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 stock media search engine.

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, flowers.

Once they submit that, well, then use the Pixabay 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 beneath of the screen over here.

Then any time a user clicks on one of these videos, we'll show a big detail up here where the user can then play the video and they'll also see a quick description like how many likes it has and download maybe tags of the video on the right side.

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.

Generating a New Project

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'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.

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 slash 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.

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 video-portal.

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

We're using this application to browse Pixabay stock 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.

Project Walkthrough

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.

So, I'm going to change into video portal 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. When you want to start the project back, 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 app 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 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.

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. There is also a special code editor build for Vue js called HBuilder. Maybe you can setup it up your pc and work with it during the course. Like I said, It is totally up to you.

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.

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. We opened 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 backtick 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, 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 Pixabay video portal application, We 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 input 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 show media file 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 show media file over here.

We know that we could always open the Vue file that contains some 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 show media component.

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.

Vue Components vs Vue Instances

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.

Now, we're going to move on to working on our video portal 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.

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 we 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 portal 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 video portal application.

So, you and I are going to make one Vue file called Search input.

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 Pixabay API.

We will also make a single Vue file that houses a component called videos.

This will this videos component will know how to take a list of videos and render them out onto the screen.

In term will also have a item, which will be a single component that represents a single video.

So, the videos component will contain many items inside of it because a item represents one video and the videos represents the entire list. We’ll also have a Vue file for the show media, 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 portal 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 show media, the search input in the videos.

And then in turn, the videos component will show a couple of copies of the 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.

Making the SearchBar

Now, we've spoken a lot about all the code that's in the src directory, and it's now time to start working on our video portal application.

We're not going to create search input component.

Remember, the idea here is that this search input component is going to show an input. Any time user types in there will then trigger a search on the Pixabay API to attempt to go find some list of videos related to that search term.

So, let's flip back over to our code editor, where we're going to create a new Vue file to contain the search input component but first I am going to App dot Vue file and delete everything inside template, script and style tag except export default name App. Then I will delete this Helloworld dot vue file.

Inside the template tag itself right here.

We are only allowed to have one root HTML element.

In other words, we cannot do something like div and another div.

These are two sibling divs. So, in this case we would have two root elements and that is not allowed.

We have to always have exactly one root element, so I cannot do div and div like so, but I could do a div with two divs inside of it if I wanted to.

We don't really want to, but just mean to say, make sure you've always got one root element.

I think that at this point we just want to get some content on the screen.

So, inside this div I'm just going to say, ‘Video Portal App’. I am going to save that and go back over to my browser localhost eighty eighty.

I'll refresh the page and you'll see the text of ‘Video Portal App’ appear on the screen, just as you might expect.

OK, so that's great. App is working perfectly.

Then, I'll find the SRC directory. Usually, we create a components directory inside of here and inside of that directory, we'll locate all the different components we have.

So, let's do that here as well. Inside of SRC, I'll make a new folder called Components and then inside there I'm going to make a new file called Search input dot Vue.

Like so inside the search input Vue file.

We're going to put down some different boilerplate tags inside of here that are going to become very familiar over time.

So, we'll first begin by placing a template tag and then underneath that we'll put a script tag.

And then finally, we'll also put a style tag which will eventually contain some CSS that is solely related to this one component and no other component inside of our application.

Back up inside the templates will immediately create an input element and we'll wrap it with a div.

So, I'm going to place a div and then inside there we'll put an input very similar to the one we had worked with previously back on JS Fiddle.

Now you might be a little bit curious why we are wrapping this with a div.

Honestly, that's just for styling.

By default, a div has CSS display property of block, which will make sure that this input element appears in the browser on its own individual line.

So, the div here is just kind of for layout purposes, not necessarily because this component requires it from any functionality standpoint.

Next, we're going to start working inside the script tag. Inside the script tag, we'll put down export default and then provide an object that's going to specify a couple of different options that we're going to use.

OK, so this looks good.

Let's take a quick break here and then define a couple of different parameters inside this object.

See you in the next one.

Nesting Components

We just put together our search input file and now ready to start adding a couple of options inside of here.

Let's first get started by adding one parameter. The name parameter, which we spoke about a couple of videos ago.

So, I'm going to provide a name of simply search input.

So now, before we go any further, I think that we should try to get this search input component right here to be displayed inside of our application.

So, if you go back over to the browser right now and refresh the page, you'll notice that the search input doesn't appear at all, which kind of makes sense if you want to get any component to show up inside of a Vue application, you have to manually wired up to another component.

So, to get that search input to appear, we have to wire it up to the app component. To do so, I'm going to go back over to my code editor and open the App Vue file.

Inside this file, we're going to add an import statement into the script tag right here.

So right above the export default, we'll add in import search input from and then we're going to provide a relative path to that search input file right here.

So, we'll say dot slash in the components directory, look for the search input file.

Then to get this thing to render inside the app component, we could add a search input tag inside the template.

So, I'm going to remove the text ‘Video Portal App’ because we don't really need that text anymore and I will replace it with search input just like that.

By convention we can do a self-closing tag right here if we do not expect this component to contain any elements of its own.

But in this case, we will do. So, I'm going to use a full tag.

OK, so let's save this right here and we'll go back over the browser and just see what happens.

So now here, if you open your console, you might notice a little error message.

So, this is maybe a little bit of a surprise.

You see, whenever you want to nest a component like this or show one component inside of another, it's not quite enough to just import the components and then place it into the template.

There's one other step that we have to go through as well.

Now, might seem like it's kind of silly of me to skip this extra last step, but I specifically want you to see this error message right here, because the last thing we have to do is something that's very easy to forget.

All right.

So back over inside of our code editor still inside the APP Vue file, we imported search input right here and then we used it inside the template. So that steps one and two.

