1-How to Get Help?

Hello and welcome.

We're going to get started with course in just a moment and I'm really excited to teach you everything I know about Vue js, but before we do, I want to tell you how you can get help if you get stuck at any point during this course.

So, of course, in this tutorial, we are going to learn all the significant aspects Vue js. So, as you start to go through the videos and write out our code, if you have any questions or need any help troubleshooting, you can reach out to me on Udemy's Q&A section down below.

Now, I always recommend that you start there, because very frequently, if you are running into an issue, the chances are someone else has as well. Don't think you were alone. And there might already be some posts describing how to solve the issue that you've come up with.

You can also reach out to me via direct message on Udemy as well.

OK, so let's take a quick pause right here and start discussing TypeScript in the next video.

2-What is Vue JS?

In the series we will start from scratch and learn the different concepts in Vue. In this introductory video, I will briefly talk about what and why of vue.js and also the prerequisites to get started with Vue.

All right. Let's begin with what is vue.js. Vue.js is a popular JavaScript framework for building user interfaces. The last three words are really important.

The core Vue library is focused on doing one thing and doing that one thing really well which is building user interfaces. Vue does not focus on other aspects of your application like routing or http requests.

Now if that raises a concern as to how would you handle routing http and other functionality. Let me tell you that Vue has a rich ecosystem of other powerful libraries that you can integrate based on your app requirement.

For example, there is Vuex (Vue X) npm package for complex state management, Vue router for routing, Veautify for UI elements and a lot more libraries for every requirement you might have for your application.

All right now that we know what Vue is and for what purpose it is used. Let's see why you might want to learn vue.

I'm going to start with a comforting fact. Vue currently has 190.000 github stars on its repo which makes it the third most star github repository in the world. Does that mean it is better than react or other JavaScript frameworks? Absolutely, No?

It simply means that thousands of developers around the world enjoy working with Vue, because of this huge community you're going to find solutions to most of the problems that you face when building complex web applications.

This for me personally is a huge plus point. Knowing that I’m going to find resources online for any problem that I might face during development is always comforting, but let's focus on the technical aspects of Vue which makes it a wonderful framework to use.

The Vue docs categorize this into three core reasons. The first one is that Vue is approachable. Getting started with Vue is really simple. All you need is familiarity with HTML, CSS, and JavaScript.

You can add a script tag with a reference to vue.js and start building Vue applications in no time at all. The developer experience is also great with tools like the Vue dev tools which gives insight into your apps and also tools like the Vue cli with which you can quickly scaffold and manage projects.

Vue also has a component-based architecture. This lets you break down your application into small, encapsulated parts which can then be composed to make more complex UIs. Components make it possible to write reusable code which is always great for any project.

Another important point is that Vue is declarative. Now what does that mean? Well basically we just have to tell Vue what we want the ui to look like and Vue will do the hard work of ensuring the DOM is modified to reflect the UI.

This is in contrast with the imperative programming model where we usually list down the steps explicitly and ask the library to do one thing after the other.

The bottom line is that Vue will make it painless for you to create complex user interfaces by abstracting away the difficult parts.

The second reason is that Vue is versatile you have the choice of creating powerful single page applications from scratch using build tools like webpack or you can choose to incorporate Vue into your existing legacy projects and make progressive enhancement by making changes only to certain parts of your application.

The third reason is that Vue is performant. Vue measures just 20 kilobytes minified and gzipped at one time. Also because of virtual DOM only the necessary pieces in the DOM are updated which results in better performance.

Vue is a framework built by taking the good parts from a lot of other libraries and frameworks and then improving where necessary. So, if you work with react for example you might actually see a lot of similarities. Let me just tell that Vue is going to be a great addition to your skill set.

Now then what are the prerequisites to get started with Vue. As I’ve already mentioned HTML, CSS and JavaScript fundamentals are absolutely necessary. For this course I will also be making use of es6 plus features. So, our knowledge of modern JavaScript is also essential.

You don't have to be an expert by any means but there are a few concepts that make it so much easier to write Vue code.

My goal is to make sure we all advance from a complete beginner to being an expert with Vue. All right then with this introduction let's get started with a simple image generator application in the next video.

See you guys in the next one.

3-App Overview

In the last section, we talked about What is Vue JS?

So, Now in this section, let's talk about the app that we're going to build and get a better idea of how we're going to put it together.

All right.

So first off, a quick little mockup.

So, we're going to build something that we're going to call an image generator. In this little generator, our users are going to enter in a little bit of text into this input right here.

We're going to take that text and we're going to generate what is called robohash. Robohash is an easy web service that makes it easy to provide unique, robot/alien/monster/whatever images for any text. Shortly we will build an image generator.

And I can almost guarantee you that even if you've never heard of a robohash before, you probably more or less have seen one in action already.

So as a very quick example, let me show you a couple of generated images over on robohash dot org.

So, these are generated images right here.

They are created by taking some input string.

We take that string and then we turn it into a robot, alien picture like, so. These pictures are traditionally used whenever we need some type of user picture where the user doesn't otherwise provide some profile image or something like that.

Now, might sound like it's really intimidating to make it this, but in truth, you and I are going to use a library that's going to kind of automate the process for us and make it a little bit more straightforward.

You and I are going to be much more concerned with how we do stuff like, taking some user input and how we call some function inside of our Vue code and eventually get some type of image to display inside of our application as well.

Again, you and I don't really have to worry too much about generating this image.

So, with this in mind, I want to give you just a little bit more background on app and how robohash work.

So, the idea behind an generating an image line is that we'll have some type of input string.

The string input can be any string you might possibly imagine, like QWERTY, 1234 or just about anything else. A little function that you and I are going to use and again, this is going to be a function that we're going to use is going to take that input string and then turn it into a picture of sorts.

And the picture is going to look vaguely like the ones you see over here.

I'm just using these pictures right here as kind of a stand in of sorts.

One of the most important aspects of robohash is that if you put in the same input twice, you'll always get back the exact same image.

So, in this diagram, you'll notice that I put in QWERT, and I got out this identical one right here and then at the bottom, if I put in QWERTY again like a second time, I will always get back the exact same icon again.

So same input, same output.

OK, so I think that we've got a kind of reasonable idea of what we're trying to build here.

So, let's continue the next section and we're going to get started working on this application.

4-JS Fiddle Setup

In the last section, we spoke about how we're going to be making an image generator. We’ll now begin working on this application by creating a new JS Fiddle instance.

Now, just to be clear, we're not going to use JS Fiddle throughout this entire course.

In fact, this is probably the last time we're going to use it after this application, will then go on to using some actual code editors on our local machine to work on our apps, which is much more realistic of how you're going to really work on new applications in your own professional and personal projects.

So, again, last time we use JS Fiddle. Now I have another preconfigured JS Fiddle instance, that we're going to use as a starting point.

Here's a link to it on the screen.

So, I'm going to copy this link.

I want to also remind you that this is a case sensitive link.

So, I'm going to open a new window and I'll go to that link and of course, I see JS Fiddle immediately appear on the screen.

Now, this JS Fiddle instance I just opened, has just a tiny bit of configuration already in it and I want to show you exactly where that is.

Just so you really understand the starting point that we are kind of arriving at right here.

I'm going to find the panel and I'll click on this dropdown menu.

So that lands me on my hand JavaScript settings.

If you click language dropdown menu, you will see the languages that you can work with in here. I make sure that I choose Vue.

You'll also notice that Vue has already been inserted into this project. So, we don't need to import it or add it in any other way or anything like that.

OK, so close the menu and now we're pretty much ready to get started on our application but this time around, as we start working on this app, I don't want to just throw code on the screen like we did last time.

Instead, I want to give you a very well detailed and well-thought-out thought-out explanation of exactly what Vue is going to do for us and how we're going to use it.

So, let's come back the next section and we're going to talk about how we're going to use Vue to build our little robohash application.

So, see you in just a minute.

5-Vue Template and Instance

Now that we've got our JS Fiddle instance put together, it's time to start thinking about how we're going to actually use Vue to build this application.

I took the liberty of taking our mockup right here and I added a couple of notes to it. Just so we can better understand some of the different things that we're going to have to do.

So, here, you'll see my two notes. Pretty straightforward notes for right now.

The first one says that, yeah, we're probably gonna need some, like starting text and some form input to appear on the screen any time a user comes to application. By this note right here, I mean, to say that any time a user comes to our application, we want them to instantly see the text.

My Robohash title appear on the screen. Along with the description, text input and a form input and the output thing right here.

So, all those elements are supposed to appear on the screen of our browser.

The other aspect of our application that we want to be aware of is the fact that any time a user types into that text input, we want to run some JavaScript code that's going to create the robot image itself and then somehow put that onto the screen as well.

So, if you really think about these two elements right here, they're kind of describing two different aspects of our application.

On the one hand, we've got the aspect of somehow rendering or getting content to appear on the screen.

And then on the second item here, we're talking about somehow interacting with user input or responding to user input.

In the Vue JS world, we map up both different tasks to two different elements or two different parts of the application.

The first part is the first item we spoke about. Somehow describing the structure or the content of our application, like what appears on the screen when a user first comes to our application.

That task is handled by creating what is called a Vue template.

So, we create new templates to show information to users.

Vue templates are created in HTML inside of our application. When I say created in HTML, I mean to say that in JS Fiddle, we're going to create our Vue template inside of this HTML panel right here.

The first template that you and I create is going to be almost indistinguishable from normal HTML that you might create in any typical application but over time we're going to add in some advanced syntax to that HTML, which is going to turn it into our actual Vue template. Thing I want to remind you about here, and I know I've said it two times, but I'm going to repeat these many times throughout the course.

The Vue template is in charge of showing content to our users.

Now, the second aspect of our application, like handling user input, is going to be handled by creating what is called a Vue instance.

A Vue instance is created by writing out some amount of JavaScript code and the JavaScript side of our application.

And so, relating that back to our code and window, our Vue instance is going to be created down this JS panel here.

This Vue instance is what is responsible for dealing with user input through some fashion.

So, any time user types out some text into that text input, we're going to use our Vue instance to take that text, generate an image out of it, and then take that image and pass it back over to the template.

