THE COMPOSITION API

EMERGING PATTERNS & BEST PRACTICES

THE COMPOSITION API

HI, I'M THORSTEN

HI, I’M THORSTEN

PREREQUISITES

🙂 I’VE SEEN GREGG’S TALK

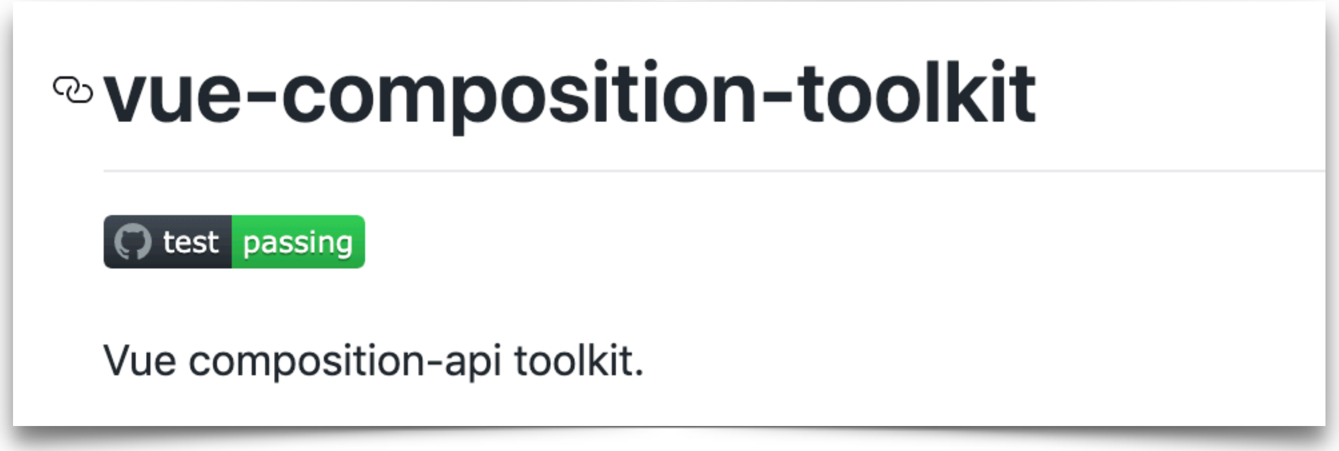
😁 I’VE PLAYED WITH THE COMPOSITION API

EMERGING PATTERNS &BEST PRACTICES

useWeb

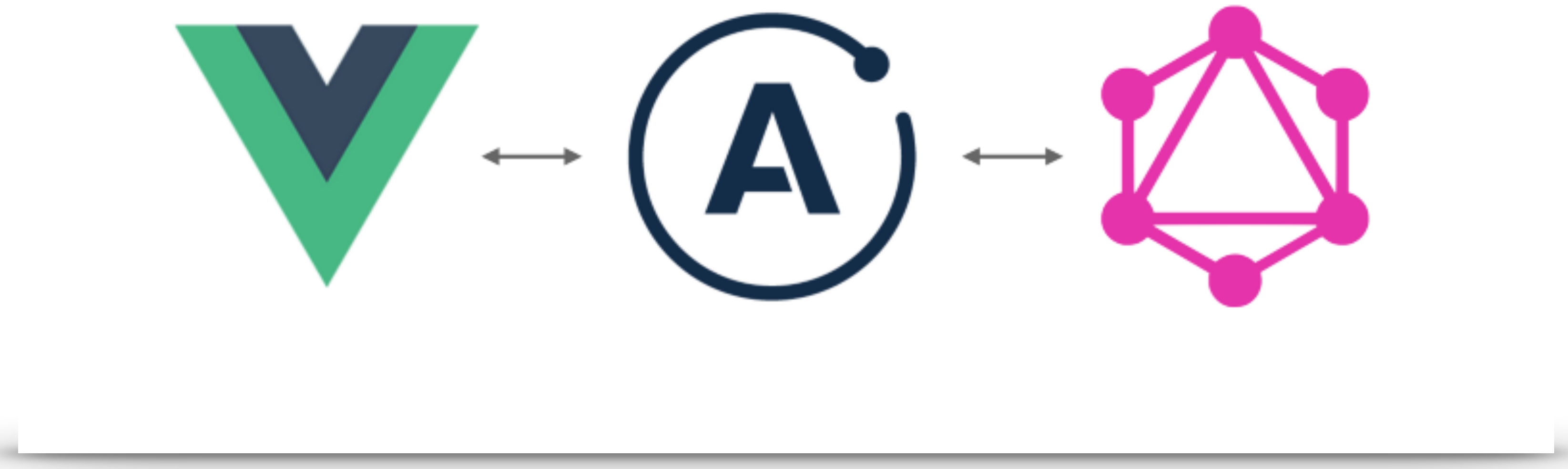
Web APIs implemented as Vue.js composition functions

vuex-feathers



vue-composition-toolkit

Vue composition-api toolkit.

