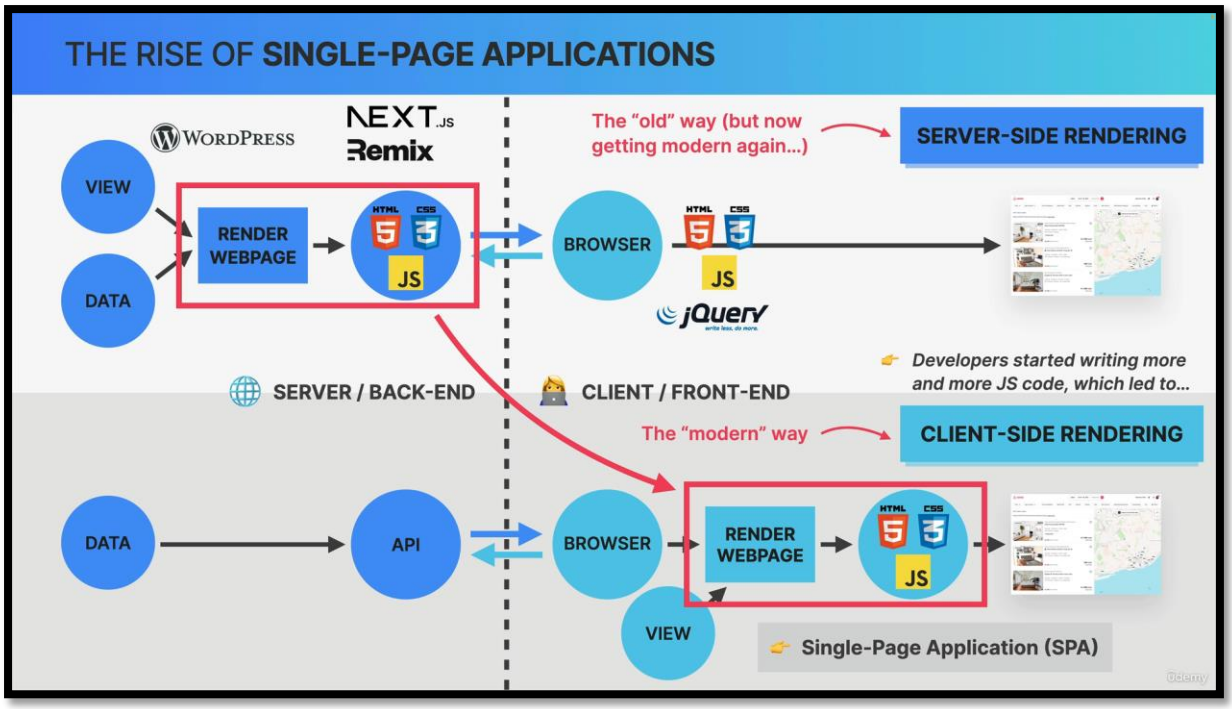


Before around 2010 all websites were always rendered on the server. So, in server-side rendering a website is basically assembled on the backend. So, on a web server, based on data and templates. The resulting HTML, CSS and JavaScript code is then sent to the client side, so to the web browser that requested the page. The browser then simply takes this code and paints it onto the screen. A typical example of server side rendered websites are all websites built with WordPress.

The JavaScript that used to be included in these websites was initially only to add some simple dynamics to the page, like simple animations, hover effects, and other stuff like that and usually a very popular library at the time called jQuery was used for this because it made JavaScript work the exact same way across all browsers back then.

However, over time, developers started writing more and more JavaScript code to be executed by the browser, until at some point these became fully fledged web applications, which then led to the rise of so-called single page applications.

So, these are basically webpages that are rendered on the client, not on the server. So, in client-side rendering, basically the work of rendering a webpage is shifted from the server to the client. So, now we don't call these webpages anymore but web applications.