And the template will then present it to the user on the screen.

Now, one thing I want to mention here in particular is that this is just one possible way of many of structuring of Vue application.

So, you might be immediately curious, well, how else would we structure a Vue application?

To give you a very quick example, I just told you that we're going to create our Vue template on a HTML file or the HTML section of JS Fiddle.

But in some other uses of Vue, you'll sometimes see this template created over in the JavaScript side instead. In this first application that you and I are going to work on, we're going to first start by creating the template on the HTML side.

But once we put our application together, I'll very quickly show you how we might define our template over on the JavaScript side of our application as well.

Don't let this little item confuse you.

However, it's a real straightforward thing to kind of redesign your application and move where these different elements are defined.

OK, so just one more time just to make sure it's really crystal clear.

We use our template to somehow define the presentation layer of our application and we use the Vue instance to somehow respond to user feedback.

So, with that in mind, let's continue in the next section where we are going to create our first Vue template.

So quick break and I'll see you in just a minute.

6-Creating A Template

In the last section, we spoke about the differences between a Vue template, an a Vue instance. We're now going to get started on creating our Vue template.

And remember, this template is going to be the sort of presentation layer for our application. For the first iteration of our template.

We're going to write out essentially just plain HTML and you're going to look at this and you're going to say, hey, this is HTML.

What do you mean template?

What's template about this?

Well, don't worry.

Over time, we're going to come back to this template thing, and we'll add in some additional elements to it, to really turn it from being just kind of plain each HTML into an actual template.

So, without further ado, let's flip back over to JS Fiddle and we're going to write out our first template. In the HTML, I'm going to begin by writing out a simple div.

Then inside of the div, I'm going to put down the HTML that is going to the same structure of the mockup that we were just looking at.

So, inside that div, I'm going to put down an H2 tag to represent the header and then put down a div to hold this text and then a text input to represent the form input right here.

So that's all going to go directly into my Vue template, which is going to be located inside of this div.

So inside of here I'll place in h2 that says Robohash Unique Image Generator.

I'll then place a div that says input and then I'll put down my actual input tag as well.

Then underneath that div I'm going to put one other div in that's going to have the text output just like that.

All right, so this is our first Vue template right here and again, I know you're thinking, still this is a plain HTML.

What do you say in template?

Don't worry, we're going to come back to this little snippet right here and we're going to add in some additional syntax that's going to make it very clear that we're not quite working with HTML here.

Instead, we're really working with the templates.

So, with that in mind, let's take a quick pause.

We'll come back in the next section and we're going to create our Vue instance on the JavaScript side of our application.

7-Linking Templates and Instances

In the last section, we created our Vue template on the HTML side of our application.

We're now going to flip on over to the JavaScript side of our application where we're going to define our Vue instance.

And remember, this Vue instance is going to be responsible for watching user input and somehow reacting to that in some fashion.

So back over inside of JS Fiddle, I'm going to open the JS panel down here. Inside the jazz panel, I'll write out new Vue. Notice I've got one set of parentheses and then I've got one set of curly braces inside of this object right here or inside of the curly braces, we're going to add in a bunch of different properties over time.

All these different properties are going to serve to customize how this Vue instance behaves.

All the different properties are going to be stuffed inside of here is really where a lot of the complexity of Vue itself comes into play and it's understanding these different properties and using them correctly, that's going to turn you into a professional Vue engineer.

So, with that in mind, we're going to introduce the first property that we're going to make use of inside of you instances right now.

So, the first property that we're going to define in here is going to be a property simply called E L.

So, E L is short for the word element.

That element property right here serves to tie one Vue instance to one Vue template that has been defined inside of our application.

I'm going to pull up a diagram to make that a little bit clearer.

All right.

Here we go.

So, here's the HTML side of our application.

And then on the left-hand side, we have the JavaScript side.

We just defined our Vue instance in.

On that Vue instance, we defined a L property.

The purpose of this EL property is to tell Vue where our template sits inside of our HTML structure.

So, we defined our Vue template inside the HTML structure.

We have to tell you to go and find that template and we do so by using that Vue property to do so.

So you really quickly how it's going to work.

We're going to write out the code for it and then I'll tell you very clearly how the two really get tied together.

So first, I'm going to go back up to the HTML section here and on the top-level div.

So, my very root div right here.

I'm going to add in an ID of root like some then down on the Vue instance at the E-L property, I'm going to provide a string that has hash root.

OK, so now you'll notice that we've got to div with ID of root right here.

Let's go back to the diagram and I'll tell you exactly what this little hash root thing is doing.

So, our Vue template now exists inside of that div with Id of root.

By providing hash root we are providing a DOM query selector to our Vue instance that tells it exactly where its template exists inside the DOM.

So, by providing the string hash root, that means go look for some element inside the DOM or the HTML structure that has an Id of root.

Specifically, that hash or pound means ID. By providing that property, we've now instructed our Vue instance where its respective template sits inside of our DOM. Now, that tells Vue to somehow take that template and it tells vue that it is responsible for dealing with all the content of that template.

This is what really links together the JavaScript and HTML sides of our application.

OK, so now we've got a better idea of how to link together our Vue instance in our Vue template.

Let's continue in the next section and we're going to continue talking about how to build this application.

8-Step by Step Implementation

In the last section, we defined our Vue instance and then provided an E-L property. This told our Vue instance, what area the DOM was going to be responsible for.

So now this Vue instance right here is tied to all the content placed inside of this div.

You'll notice, however, that there's really apparently no change to our application at this point in time.

There's no difference.

There's no different content that's being rendered on the screen.

It's only as we start to add some more code to our Vue instance that we're going to start to see any change inside of our HTML output over here.

So, with that in mind, let's get a better idea of exactly how we're going to implement our application.

Remember, we've said several times that we want to make sure that any time user types inside of here, we generate a brand-new image and then show it next to this output section right here.

So, let's walk through the flow that we're going to use to implement that behavior.

All right.

So first off, we're going to walk through the exact series of steps that are going to occur inside of our application.

The first thing we need to do is detect any time a user has entered some new text into this input right here.

And there's a very small clarification I want to make or very small detail, I want to point out. I want to make sure that any time a user types one character, we generate a brand-new image and then show it on the screen.

We're not going to wait for a user to type out a bunch of text and then press enter.

So, it's with every single key press that we're going to generate a new image.

So that means that we really have to detect every time a user enters a new character into that input.

Once that occurs, we then need to somehow get the text that the user enters.

Remember, we're using that text to generate the image.

So, it definitely makes sense that we need to know exactly what the user just entered.

Once we get that text, we're then going to use it to generate the new image itself.

Now, this step right here, a little bit scary.

We don't really know how to do that just yet outside of using that API we took a quick glance at a couple of videos ago.

So, we're going to somehow use that web service to generate the new image.

But I think it makes sense that we're going to have to probably run some custom JavaScript code to do so.

So, for this step right here, we're probably going to have to have the ability to, like, run some function or something like that, any time a user enters some new text.

Then finally, after we generate that image, we're going to take it and we're going to show it right next to that output section right there.

Now, everything I just listed out, like these four steps right here might seem like they're very obvious.

You know, it might seem like well, Stephen, of course, we want to, you know, detect any time a user type something.

And of course, we want to generate the new image and then somehow show it on the screen.

But the reason I'm showing you this diagram right here is that for every one of these steps right here, Vue has a different feature built in to help you implement each step.

So, every one of these different steps that we're going to go through, we're going to learn about a different piece of The Vue API or the layer that we use to interact with that Vue instance that we just created.

So, with that in mind, let's take a quick pause.

We're going to come back in the next section and we're going to learn how to implement every single one of these steps bit-by-bit.

So quick break and we'll see you in just a second.

9-Defining Instance Methods

In the last section, we spoke about the four different steps that we're going to implement to get our application working. In this section, we're going to start off with step number one.

So, we need to make sure that we can somehow detect any time the user enters some new text and whenever they do, we're probably going to want to run some custom JavaScript code.

So, we're going to focus on this first step right here inside the section. To implement the step, here's what we're going to do.

You and I are going to add what is called a method to our Vue instance.

And we're going to make sure that that method gets called any time our user enters some new text.

So first, let's get a definition.

What is a method?

A method is a function that is going to be tied to our Vue instance.

That function can then be called at any point in time to somehow implement or somehow update our user interface.

So, we primarily make use of methods to somehow respond to user events, which is exactly what we're trying to do right now.

Let's first begin by implementing the method and then we'll figure out how to call it any time a user enters some text.

So first, the method. To add a new method to our component instance or give me our Vue instance right here.

I'm going to add a comma after the E-L property and then I'll write out methods like, so. Methods is going to have an object tied to it and inside this object, we're going to define all the different methods that are tied to our Vue instance.

Right now, there's just one method we care about. The method that's going to be called any time a user enters some text into this input.

But over time, if we had a more complicated template with like more text inputs or more buttons or just more ways of interacting with it, we might end up having many different methods defined inside this object.

So, let's begin by implementing this method that's going to be called any time a user enters some text.

I'm going to call this new method handle input change like so. handle input change is going to be a function, and it's functionally called any time a user enters some text.

Now, one thing I want to make sure is clear here that the word handle input change was just somewhat randomly picked by me.

We did not have to call this function handle input change.

We could have called it on User Typing, or it could have been ‘when user type’. The anything we want.

But in general, I find that using the codification of handle and in the event that occurred to be a really straightforward and simple to understand. So, handle input change makes it really clear to other engineers in the future that this function right here might be called any time someone types in some new text into our text input.

OK, so that's kind of step one of step one.

We just added a new method, which is a function that will be called any time a user enters some text.

Now, just to make sure that we know when we're kind of having some success here. Let's add a console log into this function and it can say anything inside the console log.

I just want to somehow see the console log appear any time user types.

So, I'm going to add in console log, ‘User is typing now!’ and then I'm going to open my console so I can see this console log appear to open the Chrome console.

You can right click anywhere on the screen and then click on inspect and then find the console section up here at the top.

Now, you might already see a couple of errors or something here inside the console.

If you do, that's totally fine. You can click on clear console to clear them out for now.

If you do see errors, chances are they are appearing from when we were first working on our application.