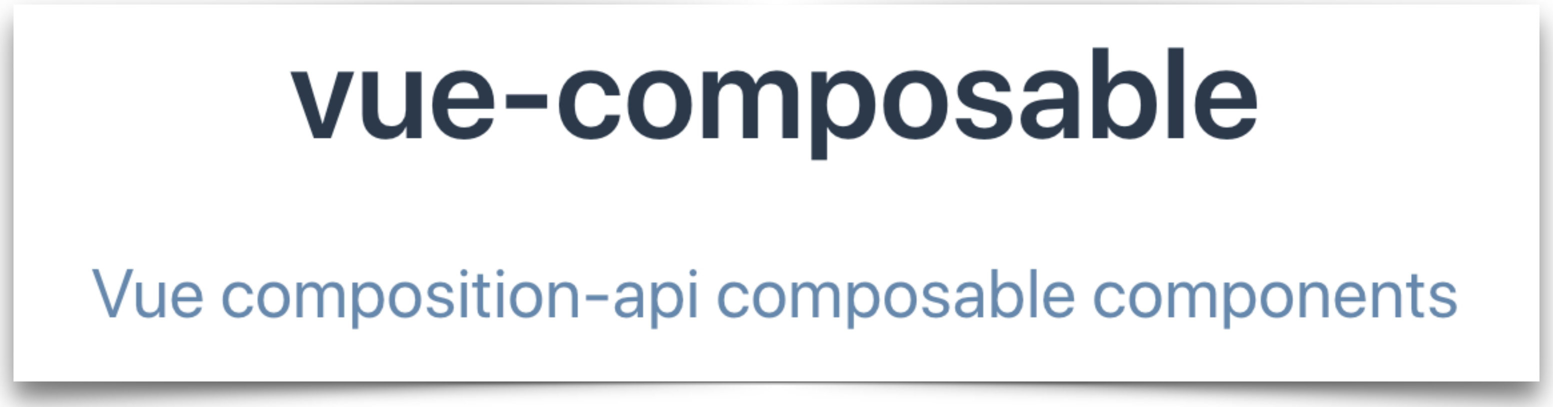


vue-apollo@next

VueUse

villus (tiny GraphQL client)

Vuelidate@next



vue-composable

Vue composition-api composable components

awesomejs.dev

Ten (?) Best Practices
State of Adoption
The Tips →
Structuring Your 'setup' Functions
What's in a name?
All About The Refs
Typps & Tricks
#__: Comp. Fn's Start with 'use...'
#__: Pass props as refs
#__: For template refs, use watch() over lifecycle hooks
#__: Extract helpers early
#__: Name 'use*' only when using lifecycle state
#__: Assume arguments to be refs
#__: Use boolean refs to make event listeners toggleable
#__: extract non-reusable 'stuff', too!
#__: Return an object, except when single result
#__: return readonly for Controlled State
Live Coding?
What's next?
#Whoami
The End
Tips are covering:
Style
Flexible APIs
Robustness
Prevent Footguns
#__: Use eslint plugin to prevent footguns
#__: Use Typescript
Release Info abt alpha/betas
Call to action: "& playful"

SO I STARTED READING A LOT OF CODE

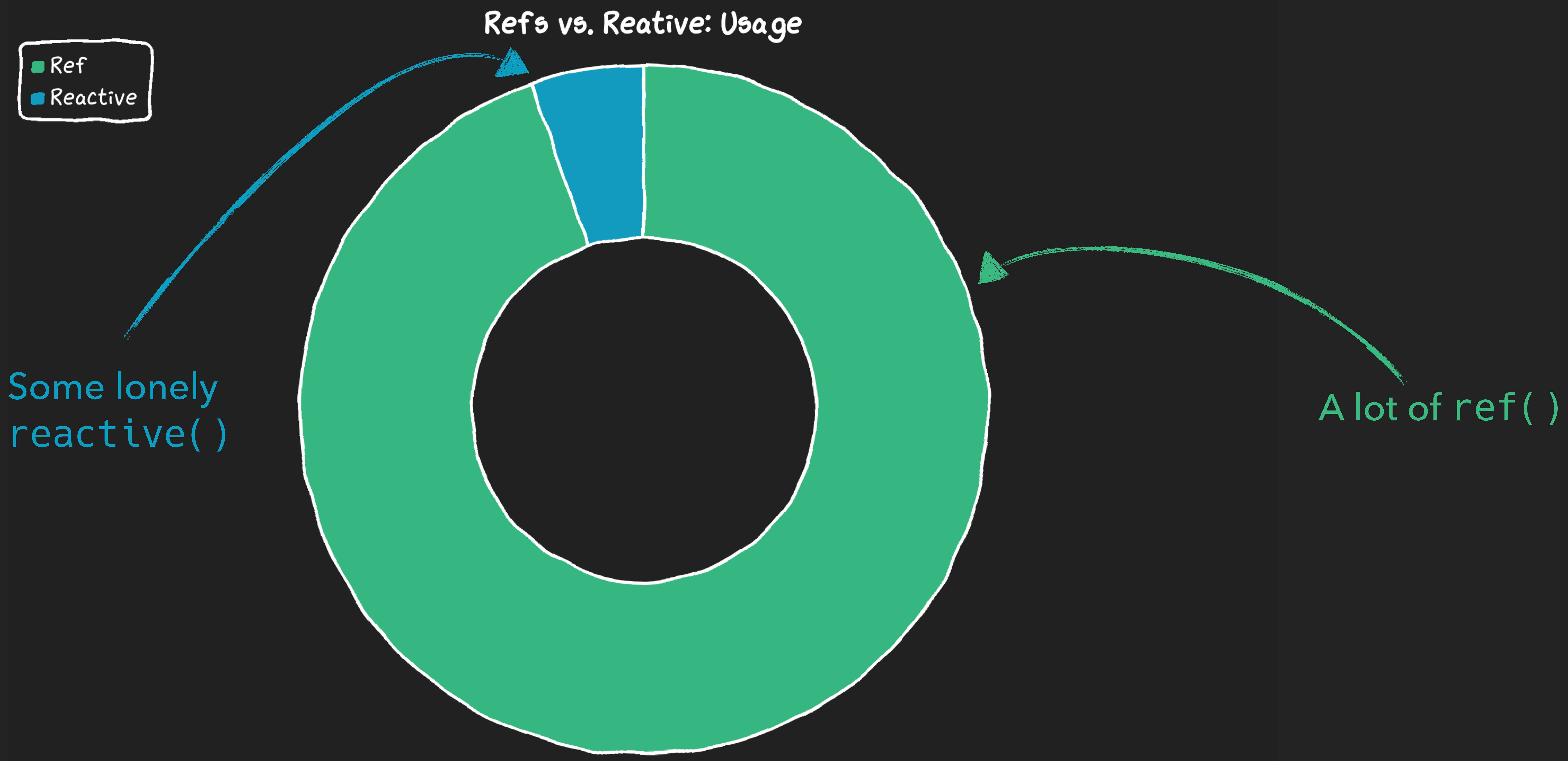
… LIKE… A LOT!