Step number three, this is the one that you might forget kind of frequently. Inside of our options object that represents the app component.

We're going to add on one additional property called components.

This is going to be an object that lists all the different components that are going to be used inside of the app's template.

So, we have to place search input inside of here so that our app component knows that it might see a search input tag inside of its template.

To do so, we'll add in search input and we're going to have a key of search input and a value of the actual component itself and that component is Search input. We can use a little bit of ES6 syntax to shorten this up because the key and the value are identical.

So, in here we can simply search input. let's now save this, I'll flip back over to the browser and now you'll notice that we get our input on the screen.

If I zoom back out, you probably see something a little bit more like this.

I just usually have my browser a little bit zoomed in so you can see everything more easily, OK.

So again, to show one component inside of another, we add the import statement. We use a tag inside of the template.

And then the last step that I don't want you to forget, and therefore I showed you that error message.

Don't forget to add the property to your components property into the scripts.

That looks good.

Let's continue in the next section and we'll start thinking about how we're going to get our search input file to recognize the user type in some text in here and then do a search on the Pixabay API.

Event Directives

I'm inside of our search input vue dot file, which is now visible inside of our application. If we go back over to the browser, you'll see our input element appearing right here.

So, our goal is to now make sure that any time a user types inside this input, we trigger some type of search for the Pixabay API.

Let's first get started by just making sure that we can get some event or some notification any timeuser types inside of here.

We already did this previously back over on one of our image generator exercises.

You'll recall that we're going to add a directive to this input tag right here.

And then we will also add a method object to our component definition inside of here.

So, let's get to it.

I'll first begin by adding in the methods object to our component definition.

Remember, methods give us the ability to change data inside of our application.

So, I'm going to make a method called on change.

So, let's define the function will be called with an event and that event right there is going to contain the text that the user just entered into the input.

So, for right now, let's just do a quick console log of events dot target dot value.

We'll also wire that method up to our input element right here.

You'll recall that we can do so by adding a custom directive.

So, we'll save v dash on, then a colon and then the name of the event that we want to watch for. In this case, an input event. Then we'll do an equals double quotes and then we'll list out the name of the function or the method that we want to call any time this event is triggered.

So, we want to run the on change method any time an input event occurs.

So, we'll write on change like so.

All right.

So, let's save this.

We'll go back over to the browser.

I'm going to do a refresh of the page and then I'll enter some text in and you'll see the console logs start to appear. Great.

There's one little refactor I want to do to this directive right here.

The refactor that we're going to do is going to have absolutely no effect on what's going on with the on change directive.

It's only a syntactic change.

So, the change we're going to make is just to make this code a little bit more legible. It does absolutely nothing else.

So, we're going to say that rather than putting in the dash on and then colon, we're going to replace all that.

So, everything from V to the colon with simply an add sign. So you can read this right here as meaning any time someone triggers an input event on this element, run the on input function.

Again, this symbol right here is one hundred percent equivalent to writing V Dashon and then a colon.

So why is there this different syntax?

Well, honestly, I think people just got tired of writing V dush on colon again and again as the exact same effect behind the scenes on setting up an event listener on this input element.

In general, I do recommend that you try using this shortened syntax where possible just because it's going to clean up your templates a little bit.

All right.

So, let's take a quick pause right here and we'll continue the next section and start thinking about how we can use this search term right here to do a search on the Pixabay API.

Communication with Props and Events

Now, we have to do is figure out how to get the search input to somehow tell the app component that it needs to execute a search in the first place.

At present, the search input component is just reading in input that is provided by the user.

So, the search input needs to somehow say to the app, hey, we got some new input. It's time for you to execute a search.

So, let's now talk about how we can communicate from a child component up to a parent one.

So, we essentially want to have the search input component right here, communicate to the app that it needs to run a search with a term.

To communicate from a parent component to a child component is slightly different, depending upon which way we're trying to communicate.

So, if we want the app component to somehow provide some information or provide some data or otherwise, just communicate something down to the search input, then we do so by going through a process called passing props.

Props is short for the term properties.

So any time we want to communicate from the app down to the search input, we will pass a prop.

A prop can be anything from an array to an object, to a string, to a function, all types of values and we'll see many examples of this over time. However, communicating from the search input up to the app.

A search uses a slightly different system.

So, to communicate upwards, we make use of the event system that is included in Vue.

So, if the search input wants to say something to the app, then the search input needs to emit an event.

The app component will then listen for that event and whenever that event is triggered, the app can run some custom code.

Like in our case, it would have the ability to run a search on the Pixabay API.

We're going to see many examples of communication from top to bottom or from bottom to top throughout this course.

So, let's take a quick pause right now.

We'll come back to the next video and we're going to start to wire up some events between the search input and the app component.

Emitting Events

In the last section, we spoke about how a parent component can communicate to the child by passing props and how a child can communicate with the parent by emitting events.

So, let's now go back over to our code editor and add some code to the search input to make sure that it emits an event any time someone changes that input element.

I'm going to open my code editor. I've got my search input Vue file open still and then I will find out on-change function.

So, every single time that this function right here is called, we are going to want to emit an event to inform the app that there is a new term to use to search the Pixabay API with.

To do so, I'm going to delete the console log that we have inside of here and then I will replace it with a function call that is going to emit an event. To do so will write out this dot dollar sign emit.

The first argument to this function is the name of the event that we want to emit.

In this case, I'll use the name of new term to indicate that, hey, our search term just changed, we got a new term to handle.

As a quick aside, I want to mention that back up here, when we were listening for an input element or an input event, this is a very special event name.

In other words, input elements, emit events called input and input event, any time someone add some input.

However, when you and I are emitting events, we have complete control over the name that we choose.

So, this right here can be ‘new term’ or it could be anything whatever you want. You can name it as you wish.

