Hello! I’m sorry for my English, but I try to learn. My name is Alexander. I would like to talk about Vue. I heard the phrase many times, Vue is cool. Vue is easy to learn, like JS is a joke. Actually it is. I did not have much time to get acquainted with this framework. But in the near future I will have to carry out a project on this framework.

>

And so,

Vue is a progressive framework for creating user interfaces. Vue can solve problems of various levels of complexity. From creating one button on the page to a full-fledged SPA application.

Because Vue is:

Approachable >

Versatile >

Performant >

>

There are currently two versions of Vue 2 legacy . And the current Vue 3. In this video, I would like to talk about the latest version of Vue 3 and its composition API syntax.

There are several different syntaxes found in Vue right now:>

* **Options API >**
* **Composition API >**
* **Class API >**
* **Class API + [vue-property-decorator](https://www.npmjs.com/package/vue-property-decorator) (npm)>**

I will try to consider the composition API, in comparison with the widely used syntax of the Option API.

The new Composition API creates many convenient ways to reuse code in components. Compared to the Option API, the Option API divided the logic into options: data, methods, computed,…

With the Composition API, we are not limited to this structure and can separate code by feature rather than by option.

The main advantages of Composition API:

* Reactivity API are ref and reactive entities.

• Replaces mixins and all related issues such as naming conflicts.

• Improved type support, since it uses mostly regular functions and variables, TypeScript doesn't require complex typing. In the old version, class components and decorators had to be used to implement TypeScript support. This is not always convenient and generally complicates the development process.

With the advent of the Composition API, everything has changed. The new syntax allows you to use typing with minimal effort.

• Smaller files - The Composition API requires less code.

• More performance - the template is compiled into a render function in the same scope, without an intermediate proxy, less work for the language server to extract types from the code

>

V

V

Let's look at an example. You can see, how much less code has become. At the same time, the functionality was completely preserved. This is just not a great example, can you imagine how much the codebase will decrease on a large project?

>

Let's look at the performance in comparison with well-known libraries and frameworks. As you can see in the screenshots, VUE wins in some tests of React and Angular. And where VUE could not win, it loses by a minimal margin.

>

Let's get back to the Composition API, which has several features. Namely:

* **Refs**
* **Reactive**
* **Teleports**
* **Hooks**

V

Ref - Refs are used to add reactivity to primitive values. By default, variables are not created reactive. In order to add reactive behavior to them, we must use ref. Ref - Gets an internal value and returns a reactive and mutable ref object. This ref object has only one .value property that points to the internal value.

V

Reactive - we have ref's but what about objects? After all, ref's can only be applied to primitive variables. But don't worry. The Vue developers have taken care of this. We have a reactive() method. Which allows you to convert an object to a reactive one. That is, all fields of this object become reactive, including nested ones!

V

Teleports - Sometimes we want to create a component that will be embedded in a specific location in a document. An example of this is modal windows. We want the modal window logic to be inside the component, and the modal window itself to be embedded in the DOM elsewhere, such as in the BODY tag. Teleports will help us here.

>

Hooks

I can't say for sure, but it seems that this feature appeared with the advent of the composition API. Hello react. Hooks allow you to reuse the same logic in different components. Like React, hooks are functions that can be a cleaner and more flexible way to define and share logic between components. Hooks can also return a state.