… AND COLLECTED IMPRESSIONS ON POST-IT NOTES

REF() VS. REACTIVE()...

WHICH SHOULD I USE?

50% of comments in the RFC



Some lonely reactive( )

A lot of ref( )

https://timqian.com/chart.xkcd

DISCLAIMER:

MOST OF THIS CODE IS LIBRARY CODE

Practices for use in components may come out different

CONSISTENCY

SOMETIMES, REACTIVE() DOESN'T WORK*

and you rather want/need a ref

* at least not in an ergonomic way

DEVELOPERS VALUE CONSISTENCY

COMPUTED PROPERTIES ARE REFS

```
export function exampleWithComputed() {
  const state = reactive({
    a: 1,
    b: 2,
    x: 2,
  })

  const sum = computed(() => state.a + state.b)

  const squared = computed(() => sum.value ** state.x)

  return toRefs({
    ...state,
    sum,
    squared,
  })
}
```

```
export function exampleWithRefs() {
  const a = ref(1)
  const b = ref(2)
  const x = ref(2)

  const sum = computed(() => a.value + b.value)

  const squared = computed(() => sum.value ** x.value)

  return toRefs({
    a,
    b,
    x,
    sum,
    squared,
  })
}
```

CONSISTENCY

DOM REFERENCES REQUIRE (TEMPLATE) REFS

```
setup() {
  const inputEl = ref<HTMLInputElement>(null)

  onMounted(() => {
    inputEl.value.addEventListener(/* */)
  })

  return {
    inputEl,
  }
}
```

```
<template>
  <div>
    <input type="text" ref="inputEl" />
  </div>
</template>
```

IF DEVELOPERS VALUE CONSISTENCY

they likely prefer refs, as those work everywhere

SO... ARE YOU TELLING ME THAT `REACTIVE()` IS USELESS?

most of your, probably

OF COURSE NOT!

If you want to use it, do!

