Vue – Composition API

Peter Kassenaar –
info@kassenaar.com

Vue 3 – Composition API

An alternative to the default Vue Option Based API

Vue Components – the standard way

- "Option based API", b/c components are written as objects with key/value pairs, the *options*, defined in Vue documentation
- Code is organized by option and not by functionality
- Data, methods, computed properties are spread all over the component

```
<script>
   export default {
      name: "CounterStandard",
      data() {
         return {
            …
         }
      },
      props: {
         …
      },
      created() {
         …
      },
      methods: {
         …
      }
   }
</script>
```

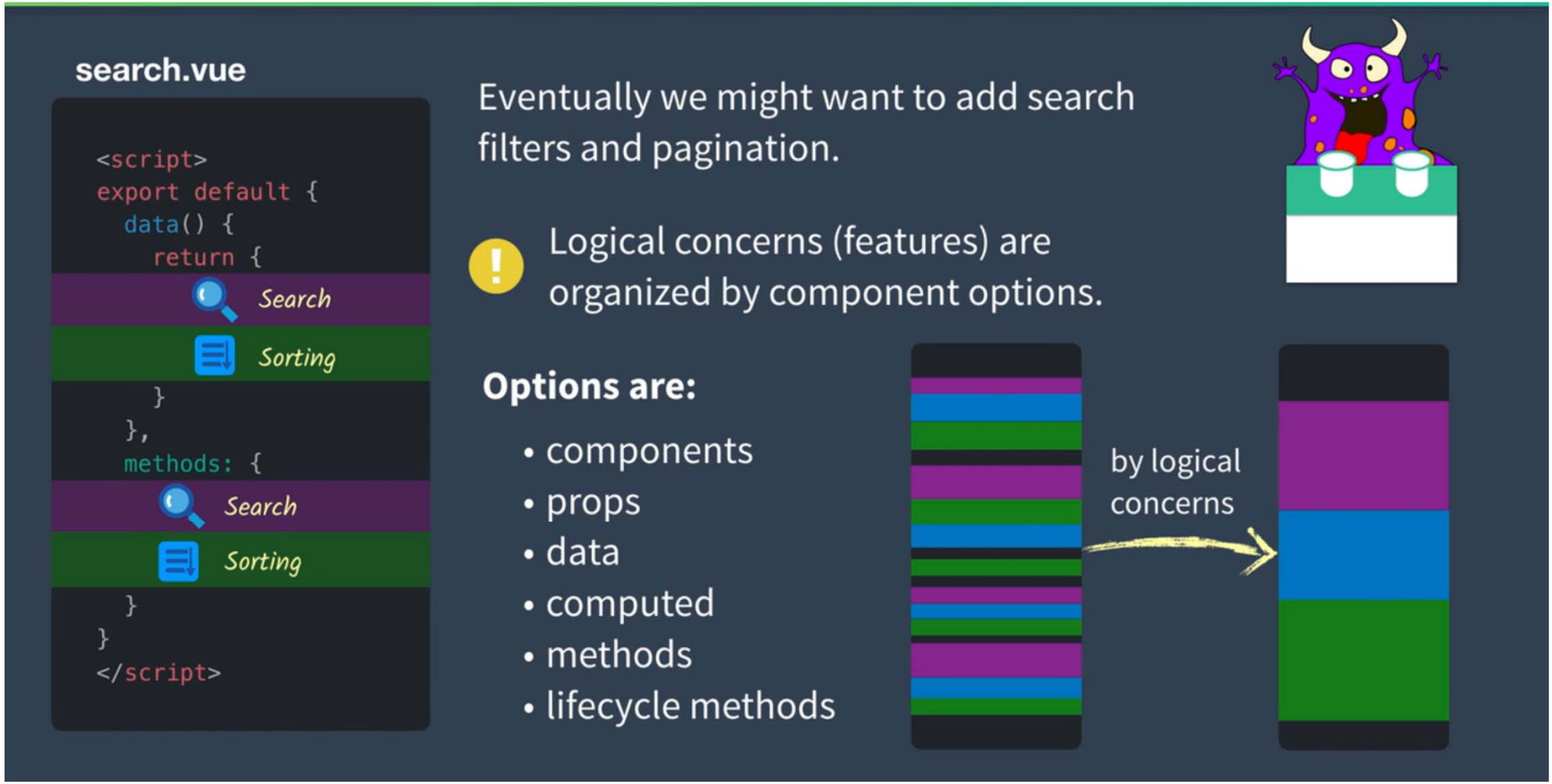
Limitations of regular components

1. As components grow, they become bigger, uglier, less readable and less maintainable

2. It is more difficult to reuse components or pieces of code

3. Limited support for TypeScript in Vue 2

1. Organization by option, not by feature



Credits: https://www.vuemastery.com/courses/vue-3-essentials/why-the-composition-api/