But in general, I recommend using the name of what is about to occur, like the thing that is about to have something happen to it, and then a verb that indicates what just we need to do. In this case handling a new term is appropriate.

As a second argument, we can provide some additional information about the event that just occurred.

So, in this case, we're probably going to want to inform the AB component about the new search term, which is available on event target value.

But we can write it a bit cleaner. So, I am going to define a variable called term and this term equals to event dot target dot value. Now we can pass this term as a second argument into emit event just like that.

Let's say I want to mention here is the dollar sign in this emit function name.

Just you know, there's nothing special about using a dollar sign with a function name.

This is one hundred percent valid JavaScript.

So, I should say the Vue authors could have just as easily called this emit, but they decided to put the dollar sign in there just in case you ever decided to add in some additional property to your component called emit as well.

So, the dollar sign here really means nothing.

It could have been any other character.

It does not make the function special or anything like that.

OK, so now every single time someone type something into our search input, we are emitting an event called new term.

Let's take a pause right here.

When we come back, we will work on our app component and make sure that it listens for the search input to emit an event and each time that occurs will then trigger a search on the Pixabay API.

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

Listening for Custom Events

Our search input component is now emitting an event, any time someone enters some new text. We're now going to open our app component and make sure that it listens to the search input.

Any time the search input emits this new term event, we'll make sure that the app has the ability to run some code that will do a search on the Pixabay API.

As a quick aside, we have not yet added in any functionality to do a search on Pixabay.

So, we will need to make sure that we add that in at some point as well.

I'll first begin by opening the App dot Vue file. Inside of here will first add a little bit of code to make sure that the app component listens for events coming from the search input.

To do so, we will add in a directive just like we have previously, to listen for input events.

So, on the search input, we could add in the v dash on then the name of the event that we want to listen for, which is new term.

Then we'll add an equals double quotes and the name of a method that we want to run any time this event right here is triggered.

So, I think that a method name of something like handle new term would be completely appropriate.

Let's now add in that method quickly and then we'll talk a little bit about the syntax right here again. Down inside of my component definition, I'll add in another key value pair of methods and then I will define a function called handle new term.

So, handle new term is going to be a function.

OK, so now once again, any time this event right here is triggered, we will run that handle new term function.

So, here's handle new term. You'll remember that back inside of search input, whenever we emit new term, we also pass along the new search term that the user just entered, because we added this as a second argument right here. It will show up as the first argument to our callback function or the event handler right here.

So, the first argument to this thing will be the new search input that we get.

So, this is going to be a string, the same string that the user just entered that search input for.

Right now, let's add in the council log of search terms.

You might mix this search input term and search input component. I know it confusing, but these are two different things. This search input starts with lowercase ‘s’.

OK, now, before we go test this out, I want to draw your attention to the listener that we added right here.

So, chances are this look very familiar.

It's almost identical to the same syntax. We used to listen to an input event on the input element.

So, the way that we listen to events triggered by normal HTML elements like, say, a button or an input or a text area is completely identical to how we listen to events on components that you and I author.

So, we just wrote out right here V on and then the event name that we're listening for, you'll recall that we can shorten the syntax right here to be simply @, which means the exact same thing.

Like I said previously, we generally want to make sure that we use the same form of syntax throughout our entire project.

And I said, like I said, we'll be favoring this shorter @ syntax throughout all the projects that we work on.

OK, so let's save this.

We'll go back over to the browser, and we'll test out our code to see how it's doing.

So back inside the browser, I'll open our application.

I'm going to do a quick refresh the page and then if I type some text in, you'll again see all the text logged out at our console.

Now, note that this is not the same console log that we had previously before we had our search input component console login, that search term.

But that console log no longer exists over here.

We moved the console log over to the app component.

So that means that every single time a user types the search input emits an event, the app component listens for that event.

And then the app components handle new term method is executed, which is where the console log takes place.

So, seems to me like the search input component is successfully communicating up to the app, which is great.

So now we need to move forward a little bit and figure out how we can use this search term right here to run a search on the Pixabay API.

So, let's take care of that in the next video.

Pixabay API Signup

Our app component is now aware of any time the user enters in a new search term, so inside this handle new term function right here is probably where we will want to locate a little bit of code to somehow initiate a search for some videos.

Let's first look at a diagram that's going to give us a better idea of how we're going to use the Pixabay API and then we'll go look at the API itself.

OK, so every single time a user type something in, we are going to use a library called Axios to make Ajax requests over to the Pixabay API. In the Ajax request we will provide this term that we are looking for.

The Pixabay API will then execute a search on its own servers and then respond to us with a list of videos that match that search term.

In order to make use of the Pixabay API, we first have to sign up for it and get an API key.

Let's take care of that right now to sign up for the Pixabay API.

I'll open up a new tab inside my browser and then I'll navigate to pixabay dot com.

At this page, at the top right corner we click join and then we will see a sign-up modal. You can sign up with any options here. You can use your Google or Facebook account already or create new pixabay account with your email.

It doesn't have to be a real account and we're only going to use it in the context of using this Pixabay API.

When you first come to the profile dashboard, you'll see menu icon on the top right corner. Click that and a menu opens up the choose API link. Now we can go to API documentation.

So, either way, we just want to open up the wizard to create a new project.

We now have to get an API key so that we can actually access the API itself.

OK, very good.

I'm going to select all API key over here and copy the key.

Then I'll go back over to my code editor.

I'm inside of my laptop Vue component and directly underneath our import statement right here.

I'll add in a new variable declaration called API Underscore Key and I'll paste my application inside of a string.

You'll notice that I use capital letters with the underscore right here to declare the variable name.

That's because this is a constant variable and I never expect this to be changed.

OK, so that's pretty much all we have to do.