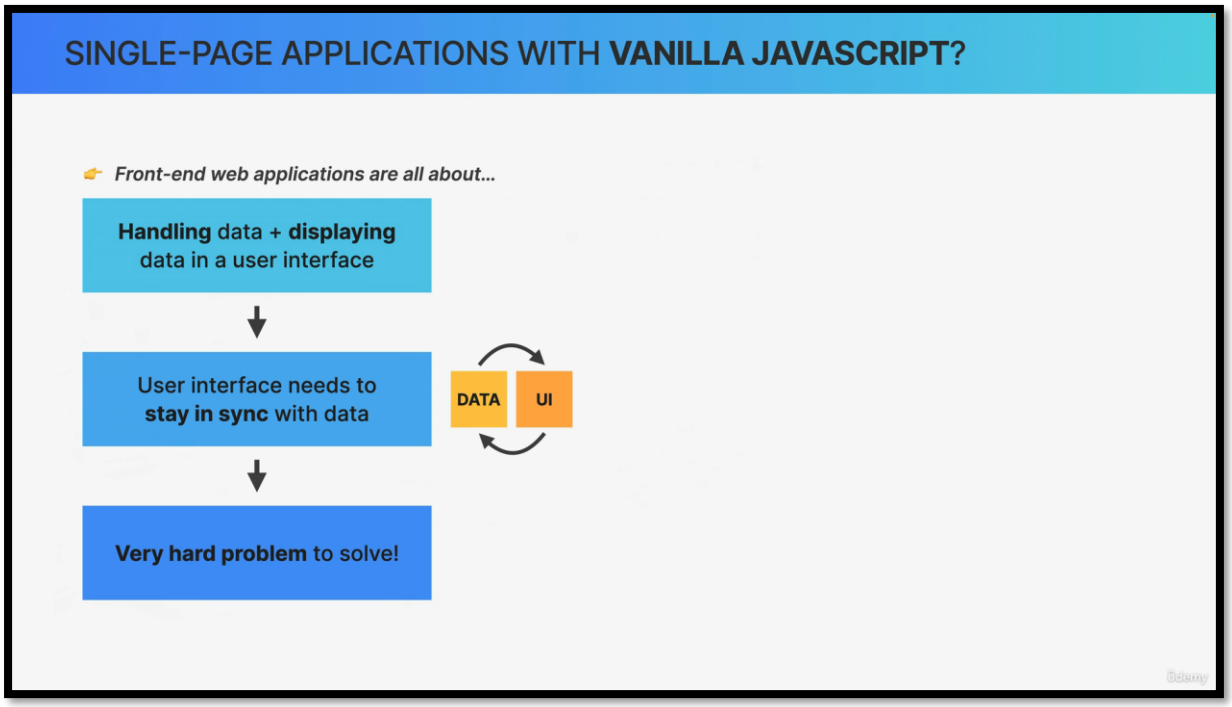
Now a web application still needs data, which usually comes from the backend in the form of an API. So, the application consumes this API data and renders a screen for each view of the application. These single page applications essentially feel as if you were using a native desktop or phone application.

So, you can click on links or submit forms without the page ever reloading. So, you're technically always on the same page and therefore the name single page app.