2. Reuse of code

- Vue 2: different ways of reusing code already exist:
- Mixins
  - Pro: code is now organized by feature
  - Con: possible naming conflicts, unclear relationships
- Creating a mixin factory – a function that returns a customized version of a mixin
  - Pro: easily reusable and configurable, better relationships
  - Con: proper namespacing requires discipline, no instance access at runtime
- Scoped slots
  - Pro: solves the limitations of mixins
  - Con: Increases indentation, reduces readability, configuration in template instead of code, more components to maintain, less performant at runtime

Vue 3 – Composition API

- We compose the functionality of a component out of plain JavaScript functions
- Pro's
  - Less code
  - Familiar: you already know and use functions
  - Extremely flexible
  - Tooling friendly (autocomplete, type support)
- Con:
  - There are now two ways of defining components – and they can be mixed and used at the same time.

Example – a counter component

Use a `setup()` method inside a component:

```
<script>
   import {ref} from '@vue/composition-api';

   export default {
      name: "CounterComposition",
      props: {
         …
      },
      setup(props) {
         // All local variables you need
         const …
         // return an object with the values you want to expose to the template
         return {
            counter,
            increment,
            decrement
         }
      }
   }
</script>
```

Installing Composition API

Setting up a Vue 2-project to use the Composition API

Installation in Vue 2 project

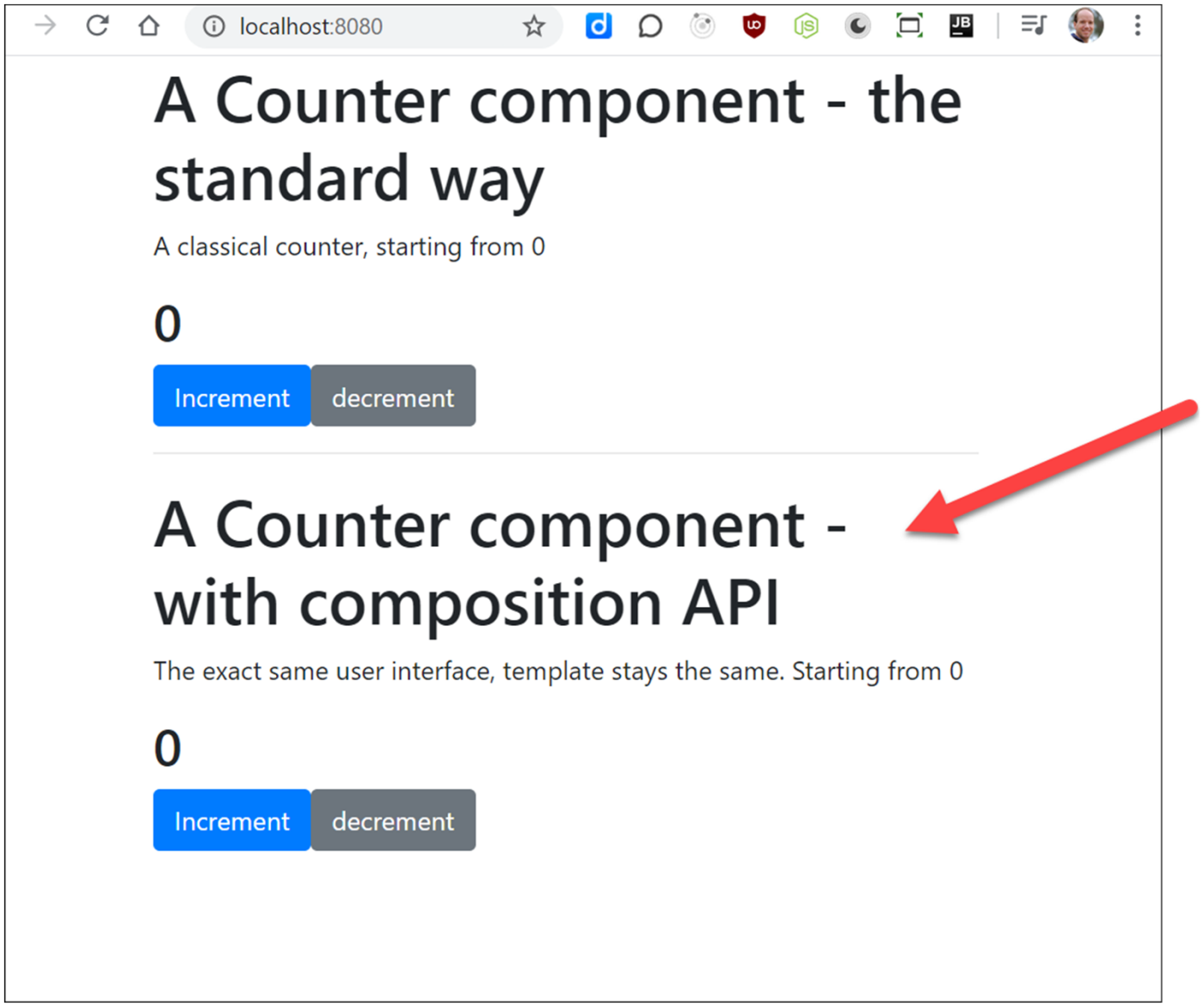
Vue 3? Composition API is available by default