Well, we'll take a quick pause.

So, I'll see you in just a minute.

Searching Pixabay

In the last section, we successfully signed up for a Pixabay API key.

Remember, We will use fetch method to actually make a AJAX request over to the Pixabay API.

We're going to add some code inside of the handle new term function right here to execute a search on the Pixabay API.

Now, before adding that code in, I do want to point you very quickly to the Pixabay API documentation.

So, if you do a search for Pixabay API, one of the first results that you're going to see is the Pixabay API Documentation right here.

So, this is the actual documentation on how we use the API.

Now, I'll tell you everything you need to know about this document.

I just want to point it out in case you want to read over some of the stuff on your own.

So, let's go back over to our app component and we'll add a little bit of code into handle new term to execute that search.

Now down inside of handle new term method we’ll add some code to fetch method make a HTTP request to the Pixabay API.

To make the request itself, we'll say fetch then we'll enter the Pixabay API specifically for the search and point the URL for this is https colon slash slash pixabay dot com slash api slash videos and question mark.

Why we put a question mark here because after this pixabay api end point we want to make a request with some queries.

API Query parameters can be defined as the optional key-value pairs that appear after the question mark in the URL.

Basically, they are extensions of the URL that are utilized to help determine specific content or action based on the data being delivered.

Query parameters are appended to the end of the URL, using a ‘?’. The question mark sign is used to separate path and query parameters.

If you want to add multiple query parameters, an ‘&’ sign is placed in between them to form what is known as a query string. It can feature various object types with distinct lengths such as arrays, strings, and numbers.

Then as a second argument, we're going to add in a couple of different parameters to configure this request and make sure that it's clear exactly what search term we want to execute the search with.

After that we say plus and create a new URL Search Parameters. Inside this parameter we are going to add an object that take some queries that include our API key inside of here as well.

Let's first take care of the API key. To attach the API key to this request.

Now we will say Key is the API key that we defined just a moment ago.

So, our key is API key then the second thing we're going to add search query. To indicate the actual term that we want to use to search the Pixabay API with will add in Q which is short for query. In our query is for our search input. So, say Q of search input just like that.

OK, so this code block right here is going to execute the actual request over to the Pixabay API.

Now I will assign this request to a variable called response.

Any time we make a request over the network, it takes some amount of time to complete that request.

So fetch method returns a promise which allows us to get a little notification when the request is complete. So we can handle that promise by using async await syntax.

It is a special syntax to work with promises in a more comfortable fashion. It’s easy to understand and use.

The word “async” before a function means one simple thing: a function always returns a promise. Other values are wrapped in a resolved promise automatically. In this case we will put async keyword in front of handle new term method.

So, async ensures that the function returns a promise, and wraps non-promises in it. Simple enough, right? But not only that. There’s another keyword, await, that works only inside async functions.

The keyword await makes JavaScript wait until that promise settles and returns its result.

Await literally suspends the function execution until the promise settles, and then resumes it with the promise result. That doesn’t cost any CPU resources, because the JavaScript engine can do other jobs in the meantime: execute other scripts, handle events.

So in this case we will put await key word before fetch method just like that.

Now we need to turn this request into a JSON object and get the data. To do so, I will define variable called data and it is going to be equal to response dot json method and we call it

Then for right now maybe we just console log out the response like so.

OK, so I'm going to save this file and we'll fly back over to the browser, and we'll give this a shot, I'm going to do a refresh of the page and then I'll do a search for, say, sky and in the console we got some results.

So, let's take a quick pause.

We'll come back to the next video.

Investigating Video Responses

So as soon as I enter that sky you'll see a collection of console logs over here.

So, remember, right now, every single time a user enters some text right here, we immediately trigger an event.

And that event instantly triggers an HTTP request over to the Pixabay API.

So, these are all results of a search on the Pixabay API.

This seems like overkill right now to do this many searches, but we'll address that in just a moment.

For now, let's investigate one of these responses and see what type of data that we get back from the Pixabay API.

So inside of here, you'll see a couple of different properties.

The data that came back these three properties

You can then look at items under hits property which will list all the different Pixabay videos that were found from our search request.

Inside of one of these videos, you'll find stuff like an comments, id likes and so on.

There are the information about that we get about this particular video.

So, this case, the tags are stars, long exposure, starry sky, which is definitely a video about sky, I'm sure.

OK, so this looks pretty good, it looks like we have some information about this list of videos coming back from the Pixabay API.

So, let's take a quick pause right here.

And when we come back, we'll talk about how we can use this list of videos to somehow display them inside of our application.

So quick pause and I'll catch you in just a minute.

Rendering a List of Videos

In the last section, we were able to execute a search on the Pixabay API for some given search term.

We then got back a list of objects that described all the different videos that were found during that search.

So, we now need to take this list of videos right here and somehow display them on the screen.

Remember, if we go back to our original mockup, we had said that at the very top we've got that search input component.

And then to render that list of videos bottom here, we would create a second component called videos.

The videos would be responsible for maintaining the overall list of videos, but we were also going to make a separate component called Item to represent one individual video.

Now, one thing I want to expand on just a little bit is that we are kind of breaking up this list into two separate components.

Just so you see a very good example of how we can easily take advantage of Vue to nest components in each other.

But if this was a real project, we could have just as easily decided to render the entire list using just one individual component.

Really, it's kind of an engineering decision.

If you have individual items inside the list that have a lot of complexity or a lot of events tied to them, well, then that might be a signal that you would want to break out a second component inside there rather than trying to stuff all that logic directly into the component that renders the overall list.

So, with that in mind, in this section, we're going to make this videos component and then wire it up to our app component, will then take a quick break and then figure out how we can communicate the list of videos down into the videos component.

