Technology Stacks

This is CS50. Harvard University. Fall 2015.

Cheng Gong

Table of Contents

1. ACT.md

- We watch a video that introduces ACT.md, a company that builds a platform that's like "project management for patients."
- The term **technology stack** just means all the technology used to build or develop a product, including frameworks, interfaces, and languages.
- Dr. Patrick Schmid, from ACT.md, joins us today to talk about tech stacks based on his experience.
- First we walk through the product through screenshots, where we see the perspective from a healthcare provider's point of view and a list of patients (along with their status and pending tasks) is shown, and then a patient's dashboard, which shows their messages, providers on their team, and what they can expect.
- Finally, there is an overall dashboard from an administrative point of view that shows all providers and patients with aggregate data over time, to help guide decisions.
- A website like this might need various components, so we ask the class to suggest questions:
- where to begin, high level or low level?
  # The founders had a vision of the product/idea, so they worked on the highest level, the user experience and solving some problem. And to prioritize features/work, they talk about what users need most and also what the company internally might need for the future. For ACT.md, they focus on bigger ideas. For example, if users want the ability to fill out a form, then they would want to think about the bigger goal of collecting data and analyzing data, and how they might accomplish that. And the first features helped guide the first version of the product, and basic blocks needed.

- what's the database's structure
  - # The tables and data models are based on what the app needed, so there might be one for patients, one for tasks and deadlines, etc.
- where is app located and where is it running?
- what OS, languages, back end, etc.? how to decide among those? tradeoffs?
  - # ACT.md uses Ubuntu, a version of Linux, and one main reason for choosing that, and other aspects, ends up being that Patrick was just more familiar with it at the beginning, so it was most efficient to start using right away. And Red Hat Linux and Windows tend to be used by many enterprises, but Windows costs money, and the free and open source Linux distributions happened to be supported just as well for the services they needed. Languages to choose from might include Ruby, Python, Java, PHP, all of which tend to be run on Linux machines, whereas a language like C# is better supported on a Windows machine. So ACT.md chose Python because Patrick is most familiar with it, and these days almost any application can be written in any language, but the tradeoffs are the frameworks that each language has, and how well developers know it. The front-end is just JavaScript, so their stack differs from a framework like Ruby on Rails or Django where all the code to generate HTML is written in one language. Instead, they have a Python back-end which only outputs data in the form of JSON (objects in JavaScript notation), and the front-end retrieves this data and displays it. And another benefit is that JavaScript is supported by devices that have a modern browser, but if someday they have a mobile app in another language, it can just use the same Python back-end to retrieve data. And all of these languages exist because they each support some features differently or are faster in certain ways.
- where is data stored?
  - # They use a third-party service called Armor which hosts the data subject to the requirements needed for health data. There are other services like Amazon's AWS, Microsoft's Azure, and Google's App Engine, but Armor monitors their data access for increased security.
- what devices will users be using to access this tool?
- how keep data secure
  - # Other companies might segregate data by organization, where each location has a separate server for its data, but ACT.md has users moving between locations

or collaborating across locations. At a high level, they use an access control list which lists accounts and groups and data they can each access. And to make their infrastructure secure, they have a service to try various attacks on their servers, even after given some username and password. And finally, all the code that is written is reviewed and tested by someone else on the team. They test features manually and also use tools that automatically test the application. Python has a built-in testing framework, and the front-end uses tools called Behave and Selenium, which reads in code and actually interact with web browsers to simulate clicks, etc. Finally, they need to test the app across many browsers and versions. Even though it seems like it takes a lot of time to thoroughly test, it's worth it because it makes sure everything works, even later on when new features are added. And there are at least three different environments that run the application: development, which could be developers' own computers or a server in the office; staging, which is like a testing server; and production, which is actually used day-to-day. Because hospitals tend to be a more traditional setting, ACT.md releases new features on a more predictable cycle, but companies can deploy new code as often as they are done.

- how do you find talented developers to work on your team?
  - # At first, the company was founded by people who have all been involved with the problem of managing patients and providers, and eventually they just look for people who believe in the mission but are reachable through a network like LinkedIn and AngelList.
- how do you prioritize features?
  - # So instead of analyzing how much a feature would cost in terms of money, they use how much time it might take to build it, and how much value it would create in bringing in new customers or keeping existing ones happy. Faxing, for example, is an important criteria that was not too difficult since there are existing services that send faxes. Other features take much more time, but whether or not it is worth it, ends up being debatable.
- One mistake might be spending time making features that aren't used as much by customers, and also when business-side people manage technical-side people too much. And since they can release often, they can build smaller parts of features incrementally, to help guide estimating time needed to complete some feature.
- And some questions we didn't get to:
  - # have you had a SQL injection attack (that you can disclose on camera)?

what sort of tools are you building in-house as opposed to using third-party?

redundancy and uptime requirements?

how to integrate with hospitals, etc.?

how do usability testing (esp. with multiple types of users)?

using any APIs?

how many users / how much traffic are you designing for?

when you scale, which frameworks did you put in place today for the future?