



UNIVERSIDADE  
FEDERAL DO CEARÁ  
CAMPUS QUIXADÁ

# Criando uma app SPA c/ Vue.js

QXD0020 - Desenvolvimento de Software para Web

Prof. Bruno Góis Mateus ([brunomateus@ufc.br](mailto:brunomateus@ufc.br))

# Agenda

- Introdução
- Single File Components
- Introdução ao Vuex
- Introdução ao VueRouter
- Migrando nossa Manga Store

# Introdução



# Introdução



- Aplicações

- Devem ter o melhor desempenho possível
  - Código compacto e conciso
- Compatíveis com o maior número de navegadores
  - Não necessariamente os mais novos

- Desenvolvedores

- Querem códigos fáceis de escrever e confortáveis de ler
- Desejam usar as funcionalidades mais modernas de JS

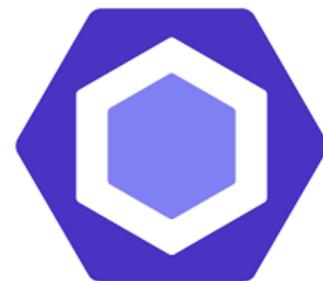
# Introdução

- Desenvolvedores Js criaram soluções para contornar tais contradições

*BABEL*



- Além disso outras ferramentas foram incorporadas ao desenvolvendo em JS



# Introdução

## Webpack

- O problema da dependências em Js
  - O HTML que não prover uma solução ideal para o problema
    - Colisão de nomes de variáveis globais
    - Ordem de carregamento
    - Otimizações de desempenho (ex, carregamento assíncrono)
- Solução: o Sistema de Módulo (*module system*)
  - No entanto, nem todos os navegadores dão suporte

# Introdução

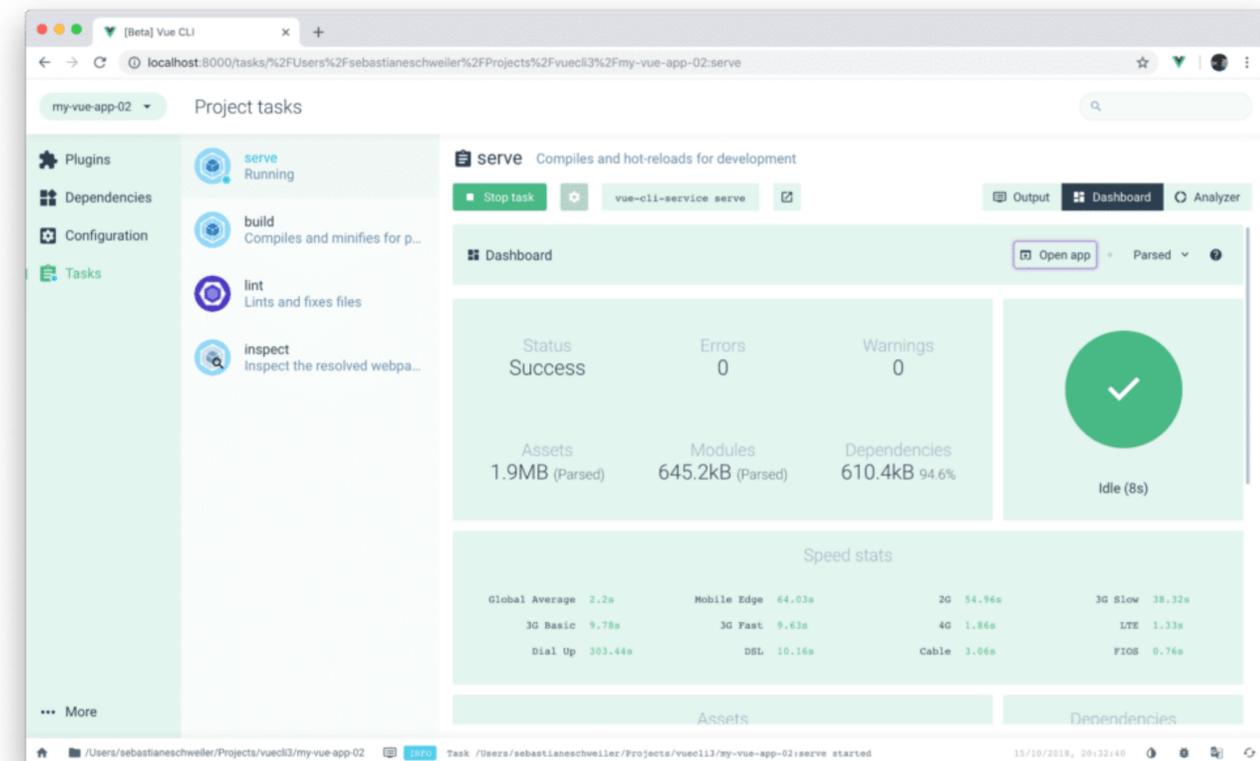
## Webpack

- Bundling
  - O Webpack analisa as dependências do arquivo de entrada (apps.js)
    - Analisa as outras dependências recursivamente
  - Gera um arquivo Js único compatível com “qualquer” navegador
- Loaders
  - Permite a transformação de qualquer arquivo antes do empacotamento
    - Babel Loader, Vue Loader, Sass Loader, TypeScript Loader
- Hot module reloading

# Introdução

## Webpack

- Configurar o Webpack é uma atividade complicada
- Em geral, os desenvolvedores precisam das mesmas configurações básicas



# Introdução

## Vue CLI

### Vue CLI

- Ferramenta padrão de desenvolvimento usando Vue.js
- Simplifica a conversão de código
  - Esconde as complexidades do Webpack
  - Usa Babel e TypeScript
- Torna o processo de desenvolvimento mais eficiente através do hot-swapping
  - Webpack transpira o código e realizar o hot-swapping a cada mudança

# Introdução

## Vite

Criado por Evan You

Baseado no sistema de módulos do ECMA6.  
Não realiza transpilação.