For right now, you should probably just ignore them unless they start to reappear, in which case then you're going to want to figure out what's going on.

OK, so now we'll try typing inside of here and just see what happens.

So, if I type inside this input, you'll notice that we do not see any console log appearing on the screen, which is totally fine.

You see, even though we've defined a method called handle input change, we have not instructed our Vue instance or our Vue template that we want this method right here to be called any time user types inside this input. We have to very directly wire up typing in this input right here or just wire up this input to calling this method.

They do not automatically get tied together.

So, let's take a quick pause and figure out how we will implement that step inside the next section.

So quick break and we'll see you in just a minute.

10-Defining Vue Directives

In the last section, we defined our first method inside of this methods object on our Vue instance, we defined a method called input, and it's a function that hopefully will eventually be ran any time a user enters some text into the input right here.

However, as we just saw, that's not the case right now.

We have to make sure that we somehow tell this input tag right here that any time a user enters text into it, we want to run this handle input change function.

To do so, we're going to start to change the HTML structure of our template, and this is where our template is no longer going to look like plain HTML. It's going to look like something a little bit more complicated.

So, let's add in the code that's going to somehow tie this input right here to our handle input change method and we'll talk about exactly what it's doing.

So, on the input element, I'm going to add something that says, ‘v dash on colon input’ and then I'll say equals and then inside of double quotes, make sure these are double quotes right here.

I'll say handle input change like so.

Now, I'm not going to let that run, I definitely don't see any errors over my console over here.

I do see some warnings, but those are totally OK to ignore for right now.

So, it looks like, you know, maybe something good happened. I don't know.

Let's try entering some text into the input and seeing what happens.

So, if I type inside of here, you'll see that I'm now getting a bunch of console logs. Let me zoom this in.

I get a bunch of console logs that say, ‘User is typing’.

So clearly, any time someone is typing inside this input right here, it is causing my handle input change function to run.

So, what happened?

Well, with this right here, we defined something called a directive.

A directive is a piece of template syntax inside a Vue that somehow enhances the behavior of otherwise normal HTML code.

So, behind the scenes, when our Vue instance first boots up, it looks at the EL property right here and as we said previously, it tries to find some element inside the DOM with an ID of root.

The Vue instant's finds are div with of evaporate here, and it scans over all the HTML that we have input inside of here.

When Vue scans over all this HTML, it looks for any directives like the one we just added in right here that have been added to our HTML.

If it finds any of these directives, it then does a couple of extra little steps to parse this directive and figure out how to correctly process it.

In this case, we have to find a directive that sets up an event handler.

The left-hand side of this that says V dash on specifically means we are trying to define an event handler.

We then place a colon and then the name of the event that we want to watch for.

In this case, we are watching for an event of type input.

So, any time a user enters input into this input tag, Vue is going to attempt to do something for us.

The thing that it actually does is defined by us on the right-hand side of the equals sign.

So, we said equals and then in big double quotes handle input change.

So, Vue says, OK, any time someone enter some new input into this text box right here, I'm going to try to run a method tied to my Vue instance called Handle Input Change.

So, it's specifically because we call this thing Handle Input Change that it matches up with a method that we defined down here, called Handle Input Change and again, we didn't have to call this Handle Input Change.

We could have called it handle input change and if we called it that, then we would have to make sure that we updated the directive as well.

So, we would have to say handle input change, when I update that, I can then still type inside of here and see the council logs appear.

We're going to be talking about a collection of different directives over time and I'm going to be completely honest with you in my opinion, I think that probably the most complicated part of you in general is understanding these directives.

So, I'm going to make sure that I put a lot of extra emphasis on making sure that you really understand how these directives work.

The event handler directive is one of the more straightforward ones, but over time, we'll start to see some more complicated directives.

OK, so in this section, we took the method that we had previously defined.

We added a directive to our input tag and that directive linked our input to the method we had previously made.

Before we move on, one thing I want to do very quickly, I just want to revert to the change that I just made there.

So, I'm going to revert this to saying Handle Input Change and I'm going to change my method back to being Handle Input Change as well.

OK, so it looks like some good progress.

Let's take a pause right here and we'll continue in the next section.

11-Retrieving Event Information

In the last section, we used a Vue directive to tie an event that occurred to this text input to the Handle Input Change method that we define inside of our Vue instance.

Like I said previously, viewed directives are one of the more complicated parts of Vue and we're going to go through too many times in this course review exactly how they work.

Now that we can run a function any time user types in this input, we need to now move on to our next step, which is to somehow get the text that the user entered.

To do so, we're going to use that Handle Input Change function.

You see, the Handle Input Change function gets called with a single argument that we usually refer to as the event object.

This is an object that contains a bunch of different information about the event that just occurred and, in this case, the event that occurred is an input event.

The event object has the actual text that the user entered.

And so, we can make use of that argument to figure out exactly what text the user typed.

So, let's go back over to our JS Fiddle instance and we'll figure out exactly how to do that.

I'm going to find my methods, object inside of my Vue instance.

Inside there, I'll find the Handle Input Change function and to that function signature, I'm going to reference the single argument that is provided to it.

Like I just said, we usually refer to this as the event object.

This event object has a property called target value and that will be a reference to the text that the user entered inside of here.

So, for right now, let's just console log that out and make sure that the text that was entered appears inside this function.

So, I'm going to do console log event dot target dot value like so.

The first event right here is a reference to the event that just occurred, Target is a reference to the HTML element that just had the event applied to it.

In this case, target will be this text input right here. Then value is a reference to the current value of that text input.

So, for us, that's the actual text that we care about.

So now we're logging that out. Let's try typing some text into this input and then just verifying that it somehow appears inside of our console over here.

So, if I enter some text, you'll very quickly see it appearing over here on the right-hand side.

OK, that's not too bad.

So, I think you can agree with me that now any time a user enters some text, we can retrieve the text that the user has entered.

So now that we've got this step two put together, let's continue in the next section where we're going to start moving through the rest of these steps.

12-Imperative vs Declarative Programming

In the last section, we figured out how to use that event, object to get access to the text that a user entered into our text input. You might be thinking that we're now going to move on to step number three, but instead we're going to do a quick little detour, OK?

We're going to do a little side topic quickly that is not super connected with our application as it stands right now.

So, in this detour, we're going to get a better idea of how to structure and build vue applications and give me a lot of fun, because we're not going to have to obsess over a little APIs or anything like that and we'll get a better idea of some of the big picture stuff around how to design the applications.

So, let's get started.

All right.

So, here's what we're going to talk about.

We're going to discuss the differences between imperative and declarative programming styles. Especially as applied to vue applications. In the world of Web development from maybe 2005 all the way to twenty thirteen or so, we practiced a programming style referred to as imperative programming.

With imperative programming, we write up code that lists exactly what our application should do step by step.

So, we say just like the diagram I had over here, whenever someone enter some text, take the text, generate new image, and then put that image on the screen.

This right here is an example of an imperative design flow where we list out exactly what to do step by step. In the world of declarative programming, we instead list out some rules that our application should follow.

We then provide what we refer to as some initial state to our application, and we let those rules kind of define how our application behaves.

Now, these very quick descriptions, I'm giving you probably don't make a lot of sense.

So, let's walk through the differences between imperative and declarative programming by walking through a little analogy.

OK, so I want you to imagine for just a second that maybe you and I are not building software anymore.

Maybe instead you and I bake cakes for a living.

So, we go into work every single day and we have to bake a cake.

Now, unfortunately, you and I maybe are very poor cooks, and we don't really know how to bake a cake.

So, our boss sits down and gives us a list of directions, essentially a recipe.

So, this right here is a recipe that might tell us exactly how to make a cake and I've got one recipe that is written in an imperative style.

And then we've got one recipe that is written in a declarative style.

So, let's walk through both of these recipes and get a better idea of the differences between imperative and declarative approaches.

Imperative approaches are characterized by step-by-step directions.

They tell you, step one, you start with step one.

You then go directly to step two.

You do step two, then you go to step three, step for step five and so on.

And then after you go through this entire list of steps, you eventually end up with some end product.

So, in the context of baking a cake, maybe our imperative recipe tells us to take these raw ingredients, mix these ingredients together, pour the ingredients into a pan, and then put that pan into an oven.

And then after you go through these four steps right here today, you end up with a cake or your end product.

In a declarative world, it's a little bit different, as you might imagine.

So, with a declarative recipe, we would separate out our recipe into many, maybe two separate parts.

We would start out with one part that lists what we might refer to as the initial state of our recipe.

The initial state would be like the variables, and in the case of a recipe, our variables would be maybe our ingredient quantities in the ingredient types.

We then take this initial state or this initial listing of ingredients, and then we apply a set of rules to them.

So, you can kind of imagine that we take these ingredients right here or this initial recipe.

We pass it through the rules one time, and then maybe after that first time, we then stick it into the rules a second time.

And then maybe after we go through the second time, we go through then a third time, and we keep repeating this process over and over until we eventually come out the other side with a baked cake.

So, it starts to get really important to understand how we might structurally structure these rules in a declarative approach.

So, for the rules in a declarative recipe, we might say, OK, let's examine our ingredients or our state.

If our ingredients have been mixed, then put them in a pan.

Well, clearly right now, with our initial state that doesn't quite match. We have not yet mixed our ingredients.

So, let's go down to rule number two here.

So rule number two says if the ingredients are mixed, mixed them together in a bowl.

OK, well, that's good.

We have unmixed ingredients, so mix them all together in a bowl.

So maybe now instead of having one egg, cup flour and sugar, we would instead have one bowl, mixed ingredients. We then take that state and apply it to our rules again. We've now got our mixed ingredients. So, we're going to look at rule number one.

We say if ingredients have been mixed, put them in a pan.

OK, well, that applies to us.

So, let's do that.

So, we now have one batch ingredients in a pan.

We now take that, and we go look at our rules again.

OK, that doesn't apply.

Our ingredients are now mixed in in the pans.

That doesn't apply.

Well, rule number three here says if we're in a pan, put it in oven for 30 minutes at three fifty.

OK, well, that definitely applies to us.

So, we take this, and we put it in the oven at for 30 minutes at 350 degrees, and then we come out with that with a baked cake.