Vue 2? Install the correct plugin:

```
npm install @vue/composition-api
```

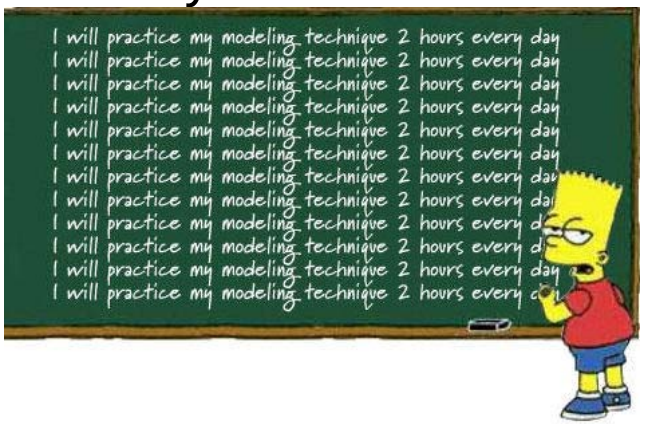
```
// main.js
// import Composition API
import VueCompositionApi from '@vue/composition-api';
Vue.use(VueCompositionApi);

new Vue({
  render: h => h(App),
}).$mount('#app')
```

Creating your first project



Workshop

- Create your own project and install the Vue Composition API (if necessary) in it
- OR: work from `../examples/550-composition-api-intro`
- Create a textbox in a standard component.
  - Text that is typed into the box, is shown in the UI after pressing Enter
  - Very simple, use `v-model` for that.
- Create the same functionality in a Composition API component
- When working from the sample code: add the functionality to the `CounterComposition.vue` component
- Update the component, so the text in the box can be passed in as a prop

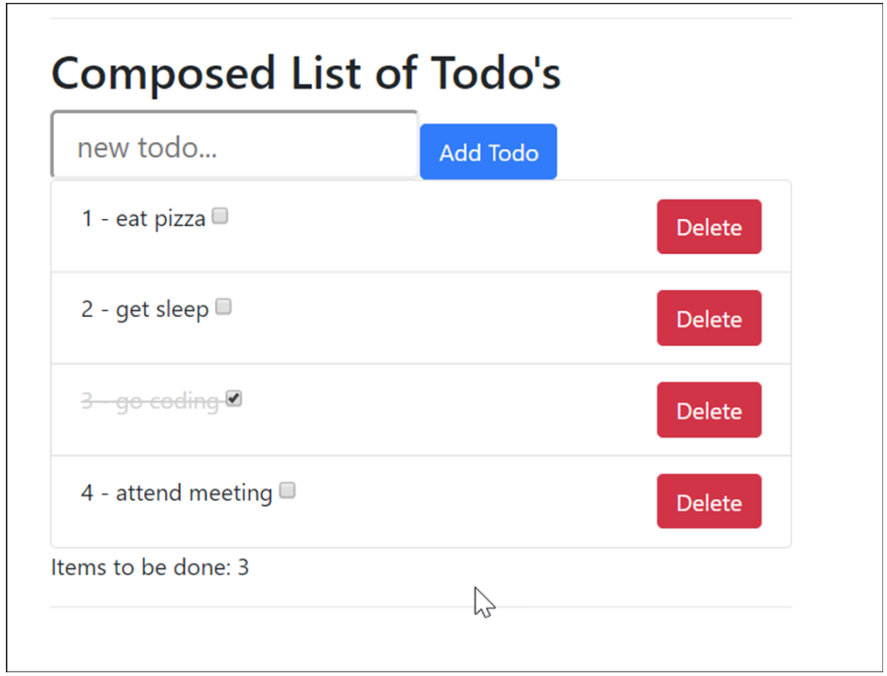


On using complex objects

Working with objects as state

Example

- Creating a simple TODO application
  - It has some data (arrays w/ objects w/ multiple properties)
  - It has some methods
  - It has some computed properties
  - You can expand the functionality to saving the data to localStorage, to a backend and so on.