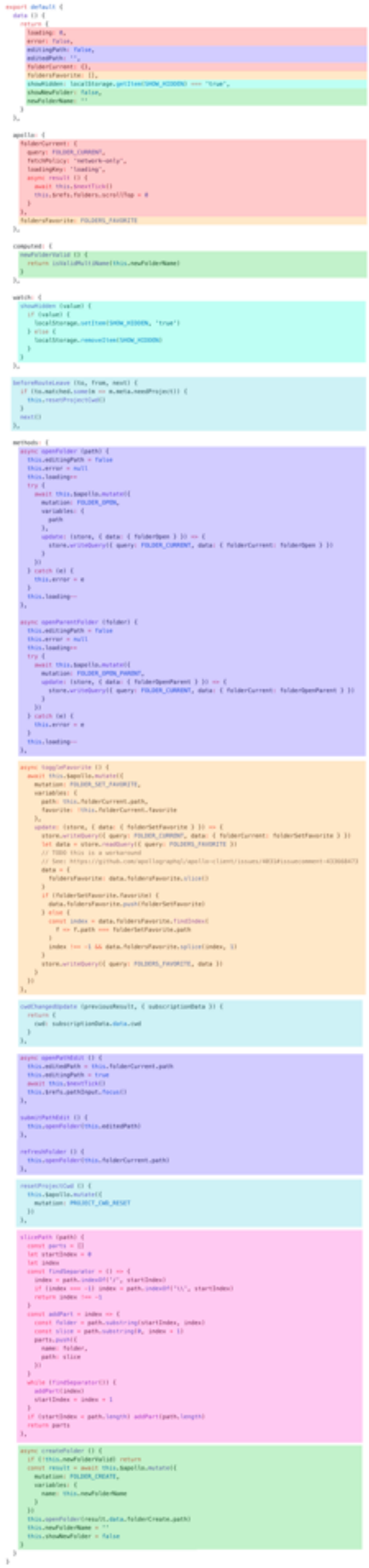
IT'S A QUESTION OF PERSONAL PREFERENCE

Just accept that you can't completely evade refs

EMERGING BEST PRACTICES

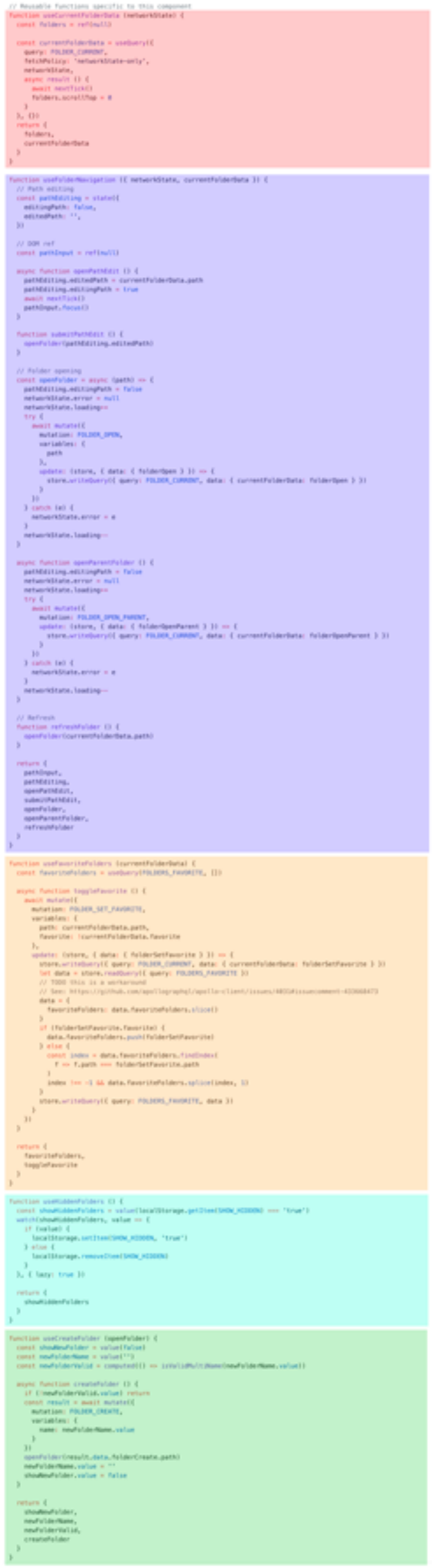
by example

Options API

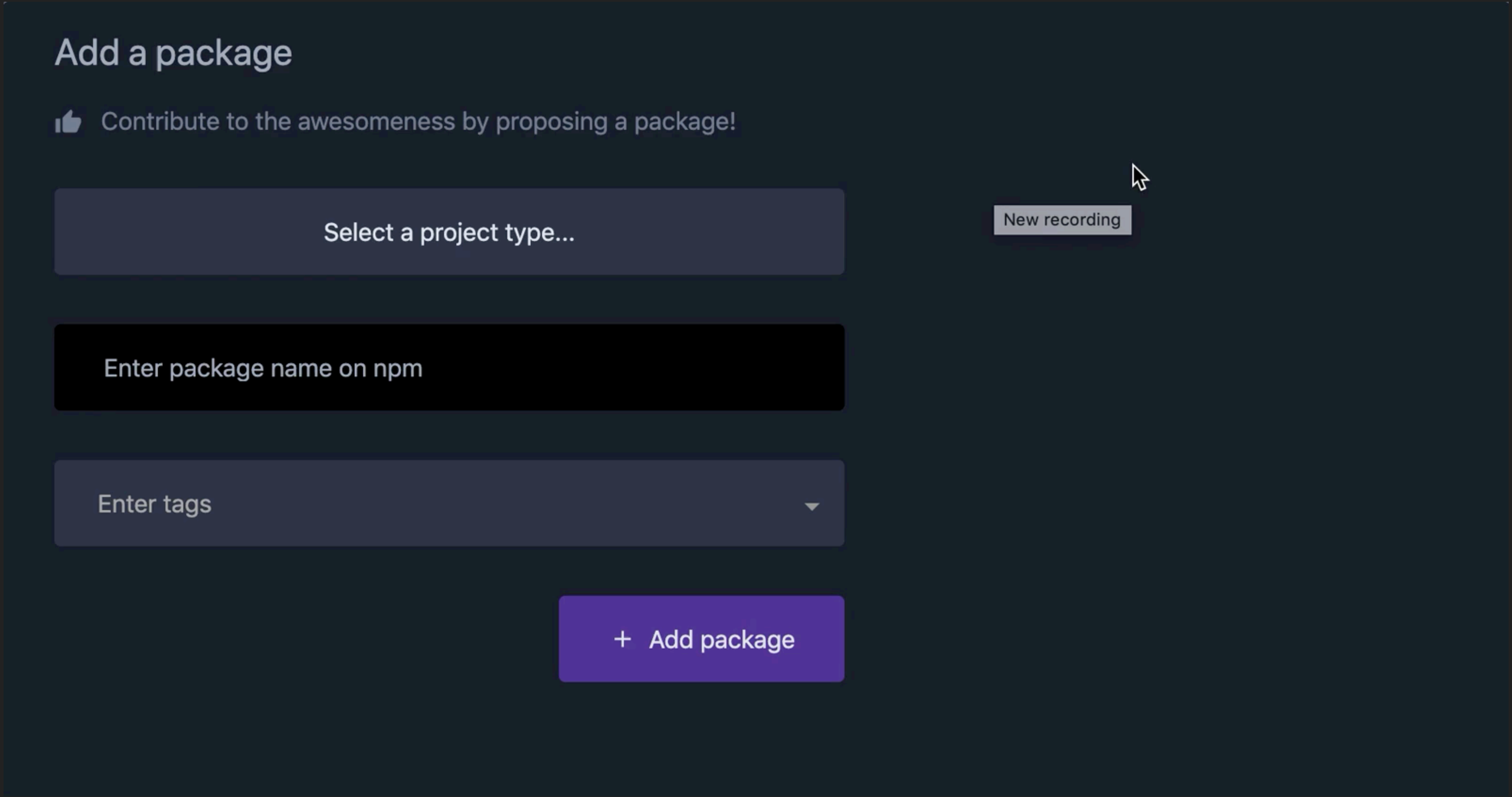


```
export default {
  data () {
    return {
      loading: 0,
      error: false,
      editingPath: false,
      editedPath: '',
      folderCurrent: {},
      foldersFavorite: [],
      showHidden: localStorage.getItem(SHOW_HIDDEN) === 'true',
      showNewFolder: false,
      newFolderName: ''
    }
  },

  apollo: {
    folderCurrent: {
      query: FOLDER_CURRENT,
      fetchPolicy: 'network-only',
      loadingKey: 'loading',
      async result () {
        await this.$nextTick()
        this.$refs.folders.scrollTop = 0
      }
    },
    foldersFavorite: FOLDERS_FAVORITE
  },

  computed: {
    newFolderValid () {
      return isValidMultiName(this.newFolderName)
    }
  },

  watch: {
    showHidden (value) {
      if (value) {
        localStorage.setItem(SHOW_HIDDEN, 'true')
      } else {
        localStorage.removeItem(SHOW_HIDDEN)
      }
    }
  },

  beforeRouteLeave (to, from, next) {
    if (to.matched.some(m => m.meta.needProject)) {
      this.resetProjectCwd()
    }
    next()
  },

  methods: {
    async openFolder (path) {
      this.editingPath = false
      this.error = null
      this.loading++
      try {
        await this.$apollo.mutate({
          mutation: FOLDER_OPEN,
          variables: {
            path
          },
          update: (store, { data: { folderOpen } }) => {
            store.writeQuery({ query: FOLDER_CURRENT, data: { folderCurrent: folderOpen } })
          }
        })
      } catch (e) {
        this.error = e
      }
      this.loading--
    },

    async openParentFolder (folder) {
      this.editingPath = false
      this.error = null
      this.loading++
      try {
        await this.$apollo.mutate({
          mutation: FOLDER_OPEN_PARENT,
          update: (store, { data: { folderOpenParent } }) => {
            store.writeQuery({ query: FOLDER_CURRENT, data: { folderCurrent: folderOpenParent } })
          }
        })
      } catch (e) {
        this.error = e
      }
      this.loading--
    },

    async toggleFavorite () {
      await this.$apollo.mutate({
        mutation: FOLDER_SET_FAVORITE,
        variables: {
          path: this.folderCurrent.path,
          favorite: !this.folderCurrent.favorite
        },
        update: (store, { data: { folderSetFavorite } }) => {
          store.writeQuery({ query: FOLDER_CURRENT, data: { folderCurrent: folderSetFavorite } })
          let data = store.readQuery({ query: FOLDERS_FAVORITE })
          // TODO this is a workaround
          // See: https://github.com/apollographql/apollo-client/issues/4031#issuecomment-433668473
          data = {
            foldersFavorite: data.foldersFavorite.slice()
          }
          if (folderSetFavorite.favorite) {
            data.foldersFavorite.push(folderSetFavorite)
          } else {
            const index = data.foldersFavorite.findIndex(
              f => f.path === folderSetFavorite.path
            )
            index !== -1 && data.foldersFavorite.splice(index, 1)
          }
          store.writeQuery({ query: FOLDERS_FAVORITE, data })
        }
      })
    },

    cwdChangedUpdate (previousResult, { subscriptionData }) {
      return {
        cwd: subscriptionData.data.cwd
      }
    },

    async openPathEdit () {
      this.editedPath = this.folderCurrent.path
      this.editingPath = true
      await this.$nextTick()
      this.$refs.pathInput.focus()
    },

    submitPathEdit () {
      this.openFolder(this.editedPath)
    },

    refreshFolder () {
      this.openFolder(this.folderCurrent.path)
    },

    resetProjectCwd () {
      this.$apollo.mutate({
        mutation: PROJECT_CWD_RESET
      })
    },

    slicePath (path) {
      const parts = []
      let startIndex = 0
      let index
      const findSeparator = () => {
        index = path.indexOf('/', startIndex)
        if (index === -1) index = path.indexOf('\\', startIndex)
        return index !== -1
      }
      const addPart = index => {
        const folder = path.substring(startIndex, index)
        const slice = path.substring(0, index + 1)
        parts.push({
          name: folder,
          path: slice
        })
      }
      while (findSeparator()) {
        addPart(index)
        startIndex = index + 1
      }
      if (startIndex < path.length) addPart(path.length)
      return parts
    },

    async createFolder () {
      if (!this.newFolderValid) return
      const result = await this.$apollo.mutate({
        mutation: FOLDER_CREATE,
        variables: {
          name: this.newFolderName
        }
      })
      this.openFolder(result.data.folderCreate.path)
      this.newFolderName = ''
      this.showNewFolder = false
    }
  }
}
```

