Ask Technically #2

Microservices, object oriented programming, and browser compatibility

Justin
Jun 21, 2022

18

Hello dear paid subscribers, and welcome to the **second issue of Ask Technically**. A few weeks ago, I asked you all for burning questions about software, hardware, and everything in between and the response was overwhelming! I've got 50+ questions to work my way through, so we'll tackle 2-3 every issue to make sure everyone gets a quality and thorough answer.

Recall that Ask Technically *of course* does not replace the exclusive paid content (deep dives, company breakdowns) that you signed up for; it's just a new format I'm adding into the mix to make sure you're getting even more out of your subscription. If you missed the last paid post about production databases, you can check it out here.

If you have any questions of your own, just reply to this email or send them to justin@technically.dev.

Sam asks…**wtf are microservices, how are they different from monolithic applications, how does it relate to REST + GraphQL?**

Great question. I'll be writing a whole post about this soon, but I'll cover it here quickly for those curious. In short, microservices describes the idea of breaking up your application into smaller pieces instead of building it as one giant piece of code.

At its core, an application is a system of interacting little services. Each service does a specific thing:

- The frontend is a service
- Your API endpoint for creating a new user is a service
- Your database is a service

- Your API endpoint for kicking off the invoicing process is a service

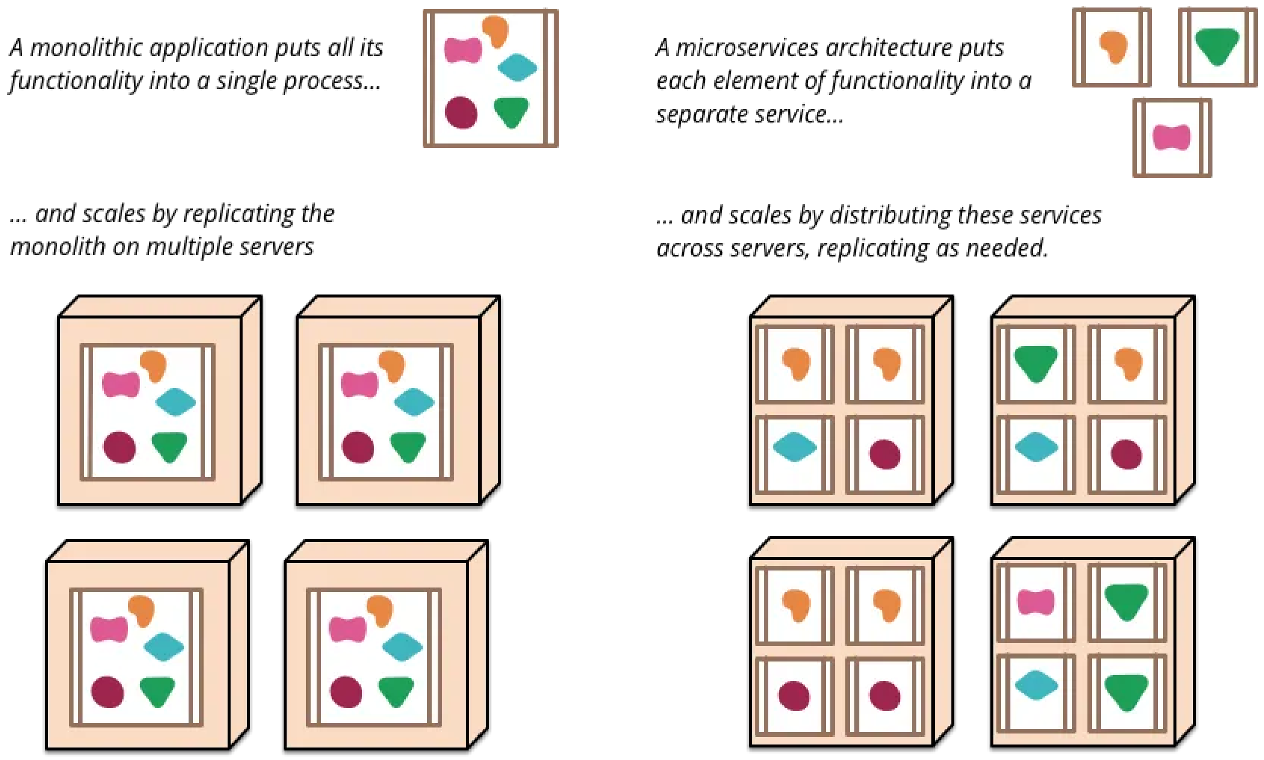
Traditionally, all of these things were written as a giant singular codebase (imagine everything being in one big folder on your desktop), and deployed on a single, really big server. In technical terms, this style of design (or architecture) is called a **monolith**.