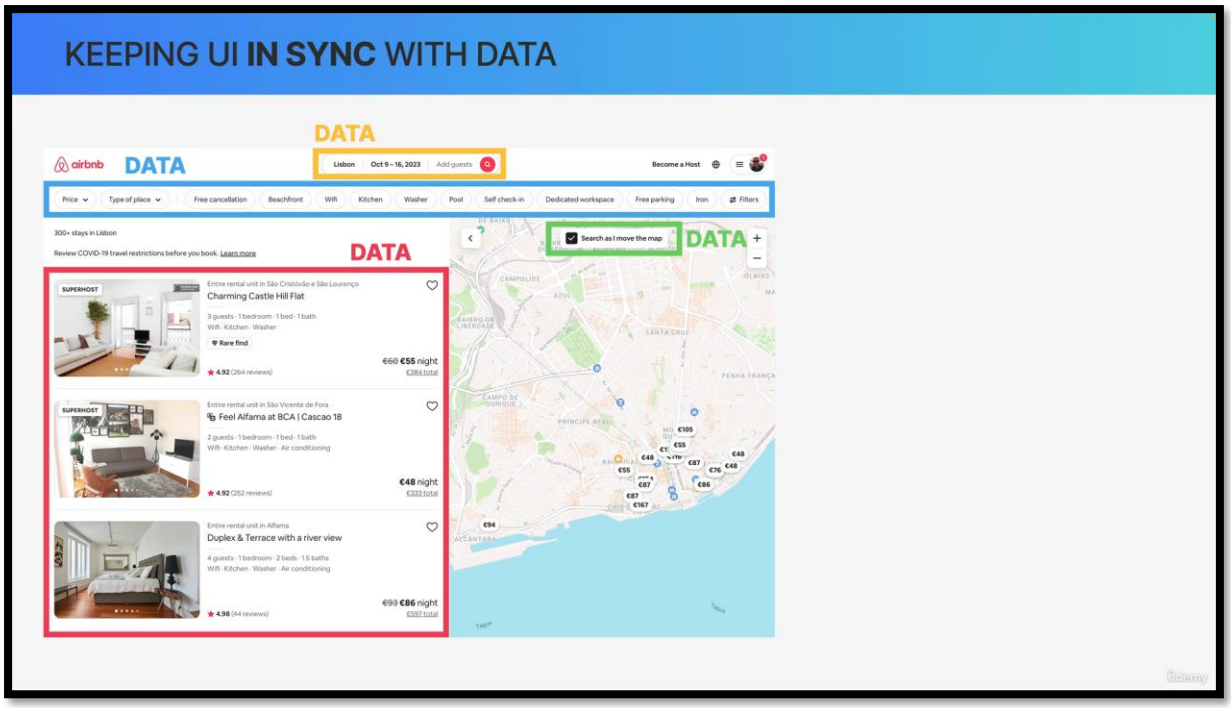
Server-side rendering is actually making a comeback right now. So, it's slowly getting modern, again, driven by frameworks that are built on top of modern client-side rendering frameworks such as Next.js, Remix and many others.

We need to learn how to build single page applications, but do we want to do so with Vanilla JavaScript? Well, actually no, we do not want that because there are actually several problems with using Vanilla JavaScript to build large scale applications.

Let's establish that building any front-end application is really all about handling data and then displaying that data in a nice user interface. So, it receives data, changes the data as the user uses the app and it always displays the current data on the screen.



What this means is that the most important task of a single page app and really of any application and website is to keep the user interface in sync with the data, or in other words, is to make sure that the UI always displays the current state of the data. Now, as it turns out, displaying the correct data and making sure that it stays correct over time is actually a really hard problem to solve.