Composition API



```
// Reusable functions specific to this component
function useCurrentFolderData (networkState) {
  const folders = ref(null)

  const currentFolderData = useQuery({
    query: FOLDER_CURRENT,
    fetchPolicy: 'networkState-only',
    networkState,
    async result () {
      await nextTick()
      folders.scrollTop = 0
    }
  }, {})
  return {
    folders,
    currentFolderData
  }
}

function useFolderNavigation ({ networkState, currentFolderData }) {
  // Path editing
  const pathEditing = state({
    editingPath: false,
    editedPath: '',
  })

  // DOM ref
  const pathInput = ref(null)

  async function openPathEdit () {
    pathEditing.editedPath = currentFolderData.path
    pathEditing.editingPath = true
    await nextTick()
    pathInput.focus()
  }

  function submitPathEdit () {
    openFolder(pathEditing.editedPath)
  }

  // Folder opening
  const openFolder = async (path) => {
    pathEditing.editingPath = false
    networkState.error = null
    networkState.loading++
    try {
      await mutate({
        mutation: FOLDER_OPEN,
        variables: {
          path
        },
        update: (store, { data: { folderOpen } }) => {
          store.writeQuery({ query: FOLDER_CURRENT, data: { currentFolderData: folderOpen } })
        }
      })
    } catch (e) {
      networkState.error = e
    }
    networkState.loading--
  }

  async function openParentFolder () {
    pathEditing.editingPath = false
    networkState.error = null
    networkState.loading++
    try {
      await mutate({
        mutation: FOLDER_OPEN_PARENT,
        update: (store, { data: { folderOpenParent } }) => {
          store.writeQuery({ query: FOLDER_CURRENT, data: { currentFolderData: folderOpenParent } })
        }
      })
    } catch (e) {
      networkState.error = e
    }
    networkState.loading--
  }

  // Refresh
  function refreshFolder () {
    openFolder(currentFolderData.path)
  }

  return {
    pathInput,
    pathEditing,
    openPathEdit,
    submitPathEdit,
    openFolder,
    openParentFolder,
    refreshFolder
  }
}

function useFavoriteFolders (currentFolderData) {
  const favoriteFolders = useQuery(FOLDERS_FAVORITE, [])

  async function toggleFavorite () {
    await mutate({
      mutation: FOLDER_SET_FAVORITE,
      variables: {
        path: currentFolderData.path,
        favorite: !currentFolderData.favorite
      },
      update: (store, { data: { folderSetFavorite } }) => {
        store.writeQuery({ query: FOLDER_CURRENT, data: { currentFolderData: folderSetFavorite } })
        let data = store.readQuery({ query: FOLDERS_FAVORITE })
        // TODO this is a workaround
        // See: https://github.com/apollographql/apollo-client/issues/4031#issuecomment-433668473
        data = {
          favoriteFolders: data.favoriteFolders.slice()
        }
        if (folderSetFavorite.favorite) {
          data.favoriteFolders.push(folderSetFavorite)
        } else {
          const index = data.favoriteFolders.findIndex(
            f => f.path === folderSetFavorite.path
          )
          index !== -1 && data.favoriteFolders.splice(index, 1)
        }
        store.writeQuery({ query: FOLDERS_FAVORITE, data })
      }
    })
  }

  return {
    favoriteFolders,
    toggleFavorite
  }
}

function useHiddenFolders () {
  const showHiddenFolders = value(localStorage.getItem(SHOW_HIDDEN) === 'true')
  watch(showHiddenFolders, value => {
    if (value) {
      localStorage.setItem(SHOW_HIDDEN, 'true')
    } else {
      localStorage.removeItem(SHOW_HIDDEN)
    }
  }, { lazy: true })

  return {
    showHiddenFolders
  }
}

function useCreateFolder (openFolder) {
  const showNewFolder = value(false)
  const newFolderName = value('')
  const newFolderValid = computed(() => isValidMultiName(newFolderName.value))

  async function createFolder () {
    if (!newFolderValid.value) return
    const result = await mutate({
      mutation: FOLDER_CREATE,
      variables: {
        name: newFolderName.value
      }
    })
    openFolder(result.data.folderCreate.path)
    newFolderName.value = ''
    showNewFolder.value = false
  }

  return {
    showNewFolder,
    newFolderName,
    newFolderValid,
    createFolder
  }
}
```