So back inside my code editor, I'm going to create a new file inside of my components directory, and I will call it videos dot Vue

Remember, we always make one separate Vue file for every component that we create.

Inside of here we'll write out our template tag, our script tag and our style tag, and you'll notice that we haven't really been using our style tags that much so far, but we are going to come back through and do a pass on styling inside of our application in just a little bit and that's going to make sure that our application looks nicer.

Inside of here, I think that we want to render a list of videos and to render that list, we probably need some appropriate element inside the template that deals with lists appropriately.

To render the list of elements, I think that will put a UL inside of here. For right now, we'll just put the text videos inside there, but we'll come back in a little bit and make sure that we get that list of items or list of videos correctly rendered inside of here.

Then inside of the script tag, we'll add in our typical boilerplate, which is export default, and then an object that has a name, property of videos.

Before we go any further, let's make sure that we wire this component up to our app component just to get it visible on the screen.

So I'm going to go back into our app component and then I'll go through that three step process that we have to go through every time that we want to show one component inside of another. We will first add in an import statement. So, I will import videos from components videos.

Inside of my template itself, I'll add in a tag for the videos component and then step number three is to add it to the components property on the component configuration itself.

So right after search input, I'll put a comma and then videos like so.

OK, so let's save this file.

Well, then go back over to our browser and we see the text videos appear.

OK, so I think that the component is showing up on the screen.

Let's now continue in the next section and we'll figure out how we can communicate our list of videos that was just retrieved from the Pixabay API down to that component and then render them out as a list.

Updating Data Causes Rerenders

Our handle new term method is now making a request to the Pixabay API to fetch a list of videos, and then we've also got that videos component set up to take a list of videos and display them somehow on the screen.

The next thing we have to do is to figure out some how to communicate the list of videos that we fetch down to that videos component.

So, we have to communicate from the parent down to the child component.

In this case, the videos component is the child because it is being displayed from within the app component.

So, let's look at a diagram that's going to give us a better idea of how we're going to take that list of videos that came back in the Pixabay API response and communicate it down to the videos.

All right.

This is a timeline of sorts where we're going to start off at the very top and end up at the bottom.

So, in our flow for getting a list of videos to appear on the screen, everything starts with that handle new term method being called.

When we call that method, we're going to make a request to the Pixabay API.

The Pixabay API will then respond with that list of videos or give me that list of objects that represents individual videos.

We looked at that list just a moment ago inside the console of our browser, and we saw that we had an array of objects and each of those objects represented one video.

They had properties like the videos ID.

They had the like and download count, tags and some other properties as well.

So now the next step is where things start to get interesting.

Remember that any time that we have some type of variable, or some type of information tied to a Vue instance or a component, we store that information on that data property.

So, we use methods to update our data.

Whenever we update our data, the component automatically renders and then we can optionally make use of that computed property to somehow twist those values into something that can be displayed on the screen.

So, we're going to take that list of videos that are returned and we're going to assign it to a property on our data object inside of the app component.

Because we are updating the data property, we are going to cause the app component to automatically rerender its template.

Now, the important thing here to realize is that when we show one component inside of another, as we are right here with the videos component, whenever the app component renders its template with some new data that causes all the child components to be rendered as well.

So, whenever we update data in the app component, videos will be updated at the same time and that all happens automatically.

So, we need to take that data property that contains the videos.

We're going to take that data and we're going to stuff it down into the videos component.

The videos will then receive that list of videos and then once they are inside there, we can figure out exactly how to get the videos to render them out on the screen.

So that's the big idea here. That's what we're going to do.

We're going to take the list of videos. We're going to store it on data. When we do that causes everything to render and then we can get our list of videos inside a videos component and render them out inside of that things template.

So, with that in mind, let's continue with the next section where we're going to take our list of videos that are being retrieved inside of our handle new Term and make them make sure that we are storing them on our data object.

So quick break.

We'll take care of that in the next video.

Updating Data

In the last section, we spoke about how we're going to take our list of videos and store them on the data property of our app component. When we store the list of videos on that data property, it will cause the app component to rerender and automatically rerender the videos at the same time. So that's how we're going to communicate the list of videos down to the videos component.

So inside of my app component, right between our list of components and our methods, will add a new property that will return our object that represents our data.

So, we'll start off by defining the data property.

This is going to be a function that's going to return an object.

Now, remember, we can use a little bit of ES6 syntax to simplify this, to be simply data and then a set of parentheses. That inside of here we will return an object. In this object right here is going to initialize our data.

This is going to be like the starting state where the initial ingredients for our component.

So, we have to start to think about what different properties our data is going to contain.

Well, we know that we need to have a list of videos because that's what we are retrieving from our handle new term function right here.

So, I think that I will add a property to my data object simply called videos and because this is supposed to be an array of objects where every object represents one video, I'm going to initialize this to be an empty array like so.

Now, down inside of handle new term method, where to wait for a response from the Pixabay API, we can replace the console log with some logic that will take the list of videos that are contained inside this response and update our data properties, specifically the videos property with that new list of videos.

So, to capture that list of videos and assign them to our videos, piece of data right here, I will replace the console log statement that we have.

I'm going will say this videos is data dot hit just like that.

So, again, remember this hits property right here?

Not at all related to our component instance.

This is the data property that is tied to our response object that comes back from Pixabay and then on that hits property is the list of videos that is contained within items right here.

OK, so that's pretty much it.

So now any time we make a request, well, then get back our response.

We take the list of videos that were found, and we update our data piece of videos right here inside of our component instance.

When this line of code right here is executed, that is going to cause our entire template inside this app component to automatically rerender.

So maybe as a quick test just to make sure that everything is working correctly, let's render all array videos that have been found on the Pixabay API on browser. To do so, I'm going to add a little space underneath our videos component and then remember to print out any value from our component, we can add in those double curly braces. Then inside of here we could put down a very limited amount of JavaScript code, kind of like just do a little quick calculation.