Compatível com outros frameworks como React e Svelte



Não é baseado no WebPack

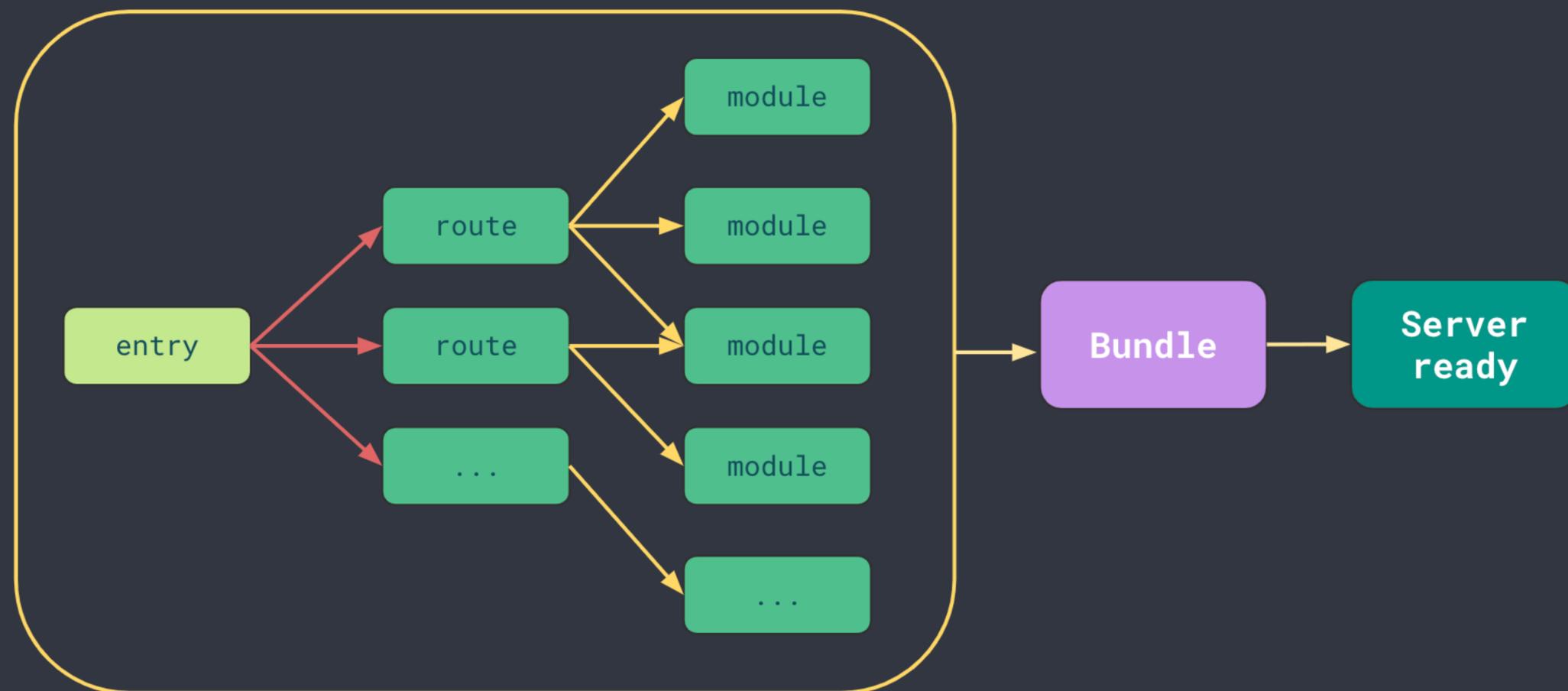
O projeto não é construído durante o desenvolvimento.  
O tempo de startup e compilação é reduzido.