Add a package

Contribute to the awesomeness by proposing a package!

```
setup (props, { root }) {
    // Form data
    const projectTypeId = ref(root.$route.query.projectTypeId || null)
    const formData = reactive({
      packageName: root.$route.query.packageName || '',
      tags: [],
    })

    watch(() => root.$route, value => {
      projectTypeId.value = value.query.projectTypeId || null
      formData.packageName = value.query.packageName || ''
    })

    // Check for existing proposals & packages
    const { result, loading } = useQuery(gql`
      query PackageProposalAndPackageByName ($name: String!) {
        proposal: packageProposalByName (name: $name) {
          ...pkgProposal
          projectTypes {
            id
            name
            slug
          }
        }

        pkg: packageByName (name: $name) {
          ...pkg
          projectTypes {
            id
            name
            slug
          }
        }
      }
      ${pkgFragment}
      ${pkgProposalFragment}
    `, () => ({
      name: formData.packageName,
    }), () => ({
      enabled: !!formData.packageName,
      debounce: 1000,
    }))
    const proposal = useResult(result, null, data => data.proposal)
    const pkg = useResult(result, null, data => data.pkg)
    const alreadyProposed = computed(() => formData.packageName && !loading.value && proposal.value)
    const alreadyExists = computed(() => formData.packageName && !loading.value && pkg.value)

    // Form validation
    const requiredFieldsValid = computed(() => projectTypeId.value != null && !!formData.packageName)
    const valid = computed(() => requiredFieldsValid.value && !alreadyProposed.value && !alreadyExists.value)

    // Added summary
    const added = ref(false)
    const addedProposal = ref(null)

    // Submit

    const { mutate, error, loading: submitting, onDone } = useMutation(gql`
```

😱

```
function usePackageCheck(formData){}

function useFormValidation(projectTypeId, formData) {}

function useSubmit(valid, formData, projectTpeId, rerquiredFieldsValid) {}

setup(props, { root }) {

    // Initial state setup left out

   // Check for existing proposals & packages
    const {
      proposal,
      pkg,
      alreadyExists,
      alreadyProposed,
    } = usePackageCheck(formData);

    // Form validation
    const valid = useFormValidation(projectTypeId, formData);

    // Submit
    const { error, submitting, submit } = useSubmit(
      valid,
      formData,
      projectTypeId,
      requiredFieldsValid
    );

    // NPM search
    const { searchText: npmSearchText, result: npmSearchResult } = useNpmSearch(
      {
        hitsPerPage: 5
      }
```

```
function usePackageCheck(formData){}

function useFormValidation(projectTypeId, formData) {}

function useSubmit(valid, formData, projectTpeId, rerquiredFieldsValid) {}

setup(props, { root }) {

    // Initial state setup left out

   // Check for existing proposals & packages
    const {
      proposal,
      pkg,
      alreadyExists,
      alreadyProposed,
    } = usePackageCheck(formData);

    // Form validation
    const valid = useFormValidation(projectTypeId, formData);

    // Submit
    const { error, submitting, submit } = useSubmit(
      valid,
      formData,
      projectTypeId,
      requiredFieldsValid
    );


   // …
```

```
function usePackageCheck(formData){}

function useFormValidation(projectTypeId, formData) {}

function useSubmit(valid, formData, projectTpeId, rerquiredFieldsValid) {}

setup(props, { root }) {

    // Initial state setup left out

   // Check for existing proposals & packages
    const {
      proposal,
      pkg,
      alreadyExists,
      alreadyProposed,
    } = usePackageCheck(formData);

    // Form validation
    const valid = useFormValidation(projectTypeId, formData);

    // Submit
    const { error, submitting, submit } = useSubmit(
      valid,
      formData,
      projectTypeId,
      requiredFieldsValid
    );

   // …
```

```
function usePackageCheck(formData){}

function useFormValidation(projectTypeId, formData) {}

function useSubmit(valid, formData, projectTpeId, rerquiredFieldsValid) {}

setup(props, { root }) {

    // Initial state setup left out

   // Check for existing proposals & packages
    const {
      proposal,
      pkg,
      alreadyExists,
      alreadyProposed,
    } = usePackageCheck(formData);

    // Form validation
    const valid = useFormValidation(projectTypeId, formData);

    // Submit
    const { error, submitting, submit } = useSubmit(
      valid,
      formData,
      projectTypeId,
      requiredFieldsValid
    );

   // …
```

```
export function myCompositionFunction(arg1, arg2) {
```