So, in the world of declarative programming or declarative cooking, I suppose, we have some state that we start off with and then we take that state and apply the set of rules to it.

Now, I know this might sound like it's a real complicated approach for what we're trying to build right now.

So, I took the liberty of writing out our applications kind of rules of sorts in an imperative and declarative approach.

So, let's consider how we might build our application in an imperative approach and a declarative approach.

So, in an imperative approach to our current program, which is kind of what we're following right now, we might say that whenever a user enters some text, we want to do step number one right here, which is to get the new value out of that input.

Then we'll move directly on the step number two, which is to turn that input value into an image and then we'll go directly to step number three, which is to take that image and put it on the screen.

Now, this might seem like a very direct and easy to understand flow and you might be thinking, hey, imperative programming like this meshes with me.

I understand this.

This is what I'm used to.

Well, maybe for very simple flows like this, that makes sense.

But most Web applications that we spend any amount of time building are flows of what occurs inside of our application is much more complicated than this.

So even though I'm giving you a very simple example here, I hope that you can kind of extrapolate this example and imagine a more complicated flow.

We're going step by step in code.

Might be a little bit more challenging.

OK, so this is the imperative approach to our current application.

So, let's now look at a declarative approach.

OK, so here's our declarative approach.

So, if we took a declarative idea to our current application, then maybe we would say that we've got some like initial state or some like initial ingredients of sorts of a single variable called something like text input and maybe text input starts off as an empty string. So, this is supposed to be empty string.

Well then take this kind of initial state of sorts and apply it to this set of rules that we've defined inside of our Vue instance.

So, we'll say, OK, maybe we'll say, well, if a user enters text update, text input, well, that doesn't really apply to us.

We'll say if template is rendered, calculate the image.

Well, that doesn't really apply to us, and we'll say we'll have text input is updated, render the template.

OK, well, no, these quite seem to apply to us, so we'll just kind of wait around and wait for something to happen.

So maybe then at some point in time a user updates our text input and when that happens, we might apply rule number one right here.

So, if a user enters some text, we want to update the value of text input.

So, I'll now update that to whatever our new value is, which maybe is like, you know, VUE JS and then we also look at the other rules that are listed in here.

So, we just updated our text in that caused text input to update as well. So now we'll look at rule number two that doesn't apply. We'll look at rule number three.

It says, if text input is updated that just occurred for us, then we want to rerender the template. OK, so we'll render the template.

And if we look back at rule number two right here, if template is updated, calculate the image and then.

OK, well, let's calculate the image and then put that on the screen.

So, I know that these rules right here don't quite translate from the idea of, like baking to what we're doing right here.

But I think you can kind of start to get the idea where inside of our Vue instance will declare some type of data or some type of variable to sit around.

And then inside of that Vue instance will declare the set of different rules.

That tells our Vue instance how it should behave whenever something occurs inside of our application.

So, again, in general, you and I inside of Vue want to strive for this more declarative programming style as opposed to a more declarative approach.

Again, I know that imperative programming might seem like a little bit clearer and more obvious, and it might be more challenging to understand.

Why we would take this approach right here, but in general, a declarative approach scales very nicely for larger applications.

OK, so now we've got this idea of declarative programming in mind.

Let's continue in the next section where we'll talk a little bit more about how we can kind of implement these rules right here into our Vue instance and get our application working.

So quick break and we'll see you in just a minute.

13-Declarative Apps with the Vue API

In the last section, we spoke about some of the differences between imperative and declarative programming.

We're now going to take this idea of declarative programming and apply it to our actual application with some real terminology that is used in Vue instances.

So, the first thing we're going to do is take a look at each of these steps right here.

And I want you to consider like the actual purpose of each of these rules or maybe a better way to phrase it would be to imagine when each of these rules would be applied.

Let's look at what I mean. Here we go.

So, at the very top, we start out with that initial state of sorts or that like initial list of ingredients.

We might also refer to that as our data inside of our application.

The first rule right here where we say watch for a user to enter some text and whenever that happens, update, text input.

We'll try to characterize the purpose of this step right here.

I would say that the purpose of this step is to update our data or to update our state.

So, this is really an update action or kind of like an action that changes things inside of our application.

The second step right here, which says that whenever the template gets re rendered, we need to somehow calculate that image to show on the screen is a step that somehow uses our data or uses our state to show stuff on the screen.

Another way to put it would be to say that this step right here kind of consumes data to present it to users inside of our application.

The last step down here, which is to say that if the text input is ever updated, rerender the template, this one is a step that just kind of happens automatically with a Vue behind the scenes.

So, any time you and I change our state or change our data inside of our application, our Vue instance is going to automatically update everything on the screen.

So, this step right here isn't super important to us. Right now, I'm a little bit more curious about how you and I can represent our initial state and these two sets of rules right here inside of our application.

So, again, just one more time, this step right here where we say, OK, let's take that new text and update, text input.

That's a step that really updates our data and then turning the text into an image is a step that kind of consumes our data, so to speak.

So, with that in mind, let's now apply some more precise terminology to each of these steps.

So, these are terms right here on the right-hand side of the different properties that you and I are going to eventually define on our Vue instances to implement each of these different steps.

So first off, at the very top, we've been referring to this kind of initial list of ingredients or our initial state of sorts.

In a Vue instance, we refer to this as a property defined as data.

So, you and I are going to define a data property inside of our Vue instance, and that's going to tell our instance what the kind of like starting ingredients of sorts are inside of our application.

Next, you and I have kind of already taken care of this step right here.

But to really break it out, you and I are going to define a couple of different methods eventually. One side of this application, we're just going to define the one, and that's the handle input change method that we've got right here, but the purpose of that method's property is to define a bunch of functions that are supposed to somehow update our data.

So, all the functions you're going to see on methods is going to describe how we might update the data inside of our application and thinking about what it's being used for inside of our application right now, yeah, that definitely matches up.

We're saying that any time a user instigates a change, you and I want to update our data and so to do that, we're going to use that method's property inside of our Vue instance.

Now, the last one right here or second to last is a little bit more complicated.

So, any time you and I want to kind of consume that data and get it into our actual template or show it on the screen in some fashion, we're going to use a computed property.

OK, so again, these are three pieces of terminology or three properties that you and I are going to define inside of our Vue instance eventually.

So, we've got data to kind of initialize our data inside of our application.

We've got methods to change it and then we've got computed to somehow consume it and get it to be visible on the screen.

OK, so I think, you know, that terminology may be starting to make a little bit of sense. Don't worry, we'll see a lot of examples around it, but just to really drill home this terminology right here, we're going to look at kind of two more slightly repetitive diagrams.

So here we go.

So just to be clear, data methods and computed, these are all parts of the API. Data defines the initial state. Methods defines the ways that we can change that state. Finally, computed defines how we're going to take that data and turn them into viewable values that can be displayed on the screen.

Now, obviously, the text that a user enters our application is itself like a viewable value but for you and me, we're trying to create an image in here.

So, a viewable value would be taking that text and turning it into image. Now, just in case all this terminology is still just way too complicated, I took even this diagram, and I made a more simple version of it, one that hopefully will even stick in your brain more easily.

OK, so here's here's as simple as I can put it. It's a really just put it down as simple as I possibly can.

We use data to describe how everything starts, like how all our data starts.

We use that method thing to change our data and we use the computed thing to turn that data into a viewable value.

So, this is as concise as I can possibly make it.

OK, so we've now got a good idea of some of these different parts of the API.

I'm sure you're still kind of unsure of where these terminology things right here apply.

So, let's continue the next section and we'll start to apply these data methods and computer things and trust me, you'll very quickly get a very good understanding of what's going on.

So quick break and let's take care of this in the next section.

14-Data, Computed, and Methods

In the last section, we took the idea of declarative programming and applied it to our application and more specifically took a look at The Vue API. In some of the different properties that were going to define on our Vue instance.

So, let's now go back over to JS Fiddle.

We're going to start defining these different properties inside of the JavaScript side of our application.

OK, so here's JS Fiddle.

I'm going to give myself a lot of space inside this JavaScript panel right here. Just so I can very easily see everything.

Now, you recall that we already defined that method's property, that methods object.

So, I'm going to add a comment inside of here just to remind myself, use these functions to change data like so.

Then I’m going to immediately define that data property and the computed property as well, and I'll add in some comments on those to remind myself of how they are used as well.

So, I'm going to say data is going to be an object.

I'm going to add a comment in to say, initialize our list of ingredients and we don't call this our list of ingredients.

I'm just putting that on here on here to remind you that you kind of imagine this as being like a recipe of sorts and then finally, we'll also add in computed and I'll add in a comment that says, turn data into viewable values like so.

OK, now one thing that might be a little bit misleading here with methods, we have clearly defined a function right here. Inside of this computer object, we are also going to define functions.

So, these are going to be functions that are executed to turn our data into viewable values but with a data property right here, we are not defining functions. Instead, we are defining simple properties and property names and property values.

So, for example, if we want to call the text, whatever kind of like piece of state we refer to right here as our text input, we would initialize that ingredient of sorts by writing out Input Content is empty string like so.

So, with data we define simple key value pairs that have, say, strings or numbers or arrays or objects.

But with computed and methods, we define functions on both these objects and those functions are executed to either update our data or turn that data into a viewable value.

OK, so let's take a pause right here.

Now that we've added in some comments to further guide us, we'll continue in the next section and start implementing data and computed as well.

15-Updated Data Values

In the last section, we added in our data property, in our computed property. We also added in some comments to describe their purpose inside of this Vue instance. In the section, we're going to go back to our handle input change function inside in the methods object right here.

Remember, all the functions inside of methods are intended to somehow update our data and any time we update our data, it's going to cause our Vue instance to update the HTML that is displayed inside the browser over here.

At present, we're just taking the value that the user types in which is event target dot value and we're logging it out.

So that's not incredibly useful right now.

I think we should probably remove that control log and replace it with something that's going to update our data's input content property right here.

Remember, input content is supposed to reflect the current value of the text input right here.

We can kind of imagine this data property of input content as being like our list of ingredients.

And based on what the value of input content is inside of this data object, we will do different things to render different stuff out onto the page.

