CRUD-based-regular-application)
* [Simple, small scale applications](https://www.educative.io/module/lesson/web-application-architecture-101/JYM0xJLNn5P#Simple,-small-scale-applications)
* [CPU and memory intensive applications](https://www.educative.io/module/lesson/web-application-architecture-101/JYM0xJLNn5P#CPU-and-memory-intensive-applications)

Before commencing the lesson, I would like to state that there is no rule of thumb that says that for a use case X, we should always pick a technology Y.

Everything depends on our business requirements. Every use case has its unique needs. There is no perfect tech, no silver bullet. Everything has its pros and cons. You can be as creative as you want. No rule holds us back.

With this being said, in the lesson, I have listed some of the general scenarios, or standard use cases, in the application development domain. And then, I’ve discussed what backend technology fits best with those use cases.

**Real-time data interaction**

Imagine building an app that needs to interact with the backend server in real-time to stream data to and fro. For instance, your app could be a messaging application, a real-time browser-based massively multiplayer game, a real-time collaborative text editor, or an audio-video streaming app like Spotify, Netflix, etc.

To implement this, we need a persistent connection between the client and server. You also need non-blocking technology on the backend. We’ve already talked about both concepts in detail.

Some of the popular technologies that enable us to write these apps are NodeJS, Python’s Tornado framework. If you are working in the Java ecosystem, you can look into Spring Reactor, Play, and Akka.

Once you start researching these technologies and go through the architecture concepts given in the developer docs, you’ll gain further insights into how things work and what other tech and concepts you can leverage to implement your use case. One thing leads to the other.

Uber used Node.js to write their core trip execution engine. The technology enabled them to manage a large number of concurrent connections without much fuss.

**Peer-to-peer web application**

If you intend to build a peer-to-peer web app, for instance, a P2P distributed search engine or a P2P Live TV radio service, something similar to LiveStation by Microsoft, look into JavaScript protocols like DAT and IPFS. Also, checkout FreedomJS, which is a framework for building P2P web apps that work in modern web browsers.

**CRUD-based regular application**

Regular CRUD-based apps include simple use cases such as a tax filing app, employee management, employee attendance systems, etc.

Today, CRUD (Create Read Update Delete) is the most common form of web app businesses build. Be it an online booking portal, an app collecting user data, or a social site. Most of them have an MVC (Model View Controller) architecture on the backend.

Some of the popular technologies that help us implement these use cases are Spring MVC, Python Django, Ruby on Rails, PHP Laravel, and ASP .NET MVC.

**Simple, small scale applications**

If you intend to write an app that doesn’t involve much complexity like a blog, a wordpress plugin, a simple online form, or simple apps that integrate with social media platforms running within the IFrame of the website, these include web browser-based strategy airline and football manager games. You can pick PHP.

PHP is ideally used in these kinds of scenarios. We can also consider other web frameworks, like Spring boot and Ruby on Rails, that cut down the verbosity, configuration, and development time by notches and facilitate rapid development. However, PHP hosting will cost much less compared to hosting other technologies. It is ideal for very simple use cases.

**CPU and memory intensive applications**

Do you need to run CPU-intensive, memory-intensive, heavy computational tasks on the backend, such as Big Data processing, parallel processing, and running monitoring and analytics on quite a large amount of data?

Performance is critical in systems running tasks that are CPU and memory-intensive. Handling massive amounts of data has its costs. A system with high latency and memory consumption can take a toll on the economy of a company.

Also, regular web frameworks and scripting languages are not meant for number crunching.

Tech commonly used in the industry to write performant, scalable, and distributed systems are:

C++ has features that facilitate low-level memory manipulation. This provides more control over memory to the developers when writing distributed systems; most of the [cryptocurrencies are written using this language](https://en.wikipedia.org/wiki/List_of_cryptocurrencies).

Rust is a programming language similar to C++. It is built for high performance and safe concurrency. Lately, it’s been gaining quite some popularity amongst the developer circles.

Java, Scala, and Erlang are also good picks. Most of the large-scale enterprise systems are written in Java. Elasticsearch is an open-source real-time search and analytics engine is written in Java.

Erlang is a functional programming language with built-in support for concurrency, fault-tolerance, and distribution. It facilitates the development of massively scalable systems. This is a [good read on Erlang](https://stackoverflow.com/questions/1636455/where-is-erlang-used-and-why).

Go is a programming language by Google to write apps for multi-core machines handling a large amount of data.

Julia is a dynamically programmed language built for high performance, running computations, and numerical analytics.

Well, Folks, this is pretty much it. In the next lesson, we’ll talk about a few key things to bear in mind while researching on a fitting technology stack for our project.

# 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 on a technology stack for our application.

**We'll cover the following**

* [Be thorough with the requirements](https://www.educative.io/module/lesson/web-application-architecture-101/gkvBnk2Ly63#Be-thorough-with-the-requirements)
* [See if what we already know fits the requirements](https://www.educative.io/module/lesson/web-application-architecture-101/gkvBnk2Ly63#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?](https://www.educative.io/module/lesson/web-application-architecture-101/gkvBnk2Ly63#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?](https://www.educative.io/module/lesson/web-application-architecture-101/gkvBnk2Ly63#Is-the-tech-being-used-by-big-guns-in-production?)
* [Check the license. Is it open source?](https://www.educative.io/module/lesson/web-application-architecture-101/gkvBnk2Ly63#Check-the-license.-Is-it-open-source?)
* [Availability Of skilled resources on the tech](https://www.educative.io/module/lesson/web-application-architecture-101/gkvBnk2Ly63#Availability-Of-skilled-resources-on-the-tech)

I’ve written numerous projects from the ground up and spent countless hours scavenging the web, 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 written apps with tech 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 in almost a decade of application development. These factors are the key to picking the right technology stack for our app.

So, without further ado, let’s get started.

## Be thorough with the requirements[#](https://www.educative.io/module/lesson/web-application-architecture-101/gkvBnk2Ly63#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 on if we will store relational data or will it be document-oriented, semi-structured, or have no structure at all.

Also, do we have to deal with quite a large amount of data expected to grow exponentially? Or is the data 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 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.

I cannot assert this enough. We cannot take a step forward if we are unclear on our business requirements.

## See if what we already know fits the requirements[#](https://www.educative.io/module/lesson/web-application-architecture-101/gkvBnk2Ly63#See-if-what-we-already-know-fits-the-requirements)

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

Things are also comparatively clearer when using the 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?[#](https://www.educative.io/module/lesson/web-application-architecture-101/gkvBnk2Ly63#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. An active community means updated tools, libraries, frameworks, and everything.

See if there is any 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?[#](https://www.educative.io/module/lesson/web-application-architecture-101/gkvBnk2Ly63#Is-the-tech-being-used-by-big-guns-in-production?)

If the tech we intend to pick is being used by big guns in the industry, this confirms it being battle-tested and okay to 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, as the codebase is continually patched with new updates, bugs, plugins and design fixes by the technology owner and the engineers of the big companies.

## Check the license. Is it open source?[#](https://www.educative.io/module/lesson/web-application-architecture-101/gkvBnk2Ly63#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 every now and then.

Also, in terms of money, we are not liable 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 and start writing new features or fix the existing known bugs.

## Availability Of skilled resources on the tech[#](https://www.educative.io/module/lesson/web-application-architecture-101/gkvBnk2Ly63#Availability-Of-skilled-resources-on-the-tech)

Once our business starts gaining traction, we would need a hand to roll out new features within a stipulated time, moving at a quick pace. It’s essential 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 in comparatively newer technology.

Well, this concludes the lesson. Moving on to the next.