For smaller applications, building a monolith (despite the name) makes a lot of sense. The Technically site is a monolithic application. It's the simplest and fastest way to build!

But as applications (specifically backends) get more complex, having *all* of your services in one place can become dangerous:

- If one service goes down, it takes all of the others with it
- If one service eats up a lot of processing power, the others will get slower
- Each service is different and may require different types of infrastructure (e.g. memory heavy, need a GPU, etc.)
- It's impossible to scale up one service independent of the others

To deal with these problems, developers will instead build and deploy each service as its own little backend with its own little server or set of servers. That way, each can operate independently, scale up or down independently, fail independently, etc. This diagram from Martin Fowler's post on microservices is fantastic:


A monolithic application puts all its functionality into a single process...
A microservices architecture puts each element of functionality into a separate service...
... and scales by replicating the monolith on multiple servers
... and scales by distributing these services across servers, replicating as needed.

While splitting up your app into many smaller services *does* make it easier to scale and more fault tolerant, it comes with a big cost: **complexity**. Now instead of one server to manage, you've got a distributed system of many. So like most infrastructure decisions in software engineering, building a monolith or a microservice architecture is a nuanced decision that comes with tradeoffs for each.

(Microservices, like Kubernetes, is a great way to make fun of developers for over-optimizing everything. See here.)

Spencer asks…**I heard Steve Jobs mentioned object oriented programming. What is that, and how does it relate to what Apple does?**

I remember reading one of the Steve Jobs biographies and being told that he was a terrible programmer. Anyway…

At its core, programming and writing code is the managing and use of data. Though there are hundreds (thousands?) of programming languages, they can all be *broadly* classified into two philosophies of how that data should be created and managed. Note that this is pretty in the weeds, so if you don't follow the details that's fine.

1. **Object oriented programming**: centered on the object model, where data and behavior is mixed together
2. **Functional programming**: centered on the function model, where data and behavior are separated

You're probably wondering what the fresh hell any this means, so let's illustrate with a simple example. Imagine you wanted to implement the concept of an apartment in your code (maybe you're a StreetEasy engineer or something).

In **object oriented paradigms**, you'll implement your apartment as an **object**, with properties (things it *is*) and methods (things it can *do*). This whole big thing is usually called a class.

```
class Apartment

  @property num_bedrooms
  end

  @property num_bathrooms
  end

  @method doRenovations
    num_bedrooms += 1
    num_bathrooms -= 1
  end

  @method get_sold
  end
```

This class is like the *blueprint* for an apartment on StreetEasy, and each actual apartment listed is a particular *instance* of that blueprint (like a mold making sandcastles at the beach). Note how the apartment's **data** (properties) and **behavior** (methods) are tied together. They both get defined in the class, and the methods actually change (mutate) the object's properties. For example, we might create an apartment with three bedrooms and three bathrooms:

```
const newApartment = new Apartment(3,3)

print(newApartment.num_bedrooms)
// should print 3
```

And then do some renovations that actually *change* the number of beds and baths in that instance of the apartment:

```
newApartment.doRenovations()

print(newApartment.num_bedrooms)
// should print 4
```

In **functional programming** though, data and behavior are separated. Instead of the `Apartment` class having properties and methods, it would just be a bunch of data like so:

```
const newApartment = [3,3]
```

And we'd have a separate, global method for doing renovations:

```
function doRenovations(apartment, new_num_bedrooms)
```

It's pretty hard to explain further than this without getting unduly in the weeds – an engineer might find our explanation "too surface level" but hey. Suffice it to say that object oriented programming is one of the two major philosophies on how to code, and it's the most popular one.