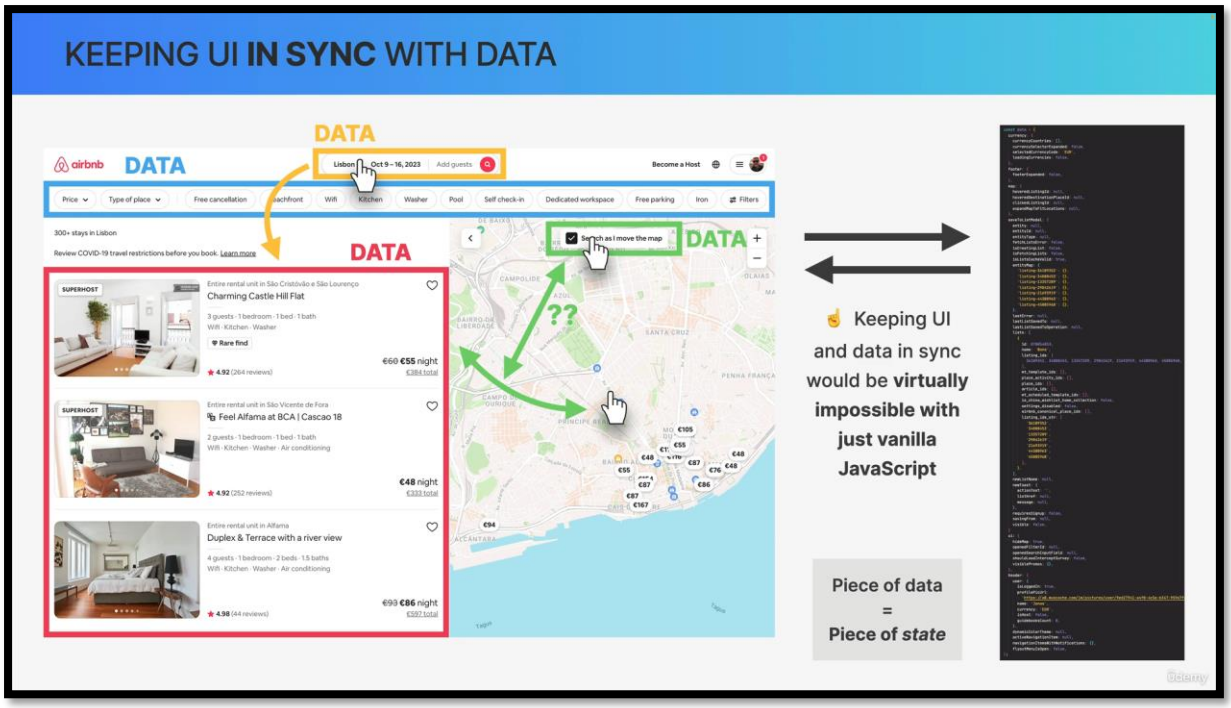


Let's take a look at this Airbnb application. So, in this interface, we can identify a few pieces of data.

We have this list of apartments, a search bar, some data about the filters that are being applied and piece of data (green rectangle box) which indicates whether the search should be updated as the user removes the map.

In the real world, Airbnb application there is just so much data. All of this data needs to be always kept in sync with the user interface and also with the other pieces of data because they're all kind of interconnected. For example, when we change the data about location or dates, then the UI needs to show those new dates and also the list of apartments needs to be updated.

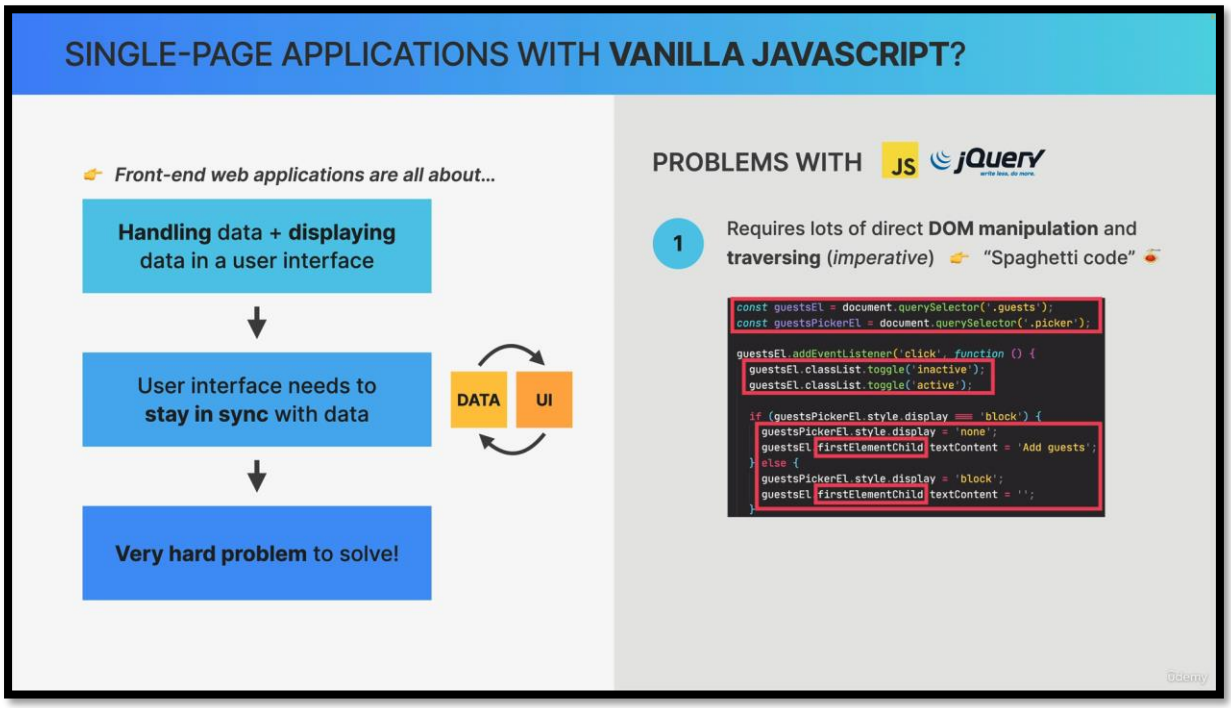
Another example, the map needs to show the location of the apartments. So therefore, when the apartments change the map must also change. The same thing should happen the other way around. So, when the map is moved, the list of apartments should change as well, but only when the user has previously clicked on the green checkbox.