On using `ref()` and `reactive()`

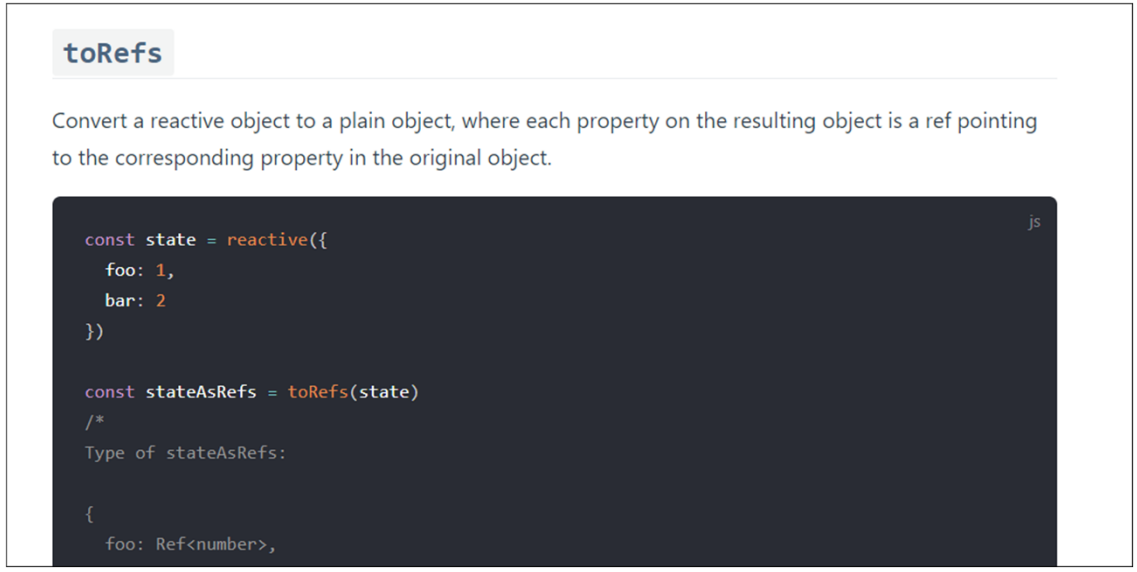
- Most of the time, you'll need your data to be reactive. We have 2 methods for that:
- `ref()` takes the value and turns it into a reactive object
  - Updating it? update `object.value`, not directly the `object`
- `reactive()` also creates a reactive object, but turns every property of an object into a reactive value.
- What to use?
  - Documentation is unclear on this.
  - In general: use `ref()` on simple/primitive values, use `reactive()` on arrays and objects.
  - https://composition-api.vuejs.org/#ref-vs-reactive

Optional / advanced – using toRef()

- You cannot destructure a reactive object into reactive properties.
  - `return {…reactiveObject}` will loose reactivity.
- You need the `toRefs()` method for that

```
import { toRefs } from '@vue/composition-api';

return {...toRefs(obj)}; // will destructure to seperate ref objects
```

- https://composition-api.vuejs.org/api.html#torefs



toRefs

Convert a reactive object to a plain object, where each property on the resulting object is a ref pointing to the corresponding property in the original object.

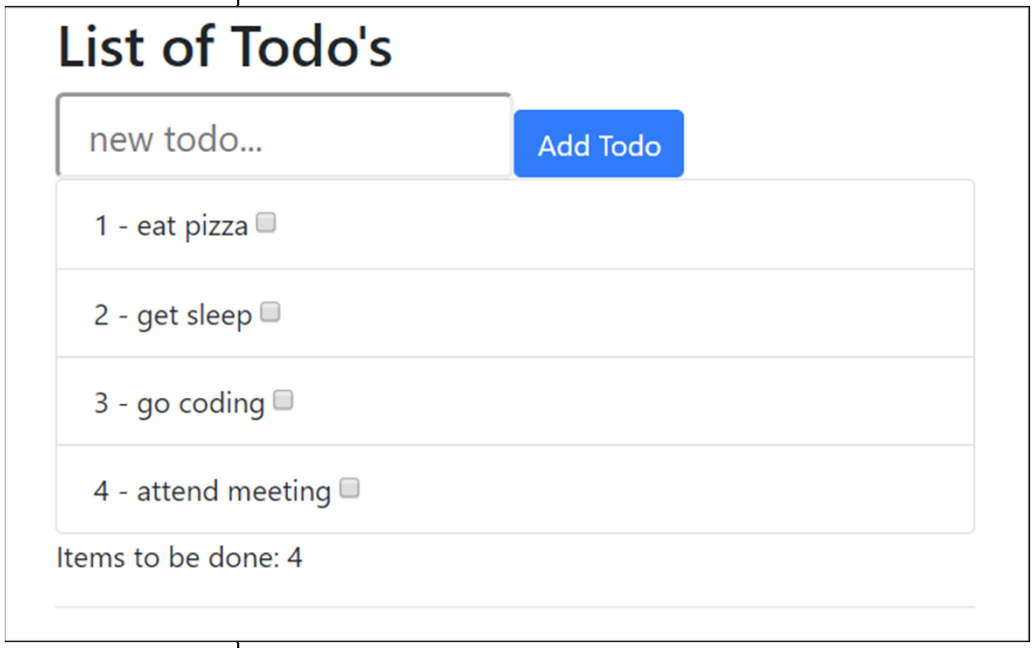
```js
const state = reactive({
  foo: 1,
  bar: 2
})

const stateAsRefs = toRefs(state)
/*
Type of stateAsRefs:

{
  foo: Ref<number>,
```