OK, so I'm going to take that console log.

I'm going to remove it and then to update the value of this input content data property right here, we will write out this dot input content equals event dot target value.

Now, one thing that's very important here that I want to point out is that we simply said this dot text input.

We did not say anything like this dot data dot input content.

We only use the word data when we are first initializing this data property.

But after that, to refer to any of the properties that are initialized inside of here, we simply say this dot and then the property name.

OK, now what do you want to be really crystal clear about?

Because I mentioned this in passing one or two videos ago, and I also just mentioned it in passing inside of this video as well, is that any time we update the value of one of our data properties, it causes our Vue instance to automatically render inside of the browser window.

At present, our template is not making use of this text input property at all, so you don't see any changes over here.

But as soon as we start updating or making use of text input inside of the actual template, you're going to see that any time we run this handle input change function, everything inside the window over here is going to automatically update as well, which is a very nice feature.

We don't have to forcibly say, please rerender my application or anything like that.

OK, so just to bring everything full circle now, I want to go back to the diagrams we are just looking at and kind of walk through the process of what we've done now with the data property and the handle input change function right here.

So, I'm going to go back to this diagram where we had said that we have some initial starting data of input content equal to empty string.

So that's how our application first starts off.

So, we start off with input content being empty string.

We then sit around and wait and then at some point in time, that method called handle input change is executed.

So, the user enters some text and inside there we update the value of input content.

So, let's imagine that maybe the user enters in text of Vue JS.

So, after they enter that text input in or we just enter that text, we're going to update the text input value.

So now data is instead Vue JS when that gets updated.

So, if input content is updated, we then rerender the template.

So that happens automatically.

And then when that template is updated, we look at we say whenever the template is rendered or updated, we calculate the image by using one of our computed functions which we've not yet defined, but we will in just a second.

So, I know that some of this stuff is awfully confusing and honestly, I really wish I could present it to you in a much more linear fashion.

But with a Vue world, you kind of have to kind of take all these different topics in all at the same time and understand them at the same time.

It's kind of challenging to to study them piece by piece, one at a time.

Nonetheless, we are going to talk about how this data stuff works and methods and computed, non-stop throughout this course.

So, if it seems confusing right now, don't sweat it because we're going to get a lot of practice with it.

So, we've made some good progress.

Let's continue the next section where we're going to start working on our computed property and hopefully start to bring everything full circle.

So quick break and we'll see you in just a minute.

16-Computed Properties

We've now got our data property being initialized here, and we've got some way to update our data over time whenever a user does something inside of our application.

It's now time to turn to the last step here, which is to take our data and turn it into an actual viewable value.

Now, to be entirely clear, our data is already in kind of like a viewable value, right?

Like it's a plain text string.

Obviously, we can show it inside of our template and have our users kind of view the output.

However, that's not what we want our users to view.

We don't care about showing them the text they just entered right here.

We want to show them an image generated by input.

So, we need to do some processing of our data before it gets displayed on the screen.

And any time that's the case, any time we want to do some calculation on a value before it gets showed on the screen, that's when we make use of a computed function.

So inside of this computed object right here, we're going to define a new function and I'm going to call it set image like so.

So, any time set image gets called, you and I are going to return something that's going to get the image to be available or viewable on the screen right here.

Now, as a quick reminder, we will use an API called robohash. I showed you before start codding on JS Fiddle.

So, we're going to write some code that's going to make use of this API right here to produce the actual image based on the text input that the user has provided to us.

So, inside the set Image function, we're going to make use of that robohash API.

So, to do so, I'll use backticks because I will right out a string include an html line.

And then we copy and paste this image htm over here.

Now, we'll pass in the input Content because that's kind of like the seed of sorts from which are image and gets calculated.

Remember that we can reference any of these data properties by simply saying this and then the property name.

So, I will delete YOUR\_TEXT and replace with string interpolation to pass in the input content. I will simply say this dot input content.

Now the very last thing we need to do is return this calculated, or this computed value from this function so I can make sure I get the return keyword inside there as well.

OK, so I know that at this point, we still don't really see any change inside of our output over here or out inside of the HTML that represents our application.

So, to get this identical function running and to displayed on the screen, we have to actually reference this function from within our template, which is inside of the HTML panel over here.

To call a computed function from within our template or to somehow get that information to appear on the screen, we can use a little bit of advanced Vue syntax.

So, underneath where we have output right here, I'm going to reference this computed set Image function right here, by writing out curly brace, curly brace, so notice that I've got like sets on both sides right here and then inside I'll say simply set Image like so.

All right, so you'll see a bunch of kind of like crazy strange text right here, and it might look like this is not what we want, but in fact, it's very close to what we really, truly do want.

What you see right here is an image tag.

So, in this case, whenever we make use of that set function right here, called to image, as you might imagine it returns an image element.

But when we try to show that inside of our template, we just see the actual raw HTML that represents the image tag. To get this to show up as a real piece of rendered image, we're going to use another Vue directive.

So rather than putting out the curly brace, curly brace, set image which simply prints out all that raw HTML, I'm going to instead replace this with another directive very similar to the one we did up here.

So right underneath the output I'm going to write v dash html equals then double quotes and set image and then I'll close off the div like so. Then eventually you see that robot image appear on the screen.

Now I know that that last step there where I said, OK, well we don't really want the image raw text to appear here.

We want to instead show the HTML.

This might seem a little bit confusing.

So, we're going to come back to how this works right here in just a second but before we do, let's try entering in some text here.

As you start to type stuff in, you'll notice that the image is changing every single time and we get some new image appearing on the screen.

So that looks pretty cool.

But I think that we've still got a lot to understand about how this application is truly working behind the scenes.

So, let's come back to the next section and we're going to do a very big review to describe exactly how

this application is working right now.

17-Review from Start to Finish

In the last section, we got our application working, but I think you'll agree with me that there's still some confusion over how everything works right now.

So, in this section, we're going to walk through a big timeline diagram that's going to give you a better idea of exactly what is occurring inside of our Vue application and it's going to really tie everything together.

So, let's get started.

At the very top we first start off by creating our Vue instance.

When that instance is created, the data property is evaluated.

So that data property is this right here.

Vue sees that we are providing an object that has a property named input content with a value of empty string because we are initializing this data property right here.

Vue is going to take that and it's going to assign it to the value of this accessible inside of our computed functions and our methods functions.

After that value is initialised, our template inside the DOM is then evaluated by Vue and eventually rendered onto the screen.

So that's when we see some starting content appear.

We then wait for some user to type into our text input.

The instant they do, our handle input change method is executed and inside of that, handle input change method, which is right here, we update the value of input content variable.

That's when some interesting stuff starts to happen.

So, it's when you start to update these data properties that have been assigned to this that you start to see some interesting behavior inside of your application.

So, in that value gets updated our template gets automatically rendered to the screen.

So, Vue says, hey, someone just updated that value.

I need to automatically rerender the template and update content that is visible on the screen. During that process Vue looks at our template and it sees that our template references a computed property when we put together that directive right underneath output right here.

So, we added in the directive of V HTML, which we're going to expand upon in just a moment and inside there we reference the set Image computed function.

So set image and right here is a reference to the computed function, set image right here.

So, because we add this reference right here, Vue says, OK, we need to go find that function and execute it and then whatever is returned from that function will be provided to this v HTML directive.

So, it seems that set Image property, it gets executed and that set image function returns some raw HTML as a string.

So that was just a raw string that had an image tag inside of it.

But as you and I very well know, we don't want to show raw HTML to the user. That's not very useful.

We want to show an actual image.

So, to get Vue to interpret that string as being actual HTML and not a plain old string, we used the V HTML directive.

So, this directive right here, V Dash HTML says we're going to give you a snippet of HTML.

It's going to be a string, but it's going to look like HTML.

You need to interpret that as HTML and render it as HTML as the output. Don't try to just print out the string.

What we did before where we use those curly braces, like that.

This is how we just print out a very simple string right here, and that's not what we wanted to do.

We wanted to somehow get our actual image to appear as some rendered HTML.

And you'll notice that this is not quite showing up properly right here for a comment out the div. Let's cut that out for just a second. And when I do, we'll instead see. OK, that's better.

There we go.

So, there's that actual output text right there.

So again, we don't want to see that HTML.

We want this to be interpreted as being actual HTML that needs to be rendered and that's why we use that directive right there.

Now, what do I want to mention very quickly is that we don't traditionally use this HTML directive right here very frequently unless you are working on an application where you need to make use of like some generated HTML.

So, it's much more frequently that we use other methods for displaying content on the screen.

It's just in this one very particular case that we're using this directive.

OK, so hopefully that's a little bit more explanation and makes what's happening inside of our application a little bit more clear.

There's still some side topics I want to address about this application.

So, let's take a quick pause right here and we'll continue in the next video.

18-Template Placement

In the last section, we reviewed our entire application and got a better idea of how it's working. In this section, I want to start going over a couple of quick odds and ends around on the application we just put together.

These odds and ends are going to be a couple of side topics that are going to help flesh out your knowledge of some of these very basic Vue topics and make some ideas a little bit clearer before we start working on our next application, which is going to be much more complex than this one.

So, the first odd and end of sorts I want to tell you about is the placement of our template.

Remember that many videos ago we had said that we have our Vue template in our Vue instance.

And I'd said that this was one possible way of structuring a Vue application.

I had said that our Vue template does not have to be created on the HTML side of our application and that we could instead declare that Vue template in the JavaScript side instead.

So, I want to show you a very quick example of that, and it's a very quick and easy refactor to do.

So back inside of my code, I'm going to take all the HTML that we had placed inside of our div with ID root.

I'm going to highlight all of it and I'm going to cut it all out.

So, notice that we still have the wrapping div inside of here that still exists. We have not deleted it.

I'm then going to take the JS panel and expand it and then down at the bottom I'm going to add a comma, I'll add on a new property called Template and I'm going to use a pair of back ticks here.

So back ticks are not normal quotes. So, this is not a single quote. It's not a double quote. It's a back tick.

It's a character on your keyboard or the key that's on your keyboard left after the one. Then inside of here I'm going to paste all that HTML that we just copied.