Em produção, Vite utiliza o Rollup.js para empacotar o projeto

# Introdução

## Vite

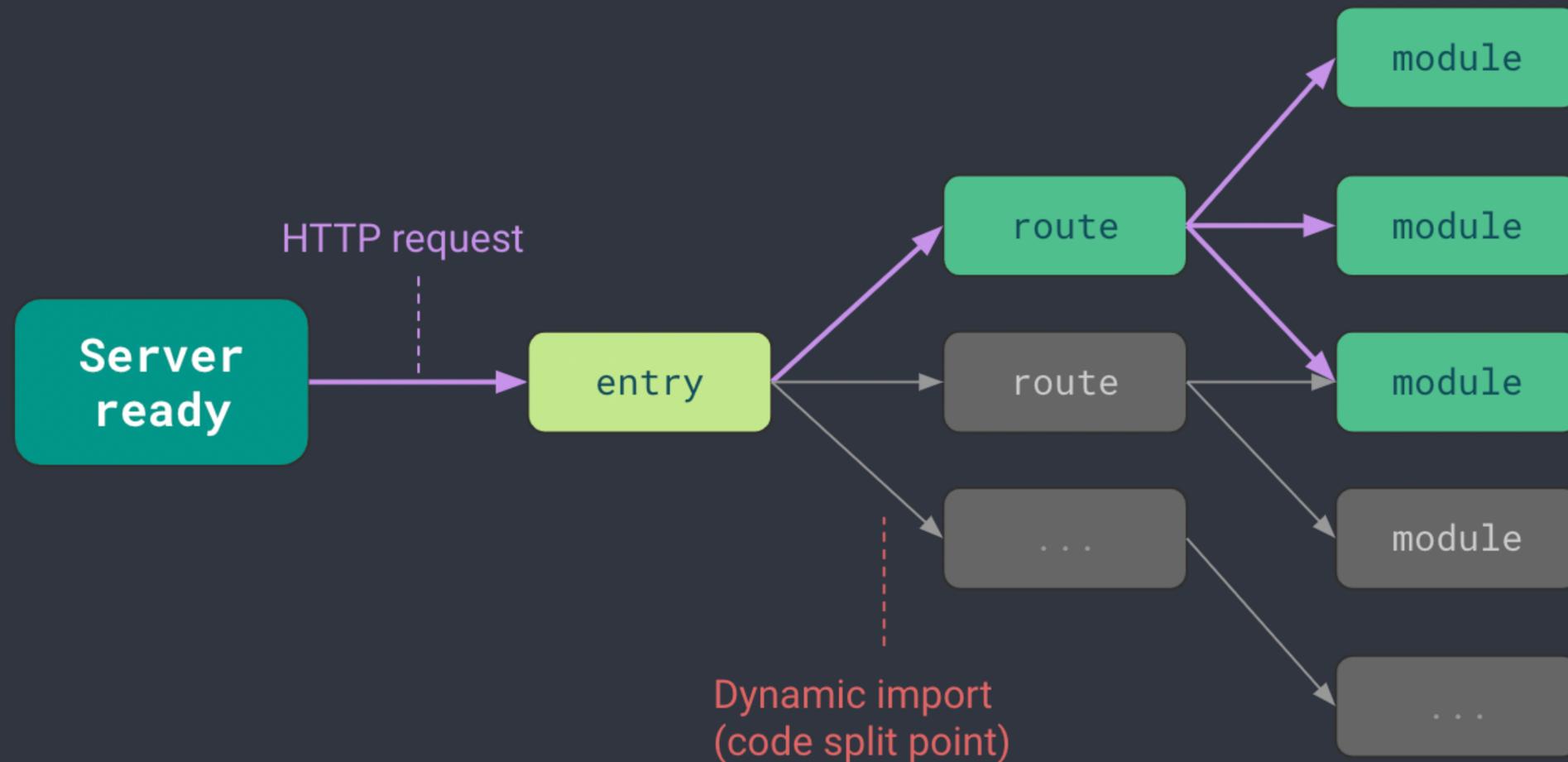
### Bundle based dev server



# Introdução

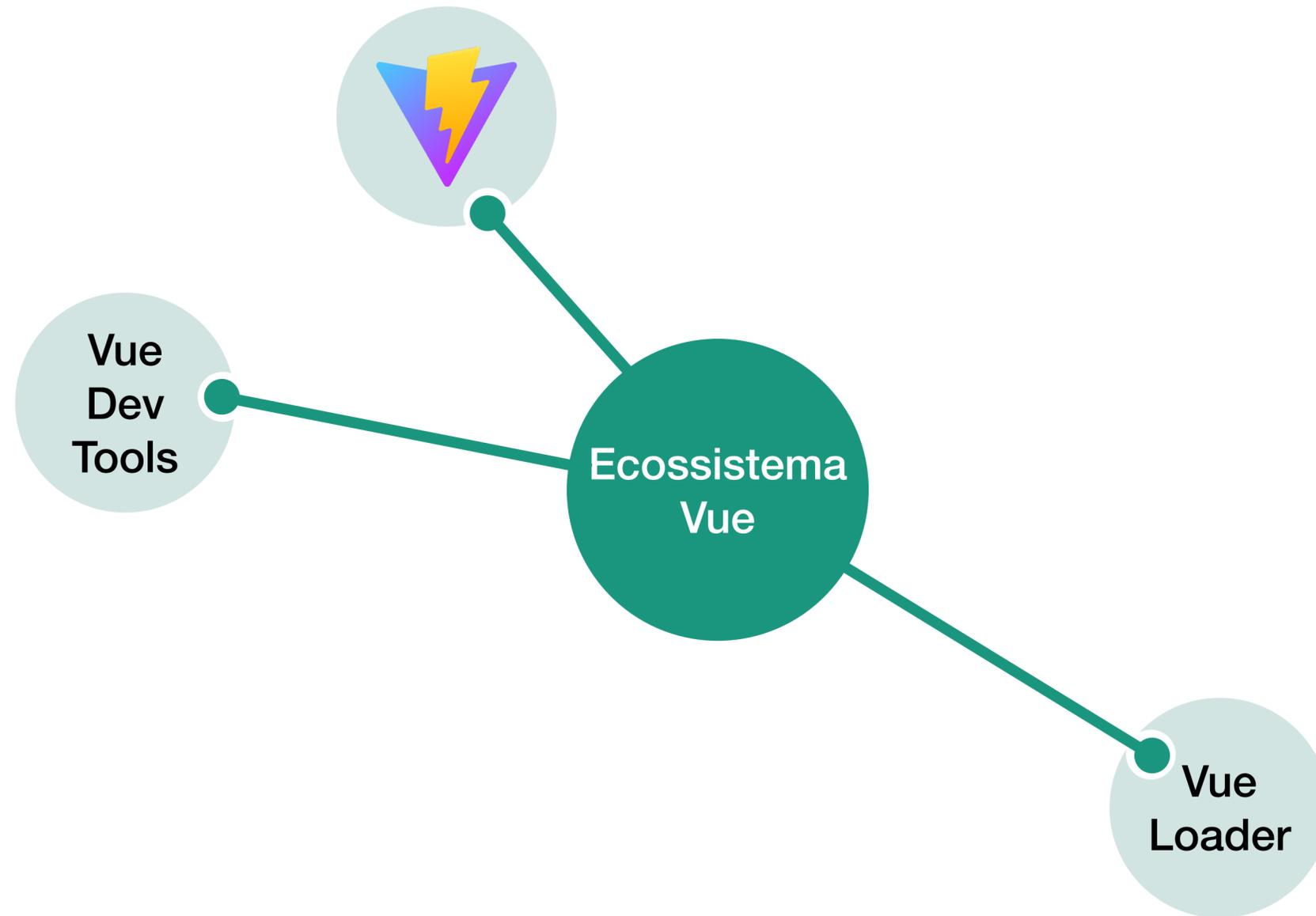
## Vite

### Native ESM based dev server

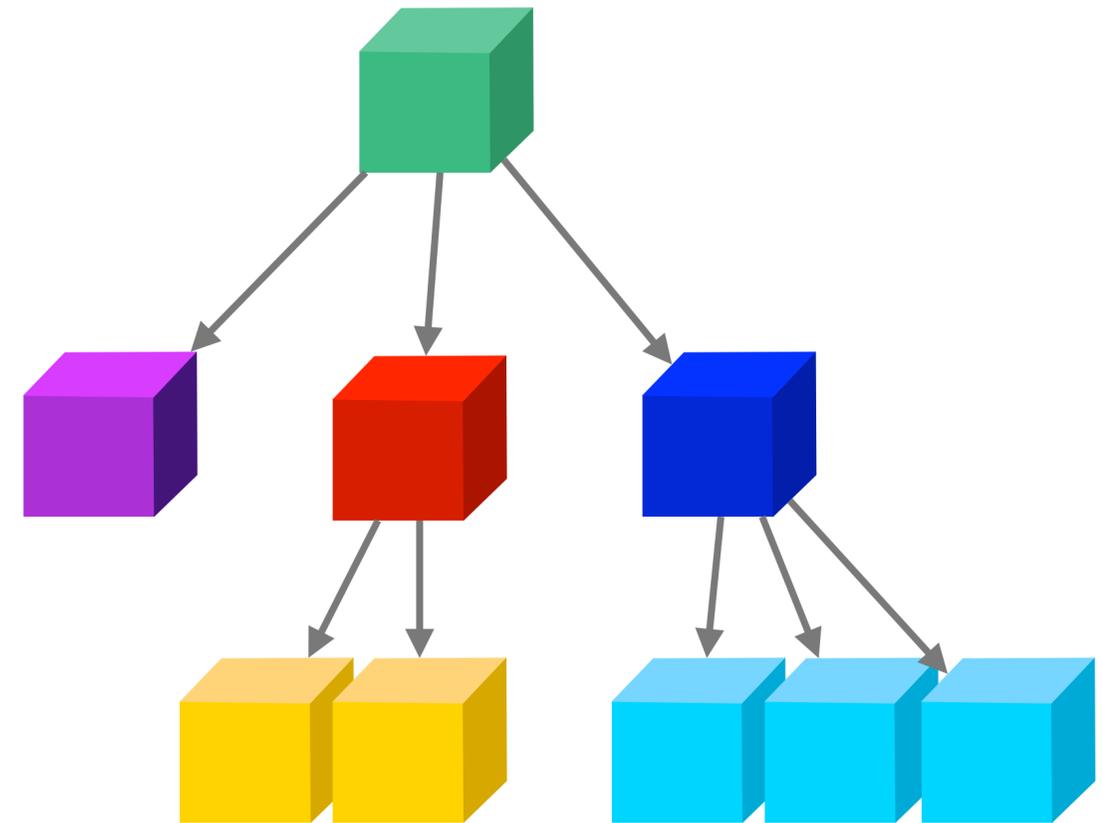


# Introdução ao VueJS

## Ecosystema



# Single File Components



# Single File Components

## Single File Components - SFC

- Formato especial que permite o encapsulamento do template (HTML), lógica (JS) e estilo (CSS) de um componente Vue em um único arquivo ( `.vue` )
- São um formato específico do Vue que **precisa ser compilado** em Js e CSS
- Em geral, nos projetos o **compilador de SFC** são integrados a ferramentas de build como **Vue CLI** e **Vite**