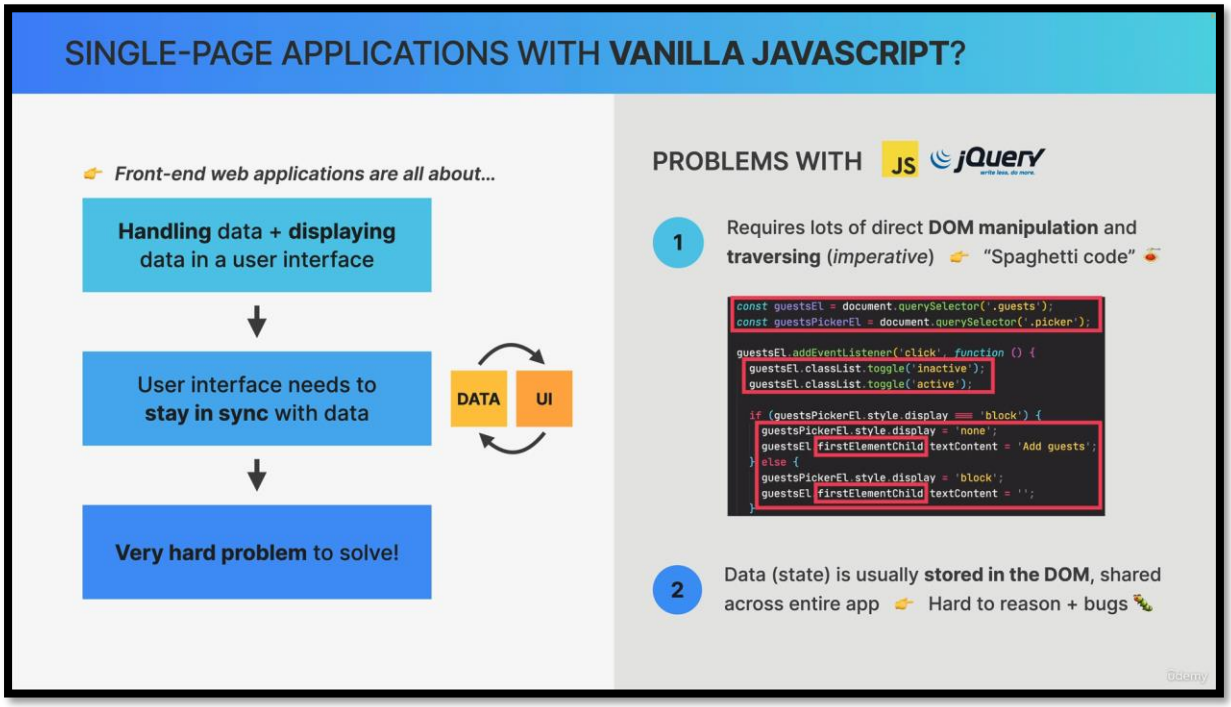
So, these pieces of data here are even more interconnected and it can become a real mess.