One other thing, I'm going to make sure that I also wrap this template or all the HTML inside of it with a div, just like we had previously.

The reason that we did not copy that div itself was that we need both divs inside of here.

So, when we make use of a template that is attached directly to a Vue instance, we provide a string that contains some amount of HTML.

Inside that string, we have to have exactly one root element.

So that's why we just added this div inside of here.

If I remove that div, so the opening tag and the closing tag, you'll see that we only see the H two right here.

That's the only thing visible. So that's why we wrap it with this root div.

We are expected to only return one element or one root element inside this template string.

So, you'll see very plainly that everything works the exact same way that it did before.

I can still type some text inside of here and I see some content appear on the screen.

So, like I said, different ways to structure your Vue application.

You can either create your template inside of your HTML or you can create the template and attach it directly to your Vue instance.

Now, in reality, the vast majority of applications that you're going to be working on is going to take this type of approach right here. Where we define our template very closely tied to our Vue instance rather than creating it over inside of our HTML.

The reason I showed you this approach at all is that all the documentation that you're going to see on The Vue official docs is going to take this type of approach. Where they show you one snippet of HTML, and it really is like HTML and then separate from that, they'll show you some code for The Vue instance.

So again, that's why I showed you this approach, even though traditionally we usually attach the template directly to the Vue instance.

OK, so that was the first little odd and end I wanted to tell you about.

Let's now continue in the next section and talk about one or two other quick topics.

19-Referencing Data in the Template

In the last section, we spoke about one quick odd and end. In this section we're going to cover another small topic that I just want you to be aware of.

So, in this one, I want to focus in on the fact that we used a computed function in putting our application together.

And what they want to be very clear about is that our template can show data that has been assigned to our data property.

We don't have to always use computed functions. In general, we only use a computed function any time that we want to somehow mess around with some piece of data before it gets shown on the screen.

So let me show you a good example of this. Back inside of our application. Let's say that we want to dramatically change how our app behaves.

Let's say that maybe instead of entering in some input and then getting back in image, we want to just echo back our text directly to the user.

So just so like, if I enter in Hi there’, the output should be ‘Hi there’. No change whatsoever.

If you want to print out some value that is assigned to data without making any change to it whatsoever, you don't have to use a computed function.

Instead, you can reference your property or your data name directly from within the template.

So, to do so, I'm going to scroll on down to where my template is now defined.

Inside of here, you'll notice that I still have that div with the directive of the V HTML.

So, I'm going to say I don't want to show the image anymore, so I'm going to delete that entirely. Now instead, I will directly reference the text content property and just try to print it out directly into the template.

To print out a direct property without trying to do any fancy directives or anything like that, we place those double curly braces and then inside there we just write out the property name that we want to display.

The name that we put inside of here can be either the name of one of our computed functions or the name of one of our data properties.

So, in this case, I just want to print out input Content directly and not do anything else fancy with it.

So, inside the double curly braces, I'll write out simply input content.

Now, if I start to type inside of my input again, you'll see that it gets directly mirrored right underneath it because I'm just taking one of my input value is in printing out directly inside the template.

One thing I want to mention inside of here very explicitly is that when we reference this input content property, we did not have to do anything like, this dot input content and we do not have to do this dot data input content or anything like that.

We simply say the name of the property, which is input content.

Now, one thing that that kind of like one issue that kind of brings up. You want to make sure that whenever you are defining computed functions right here, you don't accidentally duplicate the name of one of your data properties.

So, I would not want to make a computed function name something like input content, because then it would be really ambiguous as to whether inside my template, I'm referring to this data property right here or one of the computed function names.

OK, so I'm going to take that back to be set Image instead.

All right.

So hopefully that clears up a little bit of confusion there.

So, we don't have to use a computed property if we don't want to.

We can just reference the data name directly and that will be printed out straightforward inside of our template.

Now, I am going to revert this change right here because I do personally like the image that gets displayed.

So, I'm going to go back to the Div with the HTML directive.

OK, so it looks better.

Let's continue in the next section.

And there's just one last thing I want to mention about this application before we move on.

20-Expressions in Templates

In the last section, we spoke about one additional odd and end. In this video, we're going to close off with one very closely related topic.

So here it is, odd and end number three.

So, when we were talking the last section about using those curly braces to display some value inside of a template that made use of a technique called string interpolation.

So, whenever we use string interpolation, that means that we are kind of injecting a value directly into our template by using those curly braces any time that we use those curly braces, we are not only limited to referring to a simple data property or one of those computed properties.

If we want to, we can put in some tiny amount of JavaScript logic into those curly braces.

To be more precise, we can put exactly one JavaScript expression so we can write out, say, like a full if statement.

We can't write out a for loop or anything like that. Let's go back over to code editor and I'll show you a good example of this.

So, I'll go back over to code editor and I'm going to go back down to the very bottom of my template again, where we've got that set image being printed out inside the div by using the HTML directive.

I'm going to again delete that, and I'll replace it with my double curly braces and then I will reference inside of here about the input content piece of data again.

Now, if I type something out inside the input, just as before, I see some text appear right here. So right now, we're simply referencing input content but if we want to, we can add a very limited amount of JavaScript logic inside of these curly braces if we want to.

So, for example, we could do a little bit of string concatenation. So, I could say text input plus Vue JS and then when my app first renders text input is an empty string.

So, we add empty string to Vue JS and the output is simply Vue JS.

If I now start adding in some text, it takes Hello and adds Vue JS onto that.

We're not only limited to say adding text on like this, but we could also do something as complicated as, say, reversing the string.

So, a very simple way of reversing a string with JavaScript would be to split this into an array by every character, to then reverse that array and then to join the result back together like so.

So now if I enter some text in, all these transformations will be applied to that string.

It's going to split it into an array, reverse the array and then join the array back together into a string.

And as you can see up here, the output is completely reversed from what I entered.

So if I put in ABCDE, so on, it gets reversed into EDCBA.

Now, one thing I want to point out here is that while we can add some amount of logic directly into the string interpolation, the reason I showed you this idea of computed properties so early in the course is to direct you towards using these computed functions inside of your templates.

I only mention the fact that we can put some little bit of logic into these little interpolation locations just to be complete and just to tell you, hey, you can do this.

But in general, if you need to play around with any value or format it in any way before you print it out, then I highly recommend you instead make use of one of these computed functions instead.

So as a very quick example, we could add in another function here called something like reverse, then inside of here, we could return this dot input content, split, dot, reverse dot join.

So that's doing the same reversing operation as we were before but now inside the template, I'll reference this reverse function rather than trying to stuff all this reversing logic directly into the template itself.

So now back down inside the template, inside the curly braces, which we always use to interpolate values unless we wanted to use that voice HTML directive, which, like I said, is very rare.

So now inside the curly braces, we can say simply reverse like so and now if I enter some text, you'll see that it gets automatically reversed for us.

I highly recommend using those computed functions any time you need to play around with some data before it gets printed out, just because it keeps your templates very, very clean and very easy to read.

It also makes these computed functions much easier to understand because you can more easily document these functions up here by adding in comments, whereas adding in comments is a lot more laborious inside of your template.

OK, so there's just a little bit more on how we can kind of play around values with values inside of these little string interpolation locations.

So, I think that's pretty much it for stuff we need to talk about with this example.

I'm going to revert all my code here and go back to the actual image generator implementation.

So that's pretty much it for this app.

We're going to continue the next video by discussing the first big application we're going to create and of course.

Currently, if anything seems a little complicated or confusing. Honestly, don't sweat it just yet, because all the topics that we covered in this first application, we're going to cover again and again throughout the rest of this course, because everything we've done on this app, frankly, is stuff that you repeat all the time.

Yeah, initialize some data.

You handle user input, and then you somehow manipulate that data before printing it out inside of your application.

We're going to do this over and over again.

So let's take a quick break and we'll continue with our next application in the next section.

21-App Overview

In the last section, we wrapped up our robohash project and in this video, we're going to start talking about the next application that we're going to be working on inside this course.

This next application is a lot more ambitious and it's going have a lot more features and complexity to it.

I think that we're going to learn a lot while building it. So, let's get started.

We'll first take a look at the mockup of what we're going to built.

So, we're going to make a sort of YouTube browsing application.

This application is going to allow users to enter a search term up at the very top here.

So, this is going to be a little text input where a user is going to enter in some text.

They might search for a term like, say, JavaScript.

Once they submit that, well, then use the YouTube API to search for the term they just entered.

Well, then get back a list of videos that match that search term.

We'll print that list of videos on the right-hand side of the screen over here.

Then any time a user clicks on one of these videos, we'll show a big detail here on the left-hand side where the user can then play the video and they'll also see a quick description about the video itself right underneath it.

This is going to test our knowledge of you, and it's going to make sure that we learn many different aspects of how working with Vue occurs inside of a normal application.

So now that we understand what we're going to make, let's continue in the next section where we're going to start talking about some of the big challenges and solutions that we might come up against inside this application.

22-App Challenges

In the last section, we took a look at a mockup that describes the next application that we're going to be working on. Now We will continue by talking about some of the big problems that we're going to have to address as we start working on this app and also, of course, talk about some possible solutions as well.

So, I think that there's probably four maybe more big issues that we're probably going to run into overtime.

So, we'll talk about each of these.

And I'm going to propose a different solution for each.

And the first big problem is perhaps one of the more obvious ones.

You and I need to somehow search for videos on YouTube, given a search term. To do so, we're going to sign up for the free YouTube API.

We can use this API to take some search term provided by the user and do a search on all the videos that are hosted on YouTube.

This API will then return a list of videos that match that search term, along with thumbnail images, titles, descriptions, all that kind of good stuff.

The next big problem that we're going to have to address is the fact that we want to somehow show a collection of different elements on the screen.

In the last stop you and I put together, we had like that one simple text input and then we had the image and appear on the screen.

And so that was straightforward in nature.

But in the next step that we're going to build the YouTube app, there's a lot more stuff going on, on the screen at any given time.

As we start to make more complex Vue applications, we start to isolate different parts of our app and create them in separate components inside of our Vue code.

Each of these components will be responsible for a different area of the screen.

So, you and I might make one component to implement the search bar up here at the top.