# Single File Components

## Motivos para usar SFC

- Permite a criação de componentes modularizados usando linguagem familiares: HTML, CSS e JavaScript
- Template são pré-compilados
- CSS com escopo
- Sintaxe facilitada quando usado em conjunto com a Composition API
- Suporte das IDEs: Auto-complete e checagem de tipos
- Suporte ao Hot-Module Replacement

# Single File Components

```
<template>  
  Seu HTML  
</template>
```

```
<script>  
  Seu JS  
</script>
```

```
<style>  
  Seu CSS  
</style>
```

# Single File Components

## <template>

- Cada arquivo vue pode conter um bloco `<template>` no mais alto nível
- O conteúdo do bloco é extraído e passado para o `@vue/compiler-dom`
- Pré-processadores
  - Aceita código escrito em [Pug](#)

# Single File Components

## <script>

- Cada arquivo vue pode conter no máximo um bloco `<script>`
  - Pode ser usado em conjunto com `<script setup>`
- É executado como um módulo ES
- O export default deve ser:
  - Um componente Vue no formato option object
  - Um objeto plano
  - Retorno da chamada a função `defineComponent`

# Single File Components

`<style>`

- Cada arquivo vue pode conter vários blocos `<style>`
- Podem possuir atributos como `scoped` ou `module` que ajudam a escopular o estilo no component ao qual ele pertence
- Pré-processadores
  - É possível utilizar SASS

# Single File Components

```
<template>
  <div class="example">{{ msg }}</div>
</template>
```

```
<script>
export default {
  data() {
    return {
      msg: 'Hello world!'
    }
  }
}
</script>
```

```
<style>
.example {
  color: red;
}
</style>
```

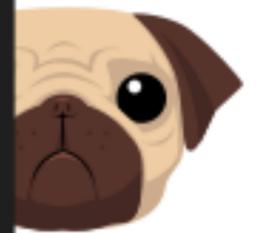
# Single File Components



```
<template>  
  ...  
</template>  
  
<script>  
  ...  
</script>  
  
<style>  
  ...  
</style>
```



```
<template lang="pug">  
  ...  
</template>  
  
<script lang="ts">  
  ...  
</script>  
  
<style lang="postcss">  
  ...  
</style>
```



# Single File Components

## <script setup> - Composition API

- Cada arquivo vue pode conter apenas um bloco `<script setup>`
- Permite que desenvolvedores definam componentes sem a necessidade do bloco `export`
  - Basta definir suas variáveis e usá-las no template
- Código é executado uma vez para cada instância do componente

# Single File Components

## Migração da Option API para Composition API

```
<template>
  <div>Hello, {{ name }}!</div>
  <input v-model="name" />
  <button :disabled="!isNamePresent" @click="submitName">Submit</button>
</template>

<script>
export default {
  data() {
    return {
      name: ''
    }
  },
  computed: {
    isNamePresent() {
      return this.name.length > 0
    }
  },
  methods: {
    submitName() {
      console.log(this.name)
    }
  }
}
</script>

<script>
import { ref, computed } from 'vue'

export default {
  setup() {
    const name = ref('')
    const isNamePresent = computed(() => name.value.length > 0)

    function submitName() {
      console.log(name.value)
    }

    return {
      name,
      isNamePresent,
      submitName
    }
  }
}
</script>
```

# Single File Components

## Migração da Option API para Composition API

```
<template>
  <div>Hello, {{ name }}!</div>
  <input v-model="name" />
  <button :disabled="!isNamePresent" @click="submitName">Submit</button>
</template>

<script>
export default {
  setup() {
    const name = ref('')
    const isNamePresent =
      computed(() => name.value.length > 0)

    function submitName() {
      console.log(name.value)
    }

    return {
      name,
      isNamePresent,
      submitName
    }
  }
}
</script>
```

```
<script setup>
import { ref, computed } from 'vue'

const name = ref('')
const isNamePresent =
  computed(() => name.value.length > 0)

function submitName() {
  console.log(name.value)
}
</script>
```

# Single File Components

## Migração da Option API para Composition API

```
<template>
  <div>Hello, {{ name }}!</div>
  <input v-model="name" />
  <button :disabled="!isNamePresent" @click="submitName">Submit</button>
</template>

<script setup>
import { ref, computed } from 'vue'

const name = ref('')
const isNamePresent = computed(() => name.value.length > 0)

function submitName() {
  console.log(name.value)
}
</script>
```