In a real-world app, we call each of these pieces of data a piece of state. Without a framework it would be virtually impossible to keep this huge amount of data in sync with this super complex UI.

Why would it be so hard to build something like this with Vanilla JavaScript? Well, it comes down to two big aspects.



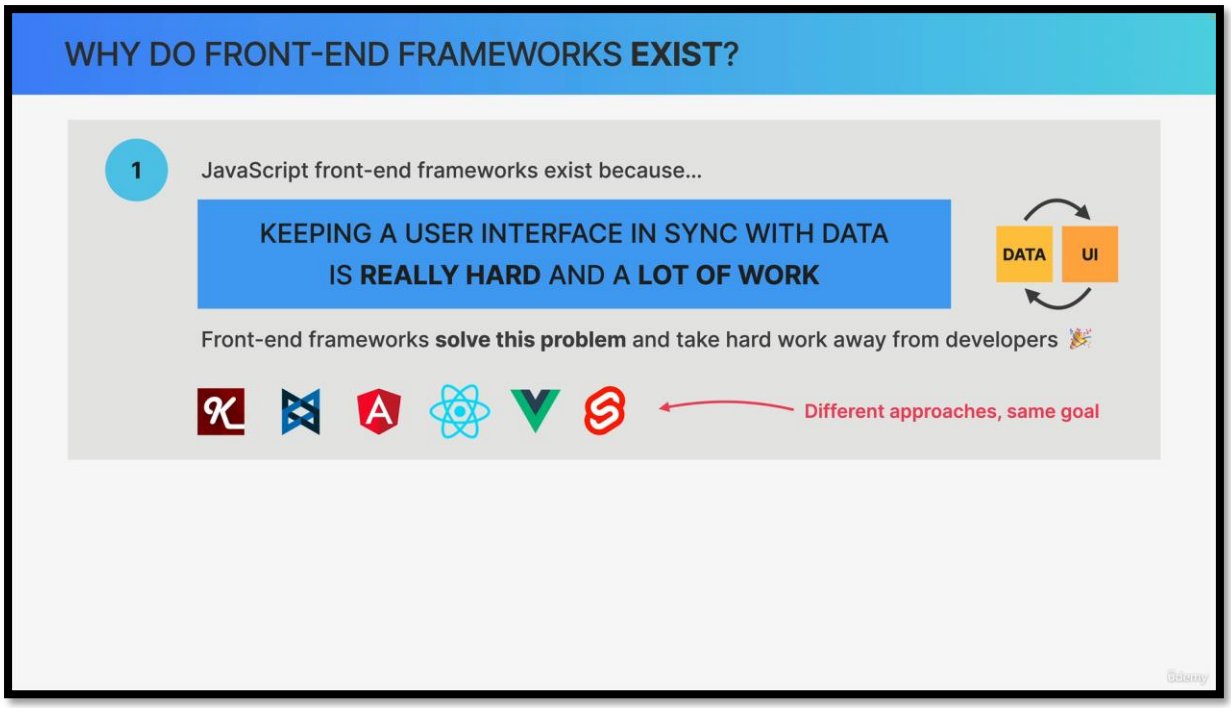
The first is that building a complex front end with vanilla JavaScript alone requires large amounts of direct DOM traversing and manipulation. Like in this code right here, where we have manual element selection, class toggling, DOM traversing and even manipulation of text and CSS styles and this is guaranteed to become an absolute nightmare in a complex app like Airbnb because our code would be extremely complex and really hard to understand, and we will probably just end up with a huge mess of entangled spaghetti code. So, this is the first problem.



The second big problem is that in typical Vanilla JavaScript apps state such as simple text or numbers are oftentimes simply stored right in the DOM. So, right in the HTML elements themselves, rather than in a central place in the application. The result is that we end up with many parts of the app accessing and changing that DOM state directly which makes the spaghetti code even harder to understand and even worse it'll most certainly introduce many bugs into our application. No one wants bugs, right?

Now, of course, you could try to solve these problems on your own, but then you will just end up creating your own framework, which will most likely be way worse than all the well-established frameworks that already exist. So, at this point, you might as well just use a battle tested framework like React.