Creating TodoList.vue

```
<template>
  <div>
    <h2>List of Todo's</h2>
    <input type="text"
        placeholder="new todo...“ class="form-control-lg“
        v-model="newTodo" @keyup.enter="addTodo()" >
    <button class="btn btn-primary"
        @click="addTodo()">
      Add Todo
    </button>
    <ul class="list-group">
      <li class="list-group-item"
        v-for="(todo, index) in todos" :key="index"
        @click="toggleTodo(todo)“ >
        <span :class="{'done' : todo.done}">
          {{ todo.id}} - {{ todo.name}}
        </span>
        <input type="checkbox"
          v-model="todo.done">
      </li>
    </ul>
    <p v-if="remainingItems">Items to be done: {{ remainingItems }}</p>
    <p v-if="!remainingItems">No more items, you're done!</p>
  </div>
</template>
```

Standard HTML,

no surprises here



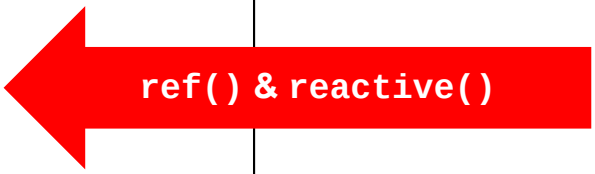
Todo-functionality

Just one method that holds all functionality. It is called `setup()`

```
import {computed, reactive, ref} from '@vue/composition-api'

export default {
    name: "TodoList",
    setup() {
   ...
 }

}
```

0. Initialization - adding state / properties

```
// 0. Initialize basic properties and state
const newTodo = ref('');
let state = reactive({
   todos: [
      {id: 1, name: 'eat pizza', done: false},
      {id: 2, name: 'get sleep', done: false},
      {id: 3, name: 'go coding', done: false},
      {id: 4, name: 'attend meeting', done: false},
   ]

});
```



We define our state as `ref()` and `reactive()` properties.

The `state` of course can be expanded with more key/value pairs.

1 & 2 - Adding functionality

```
// Functionality
// 1. Add a new todo
const addTodo = () => {
   // check if a text is typed in
   if (newTodo.value && newTodo.value.trim()) {
      state.todos.push({
         id: state.todos.length + 1, // dummy id, based on length
         name: newTodo.value, // ref-value from v-model
         done: false
      })
      newTodo.value = '';
   }
}

// 2. Toggle state of todo
const toggleTodo = todo => {
   todo.done = !todo.done;
}
```

Methods are just functions. Here they are written as ES6 arrow functions

Use `ref.value`

3. Using computed properties

```
// Computed properties
// 3. Calculate the remaining items
const remainingItems = computed(() => {
   return state.todos.filter(todo => !todo.done).length
})
```