We might make one component to show the list of videos over here on the right-hand side.

And then one component to show the big video detail in the middle.

We'll talk a lot more about what components are and how we're going to use them to implement this application.

The next big challenge is the fact that we need a place to write our code and then somehow run it.

So so far in this course, we've only made use of JS Fiddle, which is only intended for very small projects and just kind of experimenting with little snippets of code.

So, to give us a more full featured boilerplate of sorts or a project that can be more easily expanded to handle large Vue projects, you and I are going to use a tool called Vue cli, and we're going to use this to generate a new project.

The project that gets generated is going to have a lot of preconfigured stuff already added to it, that's going to make working with new projects much easier and more straightforward than if we tried to do all this from scratch.

Now, the last big challenge that we're going to run up against is the fact that we have to somehow handle user input, like when a user enters in some search string into the search bar at the top or when they click on a video with the attempt to play it.

In the last application we worked on, we did look at how we somehow handled these events or handle input from users. So, we're going to take all those same ideas around handling events with Vue directives, around adding methods and around making use of computed functions to get all this stuff to work properly and handle user interaction with our application.

OK, so that's pretty much some of the big four issues that I can foresee in the app that we're going to be working on.

So, let's continue the next section where we're going to start by generating a new project by making use of Vue cli.

So quick break and we'll take care of that in the next video.

23-Generating a New Project

In the last section, we spoke about how we're going to make use of a tool called Vue cli to generate a new project boilerplate.

We're using this tool because setting up your own Vue project from scratch takes a little bit of time and it's not the best thing to get started with when you're first trying to start learning Vue, as we are right now.

So, we're going to install the VUE CLI at our terminal and then we'll use it to generate a new project.

So, let's do that right now.

I'm going to change on over to my terminal and then I will install of VUE CLI by running the command NPM install dash G @ Vue cli like so and then I'll go ahead and run that command.

Now while that's being installed, I want to see the documentation for Vue cli very quickly.

So, here's the documentation at GitHub.

It's that GitHub dot com slash Vue js dash Vue dash cli.

There's not a tremendous amount of documentation on here because VUE CLI is still in active development right now.

You can find a link to the full documentation link in read me section and once you're over here, you'll see some information about how to do further configuration of the project.

OK, so feel free to take a look at the documentation there if you wish.

Otherwise, let's go back over to our terminal where you'll notice that the installation is now complete, or at least it is for me.

We can now use this tool to generate a new project.

So, I'm going to make sure that I'm inside of some directory where I feel like I can have kind of a workspace folder of sorts.

Make sure that you are in a similar folder as well.

Then inside of this directory, I'm going to run the command Vue create and then we're going to put down the name of our project, which in this case, I'm going to use a name of youtube-browser.

So that's what it's really doing here.

We're using this application to browse YouTube videos and then run that command, and that's going to automatically generate the new project and start to install dependencies.

When you run the command, you might see a couple of quick questions like this right here.

And if you do just select any of the default options that are represented.

So, I'm just going to say give me the Vue 2 default with babel and e s lint.

OK, and there goes the installation.

Let's take a quick pause right here, and when we come back, we'll talk about what we are getting when we install this default project here.

You know, what's this really doing for us?

So, let's talk about that in the next video.

24-Why Use Vue CLI?

In the last section, we made use of VUE CLI to generate a new project. It looks like my installation is now complete and I'm given the directions to run this command, to change into that newly created directory that contains our project and then to run the command and npm run serve to start our project up.

Before we do that, I won't have a quick discussion on exactly why we are making use of this Vue cli project generator.

Like why do we need all this infrastructure and all this preconfigured stuff anyways, let's take a look at a diagram to get a better sense of why.

OK, so this is an illustration of how you're just running code a little bit ago by making use of JS Fiddle.

When we used JS Fiddle, we automatically loaded up the Vue js file by adding it as a dependency to the project.

We then also wrote a little bit of code directly on JS Fiddle and these two together formed a usable application.

The key thing there to keep in mind is that all the code that you and I wrote, like the JavaScript code in the HTML, we're all inside of essentially a single file authored directly on JS Fiddle.

But when we start working on real projects, we don't really get to enjoy the fact of only having to create one single file. With real Vue projects, we much more frequently make a collection of files like many different files that form one single application.

And that's where this added complexity starts to come in.

That makes us want to use a tool like Vue cli.

So, the Vue cli, we get the ability to instead make many different files, each of which might have one individual or distinct purpose inside of our application.

So, we might have one file that has some code for fetching videos and then maybe another one that's in charge of that search bar at the top of our application, another one for playing a video and then one more for maybe showing a list of videos to the user.

So, all these different files are going to come together to form our one usable application.

Here's the thing about JavaScript, though.

With JavaScript out of the box, with the JavaScript language, we don't really have a great way to take all these different files and combine them down to one single file that we can then send to the user's browser and have our application start up and be executed.

So, take all these different files and combine them together, we make use of a tool called Webpack. You may have heard of Webpack before.

Essentially, its purpose is to look at a big collection of files and possibly dependencies as well like Vue js, and combine them all down to one single file, which can then be sent down to your user's browser when they try to make use of your application.

So, this Webpack tool is one of the big reasons that we're making use of Vue cli. Webpack is somewhat notorious for being a little difficult to set up out of the box.

And so, by making use of VUE CLI, it automatically sets up Webpack for us and chooses a lot of very sensible configuration defaults. Along with VUE CLI, we also get access to babel pre-configured for us right out of the box.

You may have heard of Babel before as well.

Babel allows us to write 20, 15, 16, 17 and so on, code that can be safely executed inside of a user's browser, even if their browser does not have support for a particular dialect of JavaScript that we might want to use inside of the code that you and I write.

So just like Webpack, babel is also included automatically inside of VUE CLI, and it has a lot of very sensible configuration defaults already set up for us.

So that's why we are making use of VUE CLI in general.

It does all of a set up for us and we don't have to waste any time kind of laboring over documentation and figuring out how to set that stuff up.

OK, so now we've got a better idea of why we're making use of VUE CLI at all.

Let's continue in the next section where we're going to start up our application and get a better sense of all the different files and folders that were created for us automatically when we just generated the project.

So quick break and we'll see you in a minute.

25-Project Walkthrough

In the last section, we spoke about how we get Webpack, and babel set up for us for free whenever we make use of VUE CLI.

I'm now back over at the terminal where I'm going to change into my newly created youtube browser directory and I'm going to start up the project.

So, I'm going to change into video browser and then I will start up the project by running the command NPM run serve.

Throughout the rest of this course, we're going to be using this command to start our project up.

So, if you ever stop the course, you ever take a pause or go to sleep. I don't know what people do.

And you want to start the project back up.

You can just run npm run serve again.

Now, when we run that command, you'll see some information about starting up the development server.

So, when we run NPM Run Serve, it starts up a local server that starts up Babel and Webpack takes all of our project files and bundles them together into one single JavaScript file where it can then be served up into the browser.

You'll notice that you might get an automatically open browser window and it's automatically open window is pointing at localhost eighty eighty, which is where our project is hosted.

So now I could say refresh the page and you'll see the exact same map appear on the screen.

You'll also notice that there's a little bit of default content already visible on the screen here.

So, this is a little bit of Vue code that has generated the content you see on the screen, and this is code that is given for us automatically when we generate our new project.

Let's now open our code editor inside the project directory and take a look at some of the different files and folders that were generated for us.

So back over at my terminal, I'm going to start my code editor up.

I'm going to make use of the code editor, vs code, throughout this course but you're free to use any editor you want.

And I would say that every editor can pretty much handle Vue as well as any other. So don’t feel like you have to use any one particular editor to have a good time with Vue.

OK, so now on the left hand of my screen, you'll see some of the different files and folders that were created for us automatically when we generated our project. Inside the node modules directory is a list of all the different dependencies that our project depends upon.

Underneath that, you'll find the public directory.

Inside this folder is a very important file. It's our index dot HTML file.

Any time someone tries to visit our application by coming to our server in their web browser, even as we just did right here, this index dot HTML file is always going to be loaded up automatically.

Inside this file you might notice a div with id app, just like we wrote a little bit ago inside of JS Fiddle.

This div right here represents the root location of where our app is going to be booted up.

Then inside the src directory, we'll find a bunch of different files and folders that are related to the actual implementation of our Vue app.

We're going to be spending most of our time inside this src directory.

I want to talk about some of the different files and folders that are located inside of here, because you'll very quickly notice that there's a very big difference between some of the code that you and I were writing earlier on, JS Fiddle and some of the code that you're going to find inside these files.

Let's first start off by looking at the main dot js file.

Inside of here, you'll find some very plain JavaScript code, and towards the bottom, you'll find a function call to new Vue, which we use a little bit ago, to create a new Vue instance over on JS Fiddle instance.

Then you'll notice that there's also this render option in here and also a dollar sign mount function call as well.

We'll just ignore those two little pieces of code for right now.

They will come back to those very shortly and talk about what their purposes are.

OK, so now here's the weird part.

Here's where things start to get really interesting.

You'll notice that also inside the src directory is an app dot Vue file.

Notice the extension there, its dot Vue, rather than being like dot JS or anything else like that.

Let's open that file and see what's inside of it.

At the top of this file, you'll notice that there's what looks like a little bit of HTML.

All this HTML is nested inside of a template tag and then below all that HTML, you'll find what looks like another piece of HTML, a script tag, and inside of there is some JavaScript code.

So, I don't know about you, but this looks like one weird little file.

So, let's talk about what's going on here in the next section.

A quick break and we'll cover this Vue file and it's kind of strange syntax.

See you in just a minute.

26-Vue Files

In the last section, we started looking at some the different files and folders that were generated first automatically when we made our new project. In particular, we open up the app dot Vue file.

Inside of there, we saw some very interesting looking syntax.

So, at the top we see a template tag and below that we see a script tag.

So, let's talk about what's going on here.

First off, quick reminder of what we were doing previously over Inside of JS Fiddle.

So back over here, we had said that this was one possible way of structuring of Vue app.

We might write out our Vue template inside of some HTML, like an actual HTML document and then we also might create a Vue instance over in some JavaScript code.