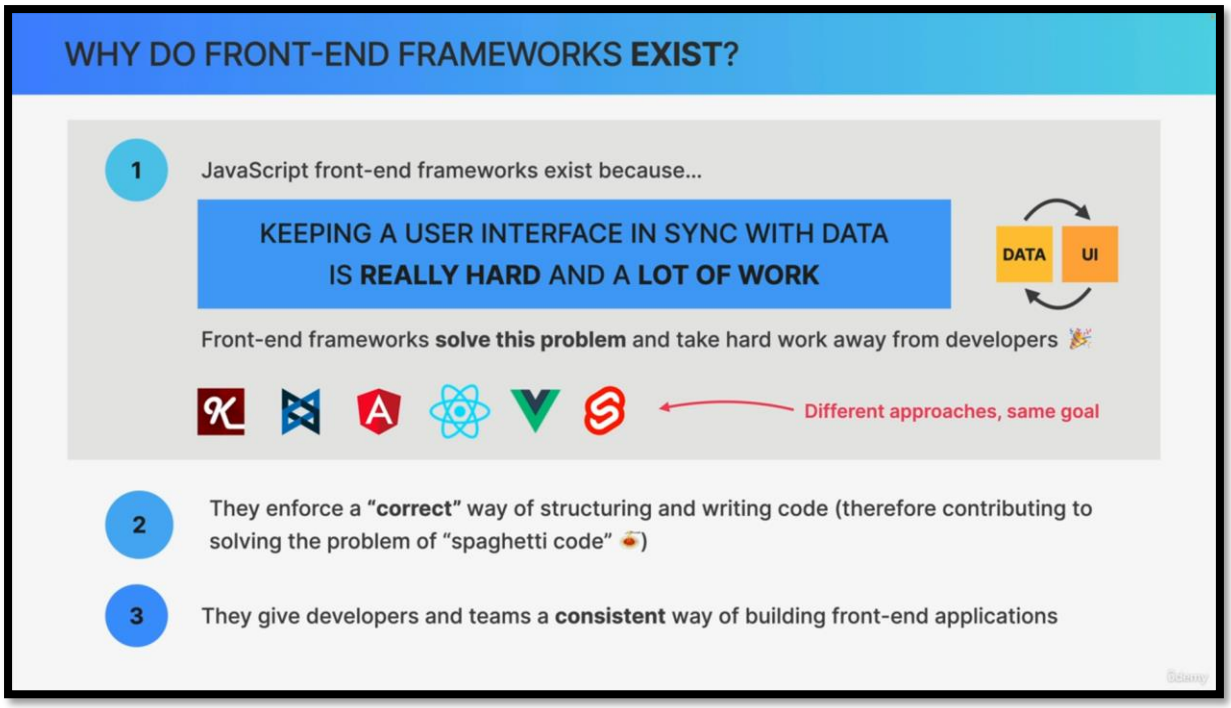
So now that we know why it's so hard to write a single page app with just JavaScript. We can answer the fundamental question that we asked in the beginning. So, why do front end frameworks actually exist? Well, we kind of already answered that question.



So, the big fundamental reason why these frameworks exist is because keeping a user interface in sync with data is really hard and it's a lot of work too.

So, basically frameworks like Angular, React or Vue take this hard work of synchronizing data with the user interface away from us developers. So, they solve this really hard problem so that we developers can focus only on the data and on building our user interfaces themselves.

Now, different frameworks have different approaches to doing this, but they are all similar in the fact that they keep UI and data in sync over time.



Another extremely valuable thing that frameworks give us is the fact that they basically enforce a correct way of structuring and writing code. So essentially, the authors of each of these frameworks came up with a good way of structuring applications. So, that other developers can then follow these conventions as well, to build better applications with hopefully a lot less spaghetti code.

Finally, frameworks give developers, and especially teams a consistent way of building web applications. This way, everyone on the team will build their part of the app in the same style as everyone else which will in the end create a more consistent code base and product.