Dealing with arguments

```
/* The magic happens here */
```

Implementation Tripwires

```
  return {
    refs,
    objects,
    functions,
  }
```

Returning the right way

```
}
```

HANDLING REFS IN ARGUMENTS

```
export function useWithRef(someFlag: Ref<boolean>) {

  watch(ref, val => {
    /* do something */
  })

  return {}
}
```

```
const isActive = ref(true)
const result = useWithRef(isActive)
```

😊

```
const result2 = useWithRef(true)
```

😱

TYPE ERROR!

```
export function useWithRef(someFlag: Ref<boolean>) {
  if (!isRef(someFlag)) warn('Needs a ref')

  watch(ref, val => {
    /* do something */
  })

  return {}
}
```

```
const isActive = ref(true)
const result = useWithRef(isActive)
```

😊

```
const result = useWithRef(ref(true))
```

🧐

CAN WE ACCEPT BOTH REF & STATIC VALUES?

```
export function useEvent(
  el: Ref<Element> | Element,
  name: string,
  listener: EventListener,
  options?: boolean | AddEventListenerOptions
) {
  const element = wrap(el as Element)

  onMounted(() => element.value!.addEventListener(name, listener, options))

  onUnmounted(() => element.value!.removeEventListener(name, listener))
}
```

```
const wrap = (value) => (isRef(value) ? value : ref(value))


export function useEvent(
  el: Ref<Element> | Element,
  name: string,
  listener: EventListener,
  options?: boolean | AddEventListenerOptions
) {
  const element = wrap(el as Element)

  onMounted(() => element.value!.addEventListener(name, listener, options))

  onUnmounted(() => element.value!.removeEventListener(name, listener))
}
```

FORGIVING API VS. STRICT API

LIFECYCLE HOOKS VS. WATCH

```
export function useEvent(_el, name, listener, options) {

  const element = wrap(_el)

  onMounted(() => element.value.addEventListener(name, listener, options))

  onUnmounted(() => element.value.removeEventListener(name, listener))
}
```

- What if the ref is empty on mount?
- What if the ref changes later?

WATCH() TO THE RESCUE

```
export function useEvent(_el, name, listener, options) {

  const element = wrap(_el)

  onMounted(() => element.value.addEventListener(name, listener, options))

  onUnmounted(() => element.value.removeEventListener(name, listener))
}
```

```
export function useEvent(_el, name, listener, options) {

  const element = wrap(_el)

  watch(element, (el, _, onCleanup) => {
    el && el.addEventListener(name, listener, options)
  })

  onMounted(() => element.value.addEventListener(name, listener, options))

  onUnmounted(() => element.value.removeEventListener(name, listener))
}
```

```
export function useEvent(_el, name, listener, options) {

  const element = wrap(_el)

  watch(element, (el, _, onCleanup) => {
    el && el.addEventListener(name, listener, options)
  })

  onUnmounted(() => element.value.removeEventListener(name, listener))
}
```

```
export function useEvent(_el, name, listener, options) {

  const element = wrap(_el)

  watch(element, (el, _, onCleanup) => {
    el && el.addEventListener(name, listener, options)

    onCleanup(() => el && el.removeEventListener(name, listener))
  })

  onUnmounted(() => element.value.removeEventListener(name, listener))
}
```

```
export function useEvent(_el, name, listener, options) {

  const element = wrap(_el)

  watch(element, (el, _, onCleanup) => {
    el && el.addEventListener(name, listener, options)

    onCleanup(() => el && el.removeEventListener(name, listener))
  })
}
```

- Listeners are only added when the element actually exists
- listeners are updated when element ref changes

RETURN COMPUTED > RETURN REF

```
createComponent({
  setup() {
    const isOnline = useOnline()

    return {
      isOnline,
    }
  },
})
```

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

export default function useOnline() {

  const isOnline = ref(true)

  isOnline.value = window.navigator ? window.navigator.onLine : true

  const onlineHandler = () => (isOnline.value = true)
  const offlineHandler = () => (isOnline.value = false)
  window.addEventListener('online', onlineHandler, false)
  window.addEventListener('offline', offlineHandler, false)

  onUnmounted(() => {
    window.removeEventListener('online', onlineHandler)
    window.removeEventListener('offline', offlineHandler)
  })

  return isOnline
}
```

🧐 This ref is mutable!

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

export default function useOnline() {

  const isOnline = ref(true)

  isOnline.value = window.navigator ? window.navigator.onLine : true

  const onlineHandler = () => (isOnline.value = true)
  const offlineHandler = () => (isOnline.value = false)
  window.addEventListener('online', onlineHandler, false)
  window.addEventListener('offline', offlineHandler, false)

  onUnmounted(() => {
    window.removeEventListener('online', onlineHandler)
    window.removeEventListener('offline', offlineHandler)
  })

  return computed(() => isOnline.value)
}
```

😊

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

export default function useOnline() {

  const isOnline = ref(true)

  isOnline.value = window.navigator ? window.navigator.onLine : true

  const onlineHandler = () => (isOnline.value = true)
  const offlineHandler = () => (isOnline.value = false)
  window.addEventListener('online', onlineHandler, false)
  window.addEventListener('offline', offlineHandler, false)

  onUnmounted(() => {
    window.removeEventListener('online', onlineHandler)
    window.removeEventListener('offline', offlineHandler)
  })

  return readonly({
    isOnline,
    a: 'A',
    b: 'B',
  })

