Architecting an application and deciding on the key components of the technology stack that go into building it is an important task. If software hadn't worked out as a career, my backup was construction. There are many similarities – from crafting the blueprints to laying the foundation, from choosing your materials to building the structure – that help explain the process of planning your tech stack.

The thought that goes into this is similar to the thought that goes into purchasing and building a house. You want to make the choices that will sit well with you for years to come because this is a big decision. However, it's equally important to put yourself in a position to be flexible. Things change. It's important to recognize that you will want to remodel at some point.

Identify the Essentials of Your Tech Stack

The difficult process of planning out a project is twofold: you have to understand what you want, and you have to understand what you need.

There is certainly overlap between these two considerations. Maybe you want to take a minimalistic approach to save time and money down the road. But you'll likely find yourself frustrated in the end if your needs aren't met – a house that doesn't have a bathroom upstairs or lights in the family room leaves something to be desired.

Planning your technology stack is no different. It's important to think about the future. But the danger lies in focusing too narrowly on it rather than making sure your stack is adaptable to changes in the development process. The common solution to these two related problems lies in (1) Identifying the essentials, and (2) Making the architecture as modular as possible so you can interchange – or remodel – key components if the need arises.

Some of the most common pitfalls we've witnessed are making decisions that force developers into a specific technology or into relying on a tool that limits their ability to expand in the future. Just like building a garage that fits only one car is limiting, building an application database access layer in a way that only works with one specific database engine can be extremely costly and risky.

What if you want more than one car in the future? What if you get a different type of car that's larger in size?

Relating this back to tech stacks, what if your success necessitates leveraging more than one engine? What if the situation lends itself to researching the potential wins of a new engine altogether?

If you're too far down the rabbit hole, it can be difficult to persuade stakeholders to make a decision when the costs to continue are climbing steadily. At that point, you will have lost out on some potential opportunities for improvements, undoubtedly hitting the morale of your team on the chin.

We recommend thinking of your application stack like a LEGO® kit. Ultimately you're going to build yourself a castle, but each brick – placed with thought and care – has its own unique purpose.

What Ecosystem Best Fits Your Business?

Consider this hypothetical application as an example.

Imagine you're building an application that will have an API and a web application. The stakeholders plan to expand to native mobile applications on iOS and Android later, so you need to make sure you can support those needs in the future.

First, you'll need to decide on the high-level architecture decisions. Do you want to use one of the major cloud service providers like Rackspace, AWS, or Azure? *Hint:* we recommend doing so!

Also, what type of services will you be using for deployment, continuous integration, testing, content delivery, security, monitoring, and logging? Chances are you're overwhelmed by the variety of important decisions you have to make!

Take a step back and simplify all those choices first.

What neighborhood do you want to live in – what tech stack ecosystem best fits your business? This is important because this decision will reverberate through the entire business. While it's great to be as flexible and modular as possible, remaining agnostic to technology is increasingly difficult once you've settled into a particular community. Once you've bought your house and moved in, it's challenging to physically move your house to the neighborhood on the other side of the river.

A housing community has standard conventions – your tech stack will as well. Grocery stores, hardware stores, and the local diners exist in any community. What you need to consider is if you want the specialty stores that one community has over the other.

For example, Amazon Web Services (AWS) has AWS Lambda which Microsoft Azure (as of the writing of this article) does not yet have a comparable tool. If real-time serverless functions can benefit the design and implementation of your product, then AWS has a real advantage. However, if your internal office environment is using Microsoft Windows and other Microsoft tools, Azure may be a seamless integration for your business.

Four Primary Decisions You Need to Make

Focusing on the web application for brevity, we can think of the first four primary decisions around the hypothetical application as follows:

With this approach, and a modern application architecture, you can interchange any of these components later if you found one benefits your business more than what you originally chose.

Let's say that you decided to use PHP, MySQL, Material Design, and AngularJS:

With those decisions made, your next decisions hinge on the frameworks or integrations of those technologies. Will you be using AngularJS 1.5 or 2.0? Will you be using Google's

Material Design, or a spin-off approach? Are you going to use a PHP framework like Symfony and Laravel? And are you going to use MySQL or MariaDB or Percona?

These are the decisions that end up being most important because they are the ones that can potentially weld themselves together down the road.

Essentially what you should always strive for is to be as agnostic as you can be while still leveraging the technology best fit for your particular application or product. Given this example, while you chose MySQL initially, you would be best prepared for future growth and/or changes by making certain your API does not explicitly require MySQL to function.

The same goes for AngularJS and Material UI. AngularJS may be your preferred solution now. But things change. If they do, and you don't anticipate how your technology might need to change as well, it can be extremely challenging to switch from AngularJS, Material UI, or whatever technology your stack is built upon.

Conclusion

Architecting an application and deciding on the key components of the technology stack that go into building it is an important task. That's a given. But crafting the blueprints, laying the foundation, choosing your materials, and building the structure is more complicated than selecting items off a menu.

We argue that understanding the essential components of your tech stack, the ecosystem that best fits your business, and how all of these elements will impact each other in the end, is crucial.

Your application will change and grow, so keep a future remodel in mind. Understanding what you need is just as important as understanding what you want, and for this reason, we recommend investing time and resources in choosing the best tech stack for your application.

Originally published at Fresh Consulting