Usually, we would use a computed function to do this.

But because this is a very temporary little thing that we're going to print out, we won't bother with making a computed function.

So inside of here, we can say videos dot valueOf just like that. So this is going to be the videos that are tied to the data property and then we're going to retrieve the array and print it out.

OK, so let's save this and we'll go back over to our browser and you'll notice that now we've got that zero empty array appearing on the screen.

But if I type in, say, car, then I whole array printed out right here.

OK, so that's great.

Not only are we taking a list of videos and assigning it to the data property on our component, but we can very easily see that every time we update that data property, it causes our component template to automatically rerender and update the content that we see on the screen.

So now we've got the list of videos stored on data.

We can move on to the next step here, which is to make sure that we take that list of videos on the data and pass it down to that videos component where they will then be rendered out as a list.

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

Communicating from Parent to Child

In the last section, we updated our data property of videos whenever we got a response back from the Pixabay API, because we updated a property that was defined on our data object that caused our app component to automatically render.

That's why we saw this videos dot valueOf thing right here, automatically change inside the browser any time we search for a list of videos.

So, remember, any time you want to update your component or cause it to render, you want to update a data property.

Now that we've got this in place, we want to somehow communicate this list of videos down to the videos component, because the videos component is responsible for rendering out information about each particular video on the screen.

If we want to communicate from a parent down to the child that we make use of the props system, and you'll recall that props is short for properties.

So, let's talk a little bit about how we communicate properties from the app down to some child component.

So, this is going to be a two-step process.

The first thing we're going to do is add a little bit of code to the parent component. Inside of the parent components template, we are going to add another directive to the child tag.

So as a quick example of that, here is our app component. It is the parent to the videos component. Here's the videos tag right here.

So, we are going to add a directive to this tag.

That directive is called the V Bind Directive, and we'll talk about what v bind means in just a moment.

Right now, all we really need to understand is that the parent components template needs to be updated. To make sure that this data flows down as a prop to the child.

Then inside the child, we're going to add a little bit of configuration to tell it that it should expect to receive some property from its parent.

So, in other words, communicating data from a parent down to a child requires some configuration not only in the parent, but inside the child as well.

Both parties must be aware that they need to exchange some amount of information.

So, let's now begin by opening our parent component and we'll take care of step number one right here, which is to add that v bind directive to the child component’s template tag.

OK, so here's my app component, it's the parent in this case. I'm going to first begin by deleting that videos dot valueOf tag that we had just a moment ago, because we don't really need to display array of videos that have been returned. Then find the tag that represents the child component that I'm trying to communicate some information to. It is right here.

So, to tag, we are going to add a V bind directive.

To do so, we will write out V Dash Bind a colon, then we will write out the name of the property that we want this to show up as inside the child.

I'm going to use the name videos. Because we are providing this name right here are videos.

That means that inside of the videos component we are going to have access to a property called videos.

If we had called this something else, like if we had this video list, then inside the videos component we would have access to a property called video list instead.

But in this case, it makes sense to simply call it videos.

Well, then place an equal sign, a pair of double quotes, and then inside the double quotes, we're going to write out the name of the data property that we want to share from the app component.

So, the app component wants to share its videos property right here down to that child.

So, we will write out videos just like that.

OK, so I know that seeing the word videos here twice is a little bit confusing, so just remember, the left-hand side says this is the name of the property that we want to have shown up inside the child and this is the name of the property that we want to share as it exists inside the parent.

Now, you might be a bit curious about what the purpose of this v bind thing right here is at all. V bind means that any time the videos property is updated inside of the parent, it should automatically try to render the videos component and provide that new list of videos down to the videos.

So, you can kind of think of v bind right here as meaning like, any time videos is updated, try to rerender the videos and provide it the new list of videos.

That's the purpose of v bind.

It kind of joins these two components together and ties them or binds them together, so to speak.

Now one other quick thing I'm going to throw in here.

Now, there is a shorthand form for defining a v bind directive as well.

This is another case yet again where there's more than one way to do things with Vue. So, we can shorten this directive right here from instead being the dash blind to instead simply saying colon videos.

So, the two are identical. The v dash blind means the same thing as simply colon videos.

So, this is another instance where you will want to be very consistent throughout your application and always either use this shorthand form of just colon and then the property name or everywhere use the dash find. I will keep use v dash bind.

OK, so we've now set up our parent component to tell it that any time its videos piece of data which is defined right here is updated, it needs to pass that list of videos down to the videos.

So, let's now continue in the next section where we're going to add some configuration to our child component to tell it about the props that it should expect to receive from its parent.

So quick break and we'll take care of step number two in the next section.

Prop Validation

In the last section, we added some configuration to our app component to take our list of videos and pass it down to the videos component.

We're now going to move on to step two of passing props, which is to add some configuration to our child component, to tell a bit about what data it should expect to receive from its parent.

So, I'm now going to open up my videos dot Vue file inside of our JavaScript section right here, we're going to add a new property called Props.

To tell this component about what data it should expect to receive from its parent, we will place an array and inside this array we will place a number of strings.

These strings are going to be the exact property names of the data that it should expect to receive from the parent.

So back inside of the app, we had said that we want to pass in our list of videos down to the videos component and inside the videos component.

We want that property to be called videos. So back inside videos component.

When I list out all the property names that we should expect to see. I will put videos right here.

So, there's a distinct tie between this name right here and how we called it inside of the parent component right here.

So, on both sides, we'll call it videos.

Now, defining our prop list right here can be as simple as listing out the names of all the different properties that we expect to receive.

But we can also add in some amount of configuration or validation to make sure that the child component right here is receiving the correct type of property as well.