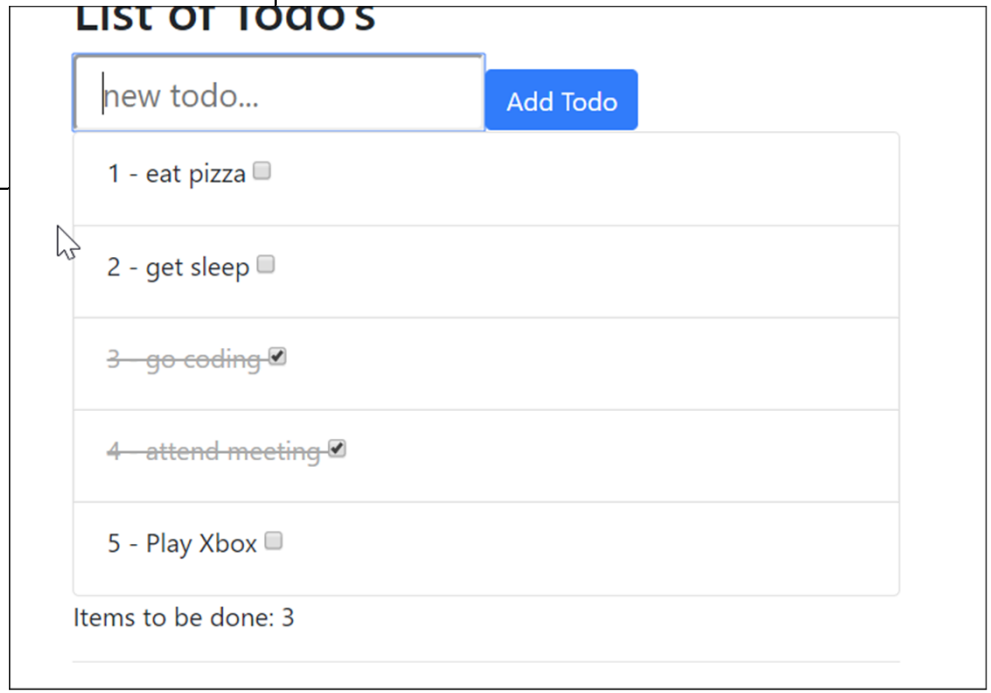
You can use computed properties as normal properties. Just wrap them in a `computed()` method, imported from `@vue/composition-api`.

Don't forget to import `computed()`!

4. Exporting / returning an object

```
// 4. Return object from the setup function to expose them to the UI
return {
   // properties
   newTodo,
   todos: state.todos,

   // computed properties
   remainingItems,

   // methods
   addTodo,
   toggleTodo
}
```

Remember to export the properties and methods in an object, otherwise they won't be exposed to the UI



You"ll now have a functional component, using the composition API.

…/components/TodoList.vue

But…what's the win here?

"Fine. We just moved all the different functionality to one `setup()` method. But it's still long, ugly and non-reusable. So what's the win here?"

We can now extract the functionality to their own JavaScript-files/functions, and compose them together again in this component!

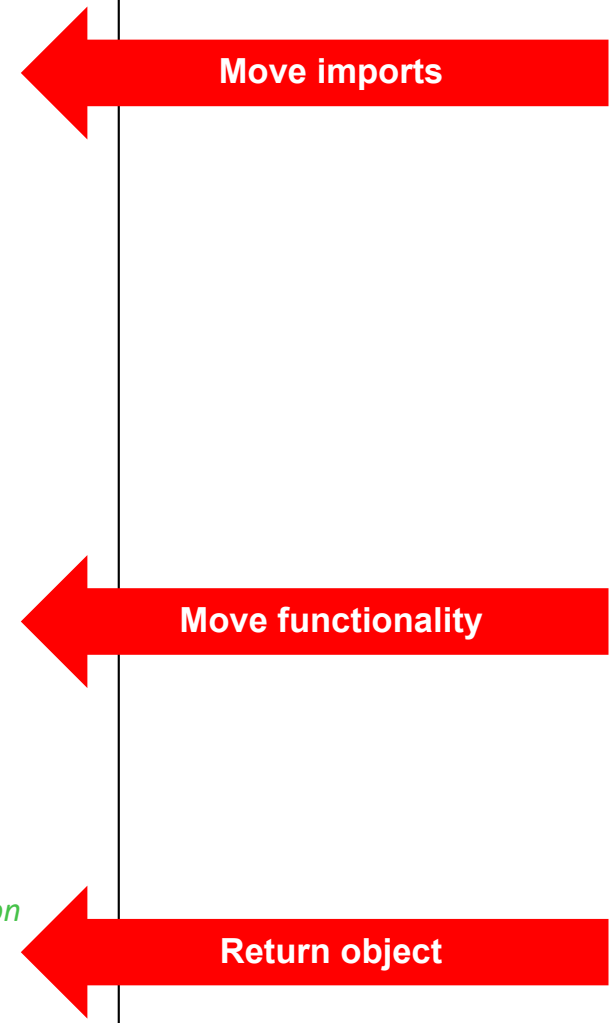
Composition API as ‘hooks’

- Maybe we want to use the list of Todo’s and their functionality elsewhere in the application
- We follow the analogy of React Hooks here
- Also using the `useSomeFunctionality.js` naming convention