}
```

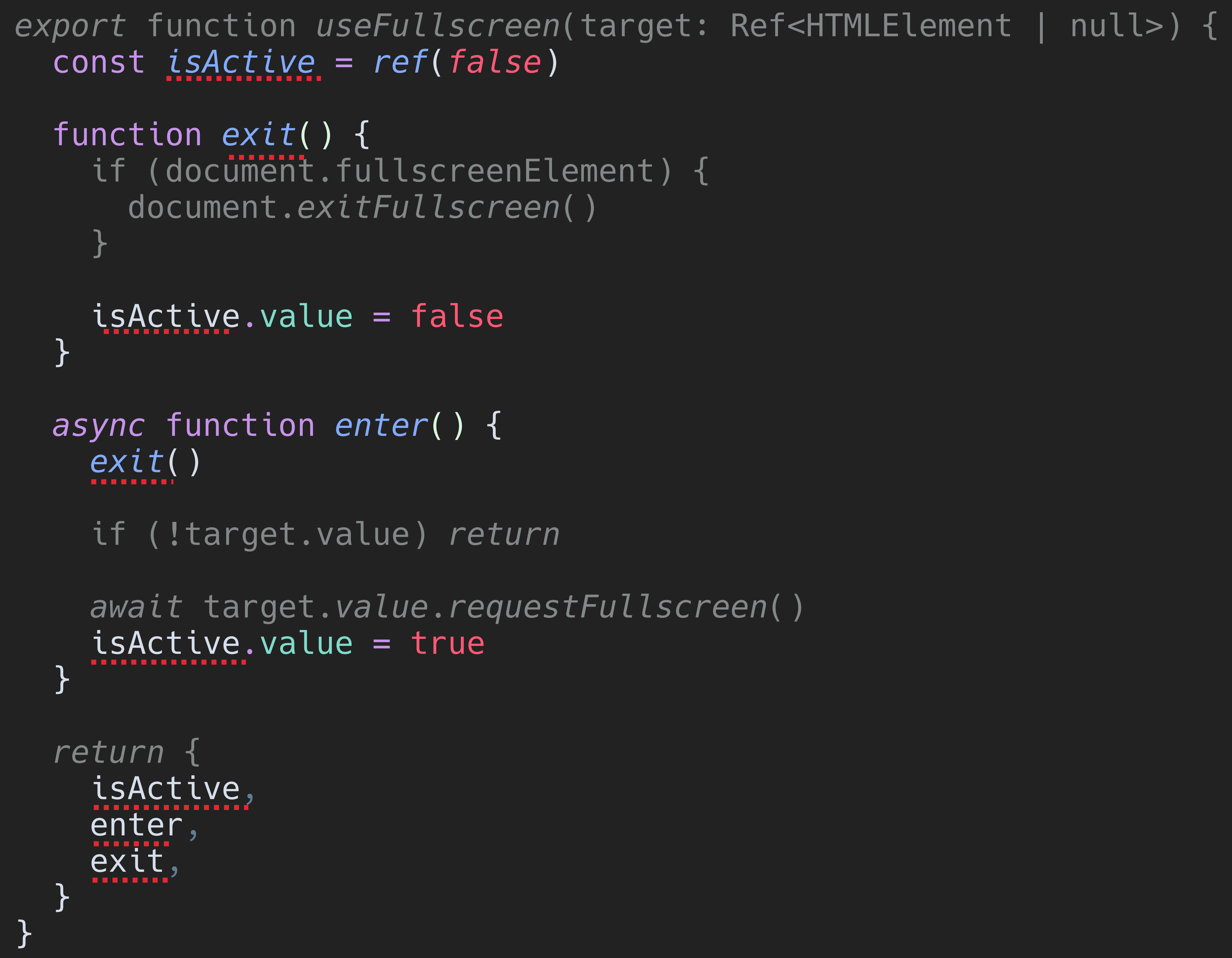
😊

NAME RETURNED PROPERTIES IN CONTEXT

```
export function useFullscreen(target: Ref<HTMLElement | null>) {
  const isFullscreen = ref(false)

  function exitFullscreen() {
    if (document.fullscreenElement) {
      document.exitFullscreen()
    }

    isFullscreen.value = false
  }

  async function enterFullscreen() {
    exitFullscreen()

    if (!target.value) return

    await target.value.requestFullscreen()
    isFullscreen.value = true
  }

  return {
    isFullscreen,
    enterFullscreen,
    exitFullscreen,
  }
}
```

```
createComponent({
  setup() {
    const el = ref<HTMLElement>(null)
    const fullscreen = useFullscreen(el)

    onMounted(() => fullscreen.enterFullscreen)
    return {
      el,
      fullscreen,
    }
  },
})
```

🧐



```
export function useFullscreen(target: Ref<HTMLElement | null>) {
  const isActive = ref(false)

  function exit() {
    if (document.fullscreenElement) {
      document.exitFullscreen()
    }

    isActive.value = false
  }

  async function enter() {
    exit()

    if (!target.value) return

    await target.value.requestFullscreen()
    isActive.value = true
  }

  return {
    isActive,
    enter,
    exit,
  }
}
```

```
createComponent({
  setup() {
    const el = ref<HTMLElement>(null)
    const fullscreen = useFullscreen(el)

    onMounted(fullscreen.enter)

    return {
      el,
      fullscreen,
    }
  },
})
```

```
createComponent({
  setup() {
    const el = ref<HTMLElement>(null)
    const { enter: enterFullscreen } = useFullscreen(el)

    onMounted(enterFullscreen)

    return {
      el,
      enterFullScreen,
    }
  },
})
```

😊

THANKS!

Twitter: @linus_borg

Github: linusborg