So, we might want to validate to make sure that the parent is passing down, say, an array or to validate to make sure it's passing down a number or a string or an object.

So, if we want to and this is optional, the Vue documentation recommends you do this if you can but this is an optional step.

We can instead of providing an array of strings right here, provide an object where the keys are the names of the properties that we expect to receive, and then the values are the type of the property that we expect.

So, in the case of our videos property right here, we expect it to be an array of objects or more specifically, an array. So, we can say array.

This tells this component that it's going to expect to receive a property called videos and that property better be an array and if it's not an array, then this component is going to throw an error.

So again, you can either specify an object to do a little bit of simple type checking right here if you want to, or you can simply list out the names of the properties that you receive.

It's up to you in your project.

Again, I want to mention that The Vue documentation always recommends that you add and validation if you can, but it is your decision.

OK, so we've now taken care of step number two for passing props for from a parent down to a child.

So now inside of our child component, we can get take access to that prop that has been provided and make use of it anywhere inside of our component. So, we can create some computed functions, or we can create some methods, or we can directly access it inside of our template as well.

Let's do that last one for right now just to make sure that this data is showing up.

So, inside my template, I'm going to try to, again, print out the number of videos that we have been passed from the parent.

To do so, I'll put down those double curly braces again, which is how we do string interpolation inside of our template to print out some simple value.

And then inside of here, I can refer to the prop that has been passed down by writing out simply videos.

So, I should say it is identical to what we did previously when we accessed some data property inside of our template. Back inside of app, here is my app template right here.

We got access to the data property videos right here by simply writing out curly brace, curly brace videos.

So, this access to our data property that was defined inside of our component, we do the exact same thing when we are accessing a prop as well.

So, the very common theme here is any time we want to access any property, any data, or any computed function from within a template, we always write out just that things name. We don't have to write out this. We don't have to call it. We don't have to do anything like that. We just write out the name.

So, in our case, we specifically want to print out the array of videos that have been passed down as props. So, we'll do videos valueOf.All right.

So, let's now flip over to our browser and make sure this thing is working. I'll go back to my application. You'll see that we now see a videos and then a empty array after it.

But if I search for, say, car, I very quickly see videos and array of videos. Great.

So, it looks like we have successfully shared some data from the app component down to the Child videos component.

So now we've got that data showing up inside of here as props, we can now start to add some logic to the videos to render those out as an actual list.

So, let's start taking care of that in the next section.

Separate List Components

Our videos component now knows about the list of videos that it needs to somehow render to the screen. As a quick reminder, remember that the entire list itself is going to be comprised of two separate components.

The first is the videos, which is what we are working on right now.

The purpose of the videos component is really very simple and straightforward. It's going to take that list of videos that we passed down to it, and then it's going to try to render one item component for each video that it received.

As a quick reminder, we did not have to strictly divide this out into two separate components. I just wanted to give you a very good example of how you would render out kind of a multi component list. Using this technique of having two separate components to render the list is most useful when you have list items or like individual rendered items inside of here that are rather complicated or complex in nature, because it allows you to more simply isolate code that is related to rendering a list from code that is related to simply rendering a single item.

So, it's a little bit of arbitrary complexity that we threw in here, and we could have used one single component to do both steps. Both render the list and each individual item, but this is the path we're going to go down. So, let's continue.

I'm back inside of my videos component right here.

We can't really do a whole lot inside a videos component until we create the item component as well.

Once we create that, then we can use the video item to actually put the list together inside a videos.

So, let's first get started by creating a new file for the item component.

Inside my components directory, I'll make a new file called Item dot Vue. Inside of here we will put together some of our usual boilerplate.

So, we'll say template, script and then style.

Inside the script tag, we will immediately place some of the boilerplate that we need whenever we are defining a component.

So, I'm going to place inside of here export default and then our lists are curly braces, and I will provide a name for this component, which is going to be ‘item’ like so.

Then inside of the template, we're going to put down just a little bit of stand in HTML for right now.

i think that since we have a ul being displayed inside a videos, we will make each individual list item L-I (li).

So inside of here, I'll say li and we'll say List Item right now.

So, we've now created the item component, let's make sure that we import this into the videos component and wire it up as a component that the videos component can display.

So, here's my videos, I will go down to the script tag, I'm going to add an import statement, so I'm going to import item from item like so.

And then remember, any time that we want to tell this parent component that it has access to this child, one, we must add the components property to the parent component itself.

So inside of the videos configuration object right here in between name and props, I'm going to add in components. That's going to be an object and then I'll add in the name of the component that this thing might need to display, which is item.

So now inside of our videos component, we have access to the videos item component. So, all we really need to do now is figure out how we can create one item component for every video that is contained within our videos prop right here.

So, to build out that actual list, we'll take care of it in the next video. We're going to start looking at a directive that you're going to be using many times inside of your projects any time you want to build a list.

So quick break and we'll take care of that in just a moment.

Lists with V-For

In the last section, we created the item component and we wired it up to its parent of videos component.

We now need to add some code to the videos to make sure that it shows exactly one instance of the item for every video that we have fetched.

So far, it looks like every single time we get a response back from the Pixabay API, we are always getting a list of 20 videos.

So, in total, I would expect to see 20 item components.

To render out this list, we are going to use another directive inside of our videos component.

So inside of the videos template right here, I'm going to delete some of the stand in text we have right now, and then we are going to replace it with an instance of item component. So, let's say videos item.

Now, as I've written this component out right now, this would only create exactly one instance or one copy of the item component. But that's not the behavior that we want.

We want to have one instance of this component right here created for every single video that has been loaded up.

So, to tell our component that that is what we want to have happen here, we are going to use a directive called the V for Directive. So, let's write out the directive itself and we'll talk about how it works.