- Move all functionality that has to do with `Todos` (duh…) to `useTodos.js`
- Expose an object from there and import it in the consuming component

The useTodos.js file

```
// ../hooks/useTodos.js

import {computed, reactive, ref} from "@vue/composition-api";

export default () => {
   // 0. initialization
   const newTodo = ref('');
   let state = reactive({
      todos: [...]
   });

   // Functionality
   // 1. Add a new todo
   const addTodo = () => {
      ...
   }

   // 2. Delete a todo

   // 3, computed properties
   ...

// return object, to be consumed by component that import this function
   return {
      newTodo,
      todos: state.todos,
      addTodo,
      ...
   }
}
```



Composing TodoListComposition.vue

```
import useTodos from "../hooks/useTodos";

export default {
  name: "TodoListComposition",
  setup() {
    // composing the API of this component together from various 'hooks',
    // using ES6 destructuring here
    const {
      newTodo,
      todos,
      addTodo,
      ...
    } = useTodos()


    // API of this component, composed out of the imported functions
    return {
      newTodo,
      todos,
      addTodo,
      ...
    }
  }
}
```

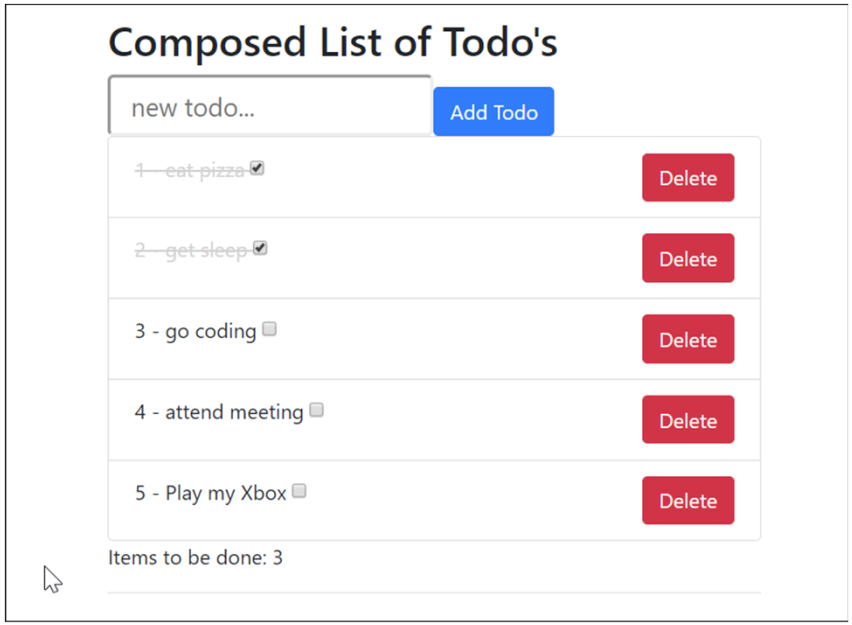
Imports hooks

Destructure into local variables

Expose to UI

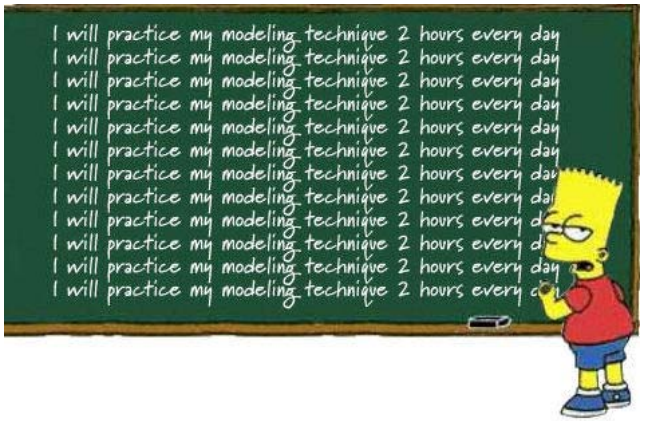
Result

- The end result is visually the same, but is much cleaner
- The resulting component is now composed out of smaller bits of code
- We can of course import/add more functionality from different hooks
- Hooks can be reused in the rest of the application
- Hooks are centered around functionality,
  not around objects/options



Workshop

- Work from `../examples/560-composition-api-todo`
- Create functionality to fetch countries from the restCountries API.
  - Create a textbox that you can type a (partial) country name in
  - Fetch countries using `https://restcountries.eu/rest/v2/name/{name}`.
- Functionality should be in its own file, and composed together with the rest of the functionality in `TodoListComposition.vue` (in real applications, you would probably use a separate component for that)