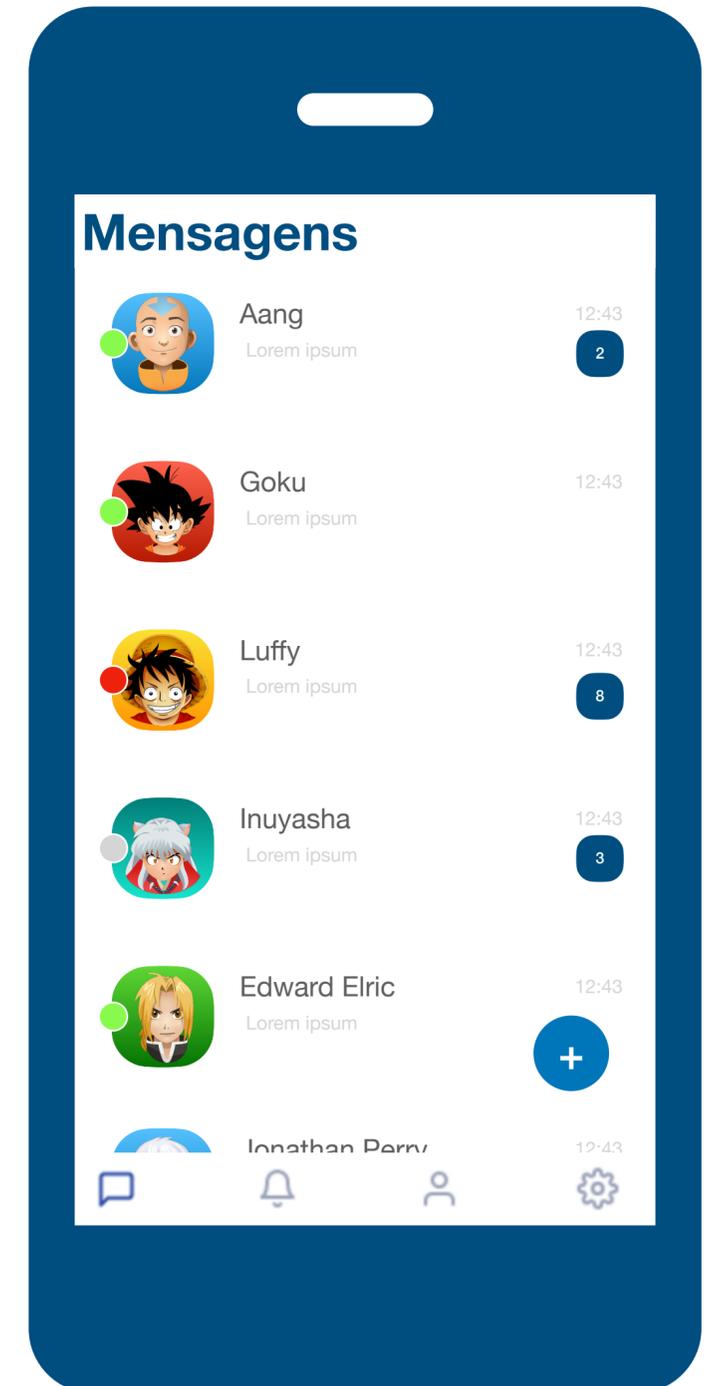
# Introdução ao Vuex



# Introdução ao Vuex

## Motivação

- Imagine que você desenvolveu uma aplicação de chat
  - Lista de usuário, chat privados, histórico de conversas
  - Barra de notificação que informa sobre mensagens não lidas enviadas por outros usuários
- Milhões de usuários usam sua aplicação todos os dias
- Reclamação: vez por outra a barra de navegação mostra notificações falsas