Inside of item I'm going to write the dash for then it equals and then video in videos just like that.

So, again, we refer to this as the V for directive and it's a directive that you're going to use every single time inside of the vue application that you want to build a list of components out.

We can interpret this directive right here as meaning for every single object or for every single video inside of the list of videos that was provided as a prop to our videos component, make exactly one item component.

So, this statement right here is going to take this item component right here and duplicate it down 20 times because we have 20 separate videos usually inside of this array.

Notice how we also declared this word right here video. So, we said video in videos, because we have video right here that means that inside of this component tag, we can access an individual video and print out some information about it. We'll see an example of that in just a second but for right now, let's save this and go back over to the browser and see what happens.

Now, as a quick reminder, our item component at present, this is a item component is simply printing out an li of videos item.

So, if this V directive works appropriately, then I would expect to see the term videos item printed out 20 times on the screen.

As you we see, we have an error right here. It says ‘Custom elements in iteration require 'v-bind key' directives’.

So, you need the v-bind:key in a v-for because it needs to differentiate each component rendered, in case of data changing. You need to use the both, v-bind:key and v-bind:value in your component option, to it work properly and this key value must be unique in iteration.

ID is the property has unique value. So, we can use id property as key. I am saying v bind key equals video dot id. That’s it.

Now, I'm going to go back over to my application inside the browser and you'll see that we have 20 items listed out here. I can, of course, refresh the application.

When I refer to the application, you'll notice that that list goes away because when we first load our application up, we have not yet attempted to retrieve any videos whatsoever from the Pixabay API.

So, at present we have zero videos and that's why we see no items listed out here.

But if we do some search that retrieves 20 videos from the Pixabay API and now we can see 20 instances of the item component appear on the screen.

OK, so that is step one of getting a list of components rendered onto the screen.

Let's continue in the next section and we're going to figure out how we can customize the item component to show some information about the particular video that is supposed to represent.

Handling Props with V-For

In the last section, we were able to get a collection of item components to print out on the screen by using the V for directive. The only issue now is that it appears that every item is printing out the hard coded text of simply video item.

So, we need to figure out some way to say that for every video in this list of videos, we want to communicate this video thing right here down to this component so that this component instance knows what video it is supposed to be displaying information about on the screen.

So, in other words, this first list item right here or this first videos item needs to communicate information about the first video that was returned from Pixabay API and then the second one needs to communicate information about these second video and so on down the list.

So, this is another instance where we want to communicate some data from the parent component of videos down to the child component of item.

So, we are going to, again, make use of the props system in Vue to communicate some information from the parent down to the child, a item.

So, we just went through this process but as a quick reminder, to pass some props for the parent down to the child, we first have to add the bind expression to the parent components template.

Now, this time around, that might seem like it's a little bit more challenging here. It's not the same type of setup that we had back inside of our app component right here.

And, how do we get access to, like one individual video to pass down to the item component?

Well, when we write out v-for and then video in videos, this declares a temporary variable that we can use inside of our template of video.

Video right here will represent one video or one object in our array.

So, this represents one video that has been retrieved from Pixabay.

So, I'm going to give myself a little bit of space inside this tag here, because we're going to add in a pretty good amount of configuration to it.

And then underneath the V for Directive, we're going to set up our V bind an expression, by writing out the dash spined colon and then the name of the prop as we want it to show up inside of the child component.

So, the name of the prop that we want to have shown up inside an item component will make simply video just to stay consistent.

So now item component is going to have access to a variable called video.

Then we need to put in our equals in our double quotes and then inside the double quotes, we will specify the piece of data or the variable of sorts from this videos component that needs to be passed down to the child.

So, this is where we're going to get access to this video variable right here.

And again, you can kind of think of this as being like a temporary variable that is available only inside of our template right here. So, we're going to passing video.

At this point, notice that we have absolutely no property tied to our component called video. It does not exist inside of here. There's nothing simply called video.

So, this value of video right here is referencing the current video that we are iterating over inside of our list of videos.

So, this thing right here is basically being shoved to right here.

If we had instead called this thing right here, like ‘item’, we would also have to update this name right here because they are the exact same thing, these two things, the exact same thing.

But again, I want to stick with simply video to have some consistent terminology.

OK, so now that we are passing from some information as a prop from the parent down to the child. We need to make sure that we tell the child component about the props that it should expect to receive as well.

So I'm going to open up my videos item component again and I'll add in our props property.

Remember that this can be either an array of strings that has the names of the properties that this thing should expect to receive, or we can make it an object if we want to do some validation on the prop when it comes in but again, totally up to you.

So, we expect to receive a property from the parent simply called video.

Now, to make sure that this thing is actually working, we're going to go into our item component template will delete the hardcoded text, a video item, and we will replace it with a reference to this video prop that just got passed down to us from the parent.

We will try to print out the tags of the video that we just retrieved. To get the tags of the video, we'll put in our curly braces like so to indicate we want to do some string interpolation and then the tags of the video will be available on video dot tags. So, you have seen this property already when we looked at the request response that came back from Pixabay inside of our browser console.

The video object right here represents a reference to the property that we were passed down into this child.

That video object had that tags property and that tags property had the tags for the video that we just retrieved.

And so, in this case, we are trying to print out just the tags right here.

OK, so this is looking good, let's save this and then we'll test our application side the browser and we'll make sure that we can see the tags up here.

All right.

So back over here.

Well, you'll see right away that, yep, we've got a list of video tags appearing on the screen inside of our list.

And if we wanted to, we could delete the search term and then we could instead search for something else like, say, pen, and then we get a list of video tags appearing related with pen.

So, this is looking pretty good.

Let's continue in the next section.

And we're going to talk about one last thing around this v-for directive that we put together inside the videos component.

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