In terms of how this relates to Apple, it doesn't, at least not directly.

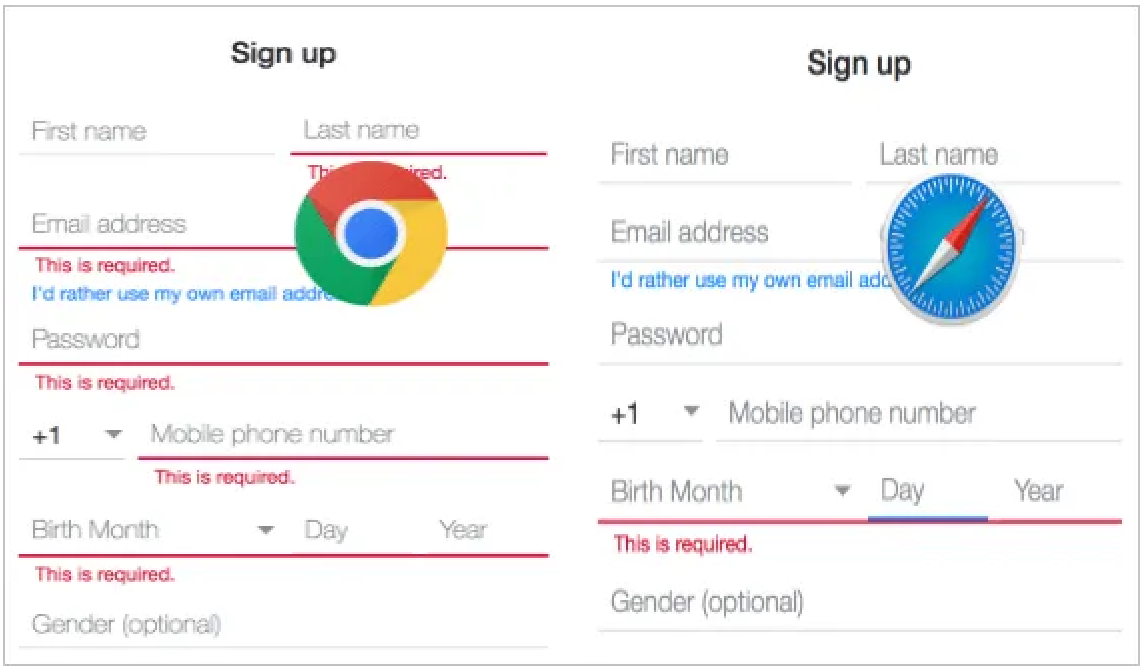
Benjamin asks...**how do you handle the compatibility of your app with different browsers, especially old ones?**

Browser compatibility is one of those problems that developers really hate to deal with, it's honestly a bit of a meme at this point. So what is it exactly?

As discussed ad infinitum in this newsletter, applications are made up of frontends and backends. In a web application, the frontend has 3 components:

1. **HTML** for page structure and elements
2. **CSS** for styling
3. **JavaScript** for interactivity

Every web page you use or see (perhaps even the one you're reading this on now) is made up of these 3 things. The *problem* is that every single browser – both in terms of the browser itself and the version – interprets these languages slightly differently. And it's incredibly annoying.



source

Developing for the web requires constantly asking yourself:

- Does this browser support this library I'm using?
- Does this browser render this object correctly?
- Does this browser support my version of JavaScript?
- How does this browser show this animation I want to run?

The problem is compounded by the fact that there are several major browsers (Chrome, Firefox, Safari, etc.) and they have hundreds of different versions. Mozilla's docs do a good job of explaining the topic more in depth; the TL;DR is that you'll never be able to support 100% of browsers, but you can make intelligent prioritization decisions.

Beyond what's documented, the only way to really know how your app is going to work across browsers is to **test it**. There are entire companies that exist to make cross browser testing easy, like Browserstack and LambdaTest . In the open source world, Babel is a popular JavaScript tool that can compile your JS to run better across different browsers.

If you have any questions of your own, just reply to this email or send them to justin@technically.dev.

Comments

Write a comment...