We then later said that we could optionally also create this Vue template inside of our JavaScript code as well and we saw an example of that over in JS Fiddle when we use those back to characters to insert our template directly attached to our Vue instance.

So again, these are both two possible ways of structuring a Vue application but what you're seeing over here inside of this app dot Vue file is yet another way of structuring your Vue code.

So, this takes use of a kind of paradigm called Vue files.

The idea behind a Vue file is that you'll have exactly one file that is responsible for creating a single component inside of your application.

A single component is created as a reusable piece of code that can be used all over your application many times. Inside of this single Vue file, you'll find not only the template that your Vue file or your Vue component is going to use, but you'll also find all the JavaScript code related to it as well.

So, in other words, inside of one single file, we get access to all our HTML for this Vue component.

We get all the JavaScript for it and in the future, we're not doing this right now but very shortly we will, we'll also locate all the CSS for this component as well.

So, one file that contains all the code related to one component or one piece of our application. As we start to implement our YouTube browser application, you and I are going to end up with a couple of different Vue files, each of which are responsible for implementing one distinct part of this application.

So, we might make one Vue file that is responsible for the search bar up here at the top.

We might make another single Vue file that is responsible for this list and then a single Vue file that is responsible for the big video detail here in the middle that's going to eventually play a video.

So, again, the idea is that with one single file, we have all the HTML JavaScript and CSS all directly placed inside there.

Now, you might think that this violates one of the core principles of writing Web applications.

The idea of separate concerns. The idea of separate concerns means that we try to not mix and match all our HTML, JavaScript, and CSS together because that could possibly make a real big mess of our application.

But one thing I want to point out here is that even when we start making use of Vue files, even though we've got our template and script and your CSS right here inside the style tag, even though this is all inside a single file. Well, technically it is still somewhat separated because inside this file, we've got one area responsible for HTML, one area responsible for our JavaScript and one area responsible for our CSS as well.

So, yes, they are all located inside of a single file, there's still at least somewhat isolated inside this file.

One of the big benefits to making use of these Vue files is that if you ever need to change your application in some fashion, it'll be a lot easier to find all the related code to one area of your app.

For example, if we wanted to make a change to the way that the search bar right here.

We know that we could always open the Vue file that contains the search bar implementation and inside there will find the HTML, JavaScript, and CSS.

And that means that we don't have to go hunting around different directories inside of our application to find all the different aspects of code that are related to the search bar.

So, it might take a little bit of time to get used to this Vue file syntax, but over time, I'm very confident that you'll probably come to enjoy it.

OK, so we've spoken a little bit about the purpose of The Vue file at this point, but we haven't really spoken about how this strange syntax is taken and used to build an actual application.

So, let's take a quick pause.

In the next video, we're going to examine some of the behind-the-scenes stuff that occurs to eventually get this code to run inside of your browser.

27-Behind the Scenes of Vue Files

In the last section, we started talking about Vue files inside of a single Vue file, we will locate all the code related to one distinct portion of our application.

We're going to use Vue files in place of making a separate Vue instance that are already created in the DOM.

So, again, I showed you this methodology right here using JS Fiddle because this is how a lot of the documentation is put together.

But the vast majority of professional Vue projects that you'll probably work on in industry are usually going to make use of these more complicated Vue files. Just because they are easier to expand and scale into large applications.

Rather than placing all your Vue templates directly into some HTML.

In the section I want to answer a question that you might have right now, and I question that I'm assuming that you might be thinking about is ‘How in the world is our browser going to execute or somehow understand this code right here?’

So, in fact, what you see on the screen right here is not what gets sent down to your user's browser.

Instead, babel in Webpack, work together to somehow convert all the contents of this file into some very plain JavaScript code that your browser can safely run.

So, in this video, I'm going to give you a quick demonstration of what this file turns into when it gets sent down to your browser.

Now, for this demonstration, I really recommend that you just watch what I'm doing because I'm going to click through a couple of steps very quickly, so you don't have to watch me very laboriously click around a whole bunch.

So, let's get to it.

I'm going to show you what this file ends up like when it gets shipped down to your browser.

I'm going to first go back over to our application and I'm going to open the request log.

Then I'm going to go back over to my code editor and I'm going to make one very small change to this file and then save it.

Once I do that, I'll go back over to the request log.

So, here's a whole bunch of JavaScript code that represents that change that we just made.

I'm going to copy it back over to my code editor.

I'm going to find and replace newline characters in here very quickly, and then I'm just going to delete a tiny little bit of code.

All right, let's do that delete. Here we go.

So, I'm going to delete that and I'm going to delete all that as well.

OK, so like I said, I just very quickly want it to go through that change just so we can very quickly see what that file gets turned into.

So, this render function right here is what our App Vue file was converted into before it got sent down to our browser.

This right here is the actual code that is running inside your browser that represents that Vue file.

Now, before we walk through the code inside of here too deeply, let's go back over to the App dot Vue file very quickly.

Inside the template up here, you'll notice that we are creating a div with an idea of app.

And then inside there we've got an image with a src logo dot png.

And then we've got this hello world tag thing right here that has a message of welcome to Your Vue JS app.

Now, with that template, now that we've got a better understanding of it, let's go back over to the file that I just created.

So inside of here, you'll notice a couple of strange variable declarations at the top. We’ll ignore those for just a second. Instead, I want you to look at the function call right here to underscore C.

Inside this, we have a first argument of div.

So, this word right here, div represents this top level div tag right here inside of our template.

Then the second argument is an object that says that div should have an ID of app and then the third argument is a list of other elements that are contained within that div.

So, the first element that exists inside that div is an image tag that has a source of logo dot png and then the second element is that ‘hello world’ tag that has a message property of ‘Welcome to your Vue JS App’.

So even though the syntax still might look very strange and might not look like any Vue code that we've looked at or written so far, I hope that you can at least get the sense that, yes, everything inside this file is somehow converted into other structure before it gets sent down to our browser.

So inside of our browser, no, we're not running this code inside of here.

It all gets translated into this other form and then execute it.

So really, the idea of making use of these Vue files that has the template tag and the script tag and the style tag down here is just to make your life and my life as developers a little bit easier and more straightforward.

That's the only purpose of it. It is just to help us organize our code inside these Vue files.

OK, so now that we've got a better idea of what's happening inside of a Vue file, let's continue in the next section and do a little bit more of a walkthrough through some of the code that we have inside of here.

So quick break and I'll see you in just a minute.

28-Vue Components vs Vue Instances

In the last section, we saw how these dot Vue files eventually get turned into normal JavaScript code before they are displayed inside the browser.

Now, I know that we've been doing a lot of lecturing in the last couple of videos.

We're going to move on to working on our video browser application in just a second but before we do, there's one last important topic I want to share with you.

So, in the last couple of videos, I've been using the term vue component a couple of different times.

Earlier on in the course back when we were working inside of JS Fiddle, I was using the term Vue instance.

So, I want to make a distinction very quickly between Vue components and Vue instances. All right, here we go.

So, you can think of a Vue component as like a blueprint or a set of rules on how to create something that can be inserted into the DOM, in our browser that the user can interact with.

So, whenever you and I make a Vue file, we are going to define exactly one component inside of it.

And that component is going to have this set of rules that tells Vue how to display some content on the screen and how a user can interact with it.

So again, think of a Vue component as being like a blueprint of sorts.

On the other hand, is a Vue instance, which is what we were making back inside of JS Fiddle. Vue instance is really an instance of a Vue component.

It represents something that has been inserted into the DOM and is something that a user can interact with.

So, I know that the relation between these two things is very tenuous.

If you've got an understanding or a background, an object-oriented programming at all, you can think of a Vue component as being like a class and a Vue instance as being like an instance of that class.

That's the real relationship that's going on here.

So, like I said, we are always going to create one Vue file for every Vue component.

So, one of the very critical steps that we're going to undergo in every Vue application we put together is to plan out the different components that we might want to make.

In general, we like to look at mockups of the applications that are going to make and do some quick brainstorming on how we might assemble a set of different components to make up that application.

So, with that in mind, I took the liberty of taking our mockup for the video browser application and thinking about how I might divide this up into a set of different components.

You'll notice I've added some color boxes on this mockup and on every box of attached a label.

The label that you see here is the name of a component that we're going to make to implement the youtube browser application.

So, you and I are going to make one Vue file called Search Bar.

This is going to have a single Vue component that is going to contain a text input.

And any time a user types inside there, we're going to somehow trigger some searching operation on the YouTube API.

We will also make a single Vue file that houses a component called video list.

This will this video list component will know how to take a list of videos and render them out onto the screen.

Interm will also have a video list item, which will be a single component that represents a single video.

So, the video list component will contain many video list items inside of it because a video list item represents one video and the video list represents the entire list. We’ll also have a Vue file for the video detail, which is responsible for showing details about one single video.

And then all these components will be assembled underneath one component that we'll call the app component.

This app component is kind of like the central brains of our entire application. It represents the single point that kind of organizes these different components and controls how data flows between them.

It's extremely common to always have a single component called App in a Vue application. It's always going to serve that same kind of purpose of serving like the brains of your app.

It's the last thing I want to show you with all these component names in mind is a quick diagram of how they are all related.

OK, so this is the overall structure that you and I are going to eventually head towards as we start to build our video browser application.

At the very top, we've got our main JS file, which is sort of responsible for kind of like booting up our entire application and rendering onto the screen.

That may not just file is then going to show our app component and internally the app component will show the video detail, the search bar in the video list.

And then in turn, the video list component will show a couple of copies of the video list item component.

So that's how all these different pieces are going to be wired together.

Again, we're going to make one separate Vue file for each of these different components.

And each of those Vue files are going to contain all the HTML, CSS, and JavaScript related to that single component.

OK, so that's pretty much it for getting a better idea.

Some of the high-level architecture.

If any of this kind of component stuff still seems a little bit strange, this is another one of those topics where we're going to be doing this breakdown of talking about components throughout the course.

This is another topic that we're just going to come back to again and again.

We'll get a lot of practice and understanding how components are really working.

So, with that in mind, let's continue in the next section and we're going to start working on our app.

So, we'll see you in just a minute.