Checkpoint

- The composition API has many new methods:
  - `setup()` - resides on the component, runs after initial props resolution. Receives optional `props` and `context` as arguments
  - `ref()` - returns a reactive variable, triggers re-render of the template on change.
  - `reactive()` - returns a reactive object, triggers re-render of the template on reactive variable change.
  - `computed()` - returns a reactive variable based on the getter function argument, tracks reactive variable changes and re-evaluates on change
  - `watch()` – (not covered here) handles side-effects based on the provided function, tracks reactive variable changes and re-runs on change

https://composition-api.vuejs.org/

Vue Composition API

RFC API Reference Languages

Composition API RFC

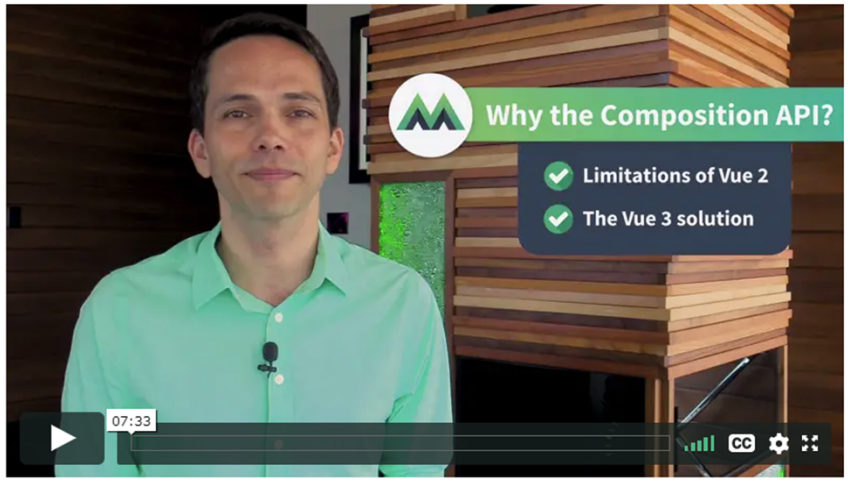
- Summary
- Basic example
- Motivation
  - Logic Reuse & Code Organization
  - Better Type Inference
- Detailed Design
  - API Introduction
  - Code Organization
  - Logic Extraction and Reuse
  - Usage Alongside Existing API
  - Plugin Development
- Drawbacks
  - Overhead of Introducing Refs
  - Ref vs. Reactive
  - Verbosity of the Return Statement
  - More Flexibility Requires More Discipline
- Adoption strategy
- Appendix
  - Type Issues with Class API
  - Comparison with React Hooks
  - Comparison with Svelte

Composition API RFC

- Start Date: 2019-07-10
- Target Major Version: 2.x / 3.x
- Reference Issues: #42
- Implementation PR: (leave this empty)

Summary

Introducing the **Composition API**: a set of additive, function-based APIs that allow flexible composition of component logic.



Watch Vue Mastery's Vue 3 Essentials Course. Download the Vue 3 Cheat Sheet.

More information

Vasily Kandinsky (1866–1944): Composition 8 (Komposition 8)

React developers will love the Vue Composition API

Steven Straatemans Follow
Nov 12, 2019 · 12 min read

Ok, let me be clear first. I'm not really big on Vuejs. I like React. And not just the way of writing Class components. No I like the functional components. Just functions, love it!

A colleague asked me, last October, to do a meetup talk with him about Vue and especially about it's Composition-API. It was a new API, still in the RFC

https://medium.com/frontmen/react-developers-will-love-the-vue-composition-api-91424d221419

DEV composition api

80

33

162

CUSS (15)

Build a movie search app using the Vue Composition API

Gábor Soós Oct 17 '19 *Updated on Oct 27, 2019* · 7 min read

#vue #javascript #webdev

https://dev.to/blacksonic/build-a-movie-search-app-using-the-vue-composition-api-5218

Comparing React Hooks with Vue Composition API

Guillermo Peralta Scura Sep 6 '19 · 15 min read

#react #vue #javascript #frontend

Vue recently presented the Composition API RFC, a new API for writing Vue components inspired by React Hooks but with some interesting differences that I will discuss in this post. This RFC started with a previous version called Function-based Component API that received lots of criticism from certain part of the community, based on the fear of Vue starting to be more complicated and less like the simple library that people liked in the first place.

The Vue core team addressed the confusion around the first RFC and this new one presented some interesting adjustments and provided further insights on the motivations behind the proposed changes. If you are interested in giving some feedback to the Vue core team about the new

https://dev.to/voluntadpear/comparing-react-hooks-with-vue-composition-api-4b32