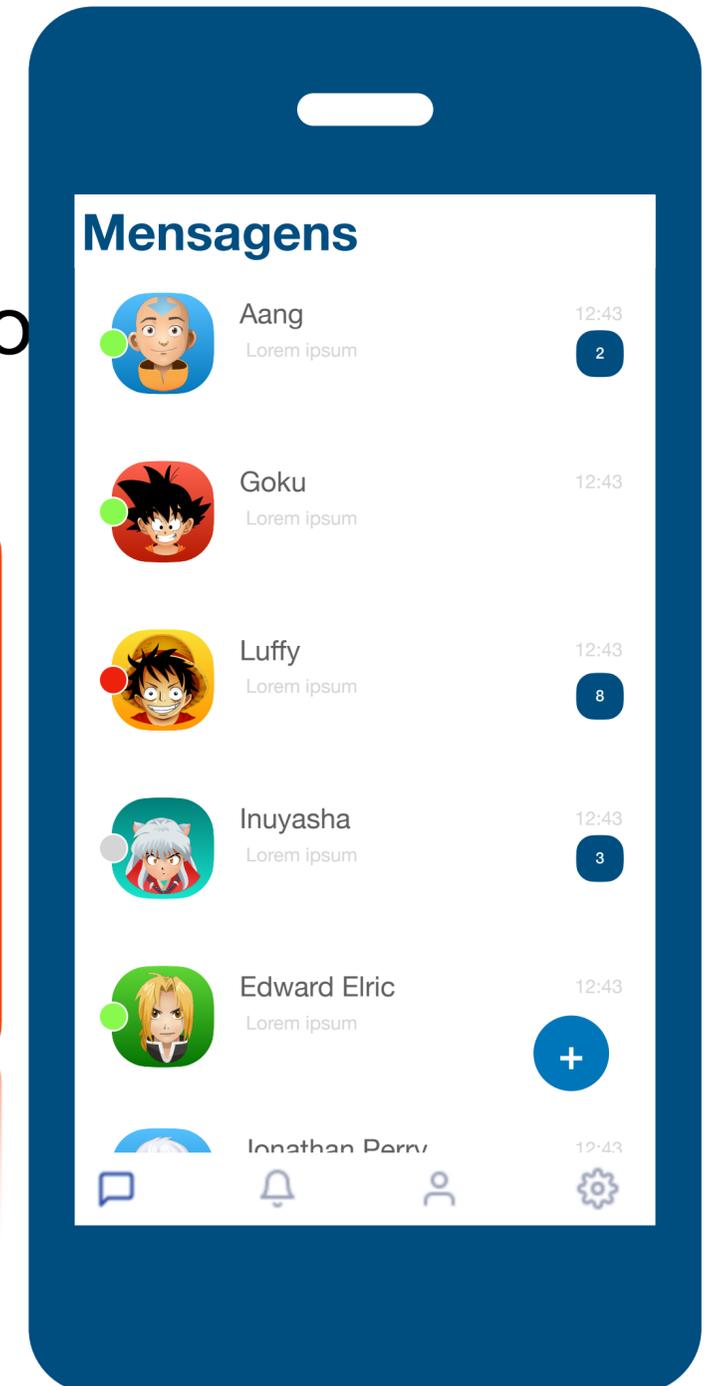
# Introdução ao Vuex

## Motivação

- A situação anterior “zombie notification” foi enfrentada pelo desenvolvedores do Facebook a alguns anos atrás

Quando múltiplos componentes de uma aplicação compartilham data, a complexidade das interconexões irão aumentar até que não seja mais possível prever ou entender o estado dos dados. Conseqüentemente, a aplicação se torna impossível de estender ou manter.

- A solução do problema serviu de inspiração para a criação de um padrão arquitetural



# Introdução ao Vuex

## Flux

- É um padrão arquitetural e não um biblioteca
- Conjunto de princípios que descrevem um arquitetura escalável para frontend
  - Aplicável em qualquer aplicação complexa
- Implementações



# Introdução ao Vuex

## Princípios

### Single Source of Truth

- Qualquer dado compartilhado entre componentes, devem ser mantidos em um único local, separado dos componentes que o utilizam
  - Este local único é chamado de **store**
  - Componentes devem **ler dados da store**
- Componentes podem ter dados locais que apenas eles devem conhecer
  - Ex: A posição de uma barra de navegação em um componente de lista

# Introdução ao Vuex

## Princípios

### Data is read-only

- Componentes podem ler os dados da **store** livremente, no entanto, eles não podem alterar os dados contidos na **store**
  - Componentes informam a intenção de alterar algum dado
  - A **store** realizar essas mudanças (*mutations*)

# Introdução ao Vuex

## Princípios

### Mutations are synchronous

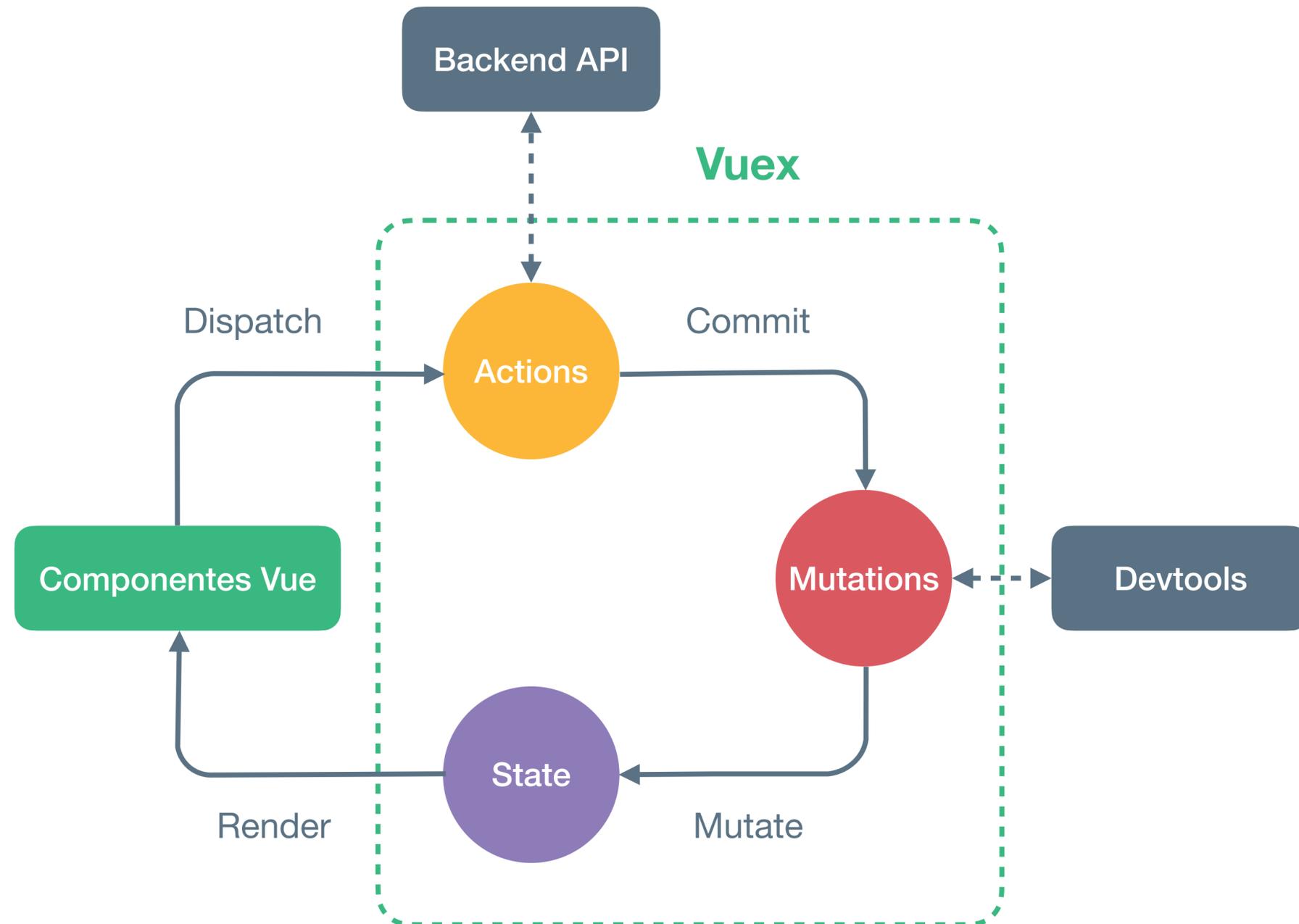
- *Mutations* síncronas garantem que o estado dos dados não dependem de um sequência e do tempo de execução de eventos imprevisíveis

# Introdução ao Vuex

## Vuex

- Biblioteca que facilita a implementação da arquitetura Flux
  - State Management Pattern + library
- Armazena os dados de forma centralizada garantindo que os estados só podem ser mudados de uma forma previsível

# Introdução ao Vuex



# Introdução ao Vuex

## Core concepts

### State

- Vuex utiliza uma *single state tree*, árvore única para todos os estados
  - *Single source of truth*
  - Torna simples a tarefa de localizar um pedaço específico dos estados

# Introdução ao Vuex

## Core concepts - State

```
import { createApp } from 'vue'
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state () {
    return {
      count: 0
    }
  }
})

const app = createApp({ /* your root
component */ })

app.use(store)
```

```
const Counter = {
  template: `<div>{{ count }}</div>`,
  computed: {
    count () {
      return this.$store.state.count
    }
  }
}
```



# Introdução ao Vuex

## Core concepts

### Getters

- Algumas vezes precisamos de um estado derivado do estado da store
- Funcionam como propriedades computadas (computed) em stores

# Introdução ao Vuex

## Core concepts - Getters

```
const store = createStore({
  state: {
    todos: [
      { id: 1, text: '...', done: true },
      { id: 2, text: '...', done: false }
    ]
  },
  getters: {
    doneTodos (state) {
      return state.todos.filter(todo => todo.done)
    },
    doneTodosCount (state, getters) {
      return getters.doneTodos.length
    },
    getTodoById: (state) => (id) => {
      return state.todos.find(todo => todo.id ===
id)
    }
  }
})
```

```
computed: {
  doneTodosCount () {
    return this.$store.getters.doneTodosCount
  }
}
```

```
import { mapGetters } from 'vuex'

export default {
  // ...
  computed: {
    // mix the getters into computed with
    object spread operator
    ...mapGetters([
      'doneTodosCount',
      'doneTodos',
      // ...
    ])
  }
}
```

# Introdução ao Vuex

## Core concepts

### Mutations

- A única forma de alterar um estado da store é por meio de um commit de mudança
- São similares a eventos
  - Tipo (string)
  - Handler
    - Função onde as mudanças realmente acontecem
    - Não é possível invocá-los diretamente
- **Obrigatoriamente devem ser síncronos**

# Introdução ao Vuex

## Core concepts - Mutation

```
const store = createStore({
  state: {
    count: 1
  },
  mutations: {
    increment (state, n) {
      // mutate state
      state.count += n
    }
  }
})
```

```
store.commit('increment', n)
```



# Introdução ao Vuex

## Core concepts - Mutation

```
const store = createStore({
  state: {
    count: 1
  },
  mutations: {
    increment (state, payload) {
      // mutate state
      state.count += payload.amount
    }
  }
})
```

```
store.commit('increment', {
  amount: 10
})
```

```
store.commit({
  type: 'increment',
  amount: 10
})
```

# Introdução ao Vuex

## Core concepts

### Actions

- Similares a *mutations*
  - Não alteram o estado e sim commitam mudanças
  - Podem ser assíncronas

# Introdução ao Vuex

## Core concepts - Mutation

```
const store = createStore({
  state: {
    count: 0
  },
  mutations: {
    increment (state) {
      state.count++
    }
  },
  actions: {
    incrementAsync (context) {
      context.commit('increment')
    }
  }
})
```

```
store.dispatch('incrementAsync', {
  amount: 10
})
```

```
store.dispatch({
  type: 'incrementAsync',
  amount: 10
})
```

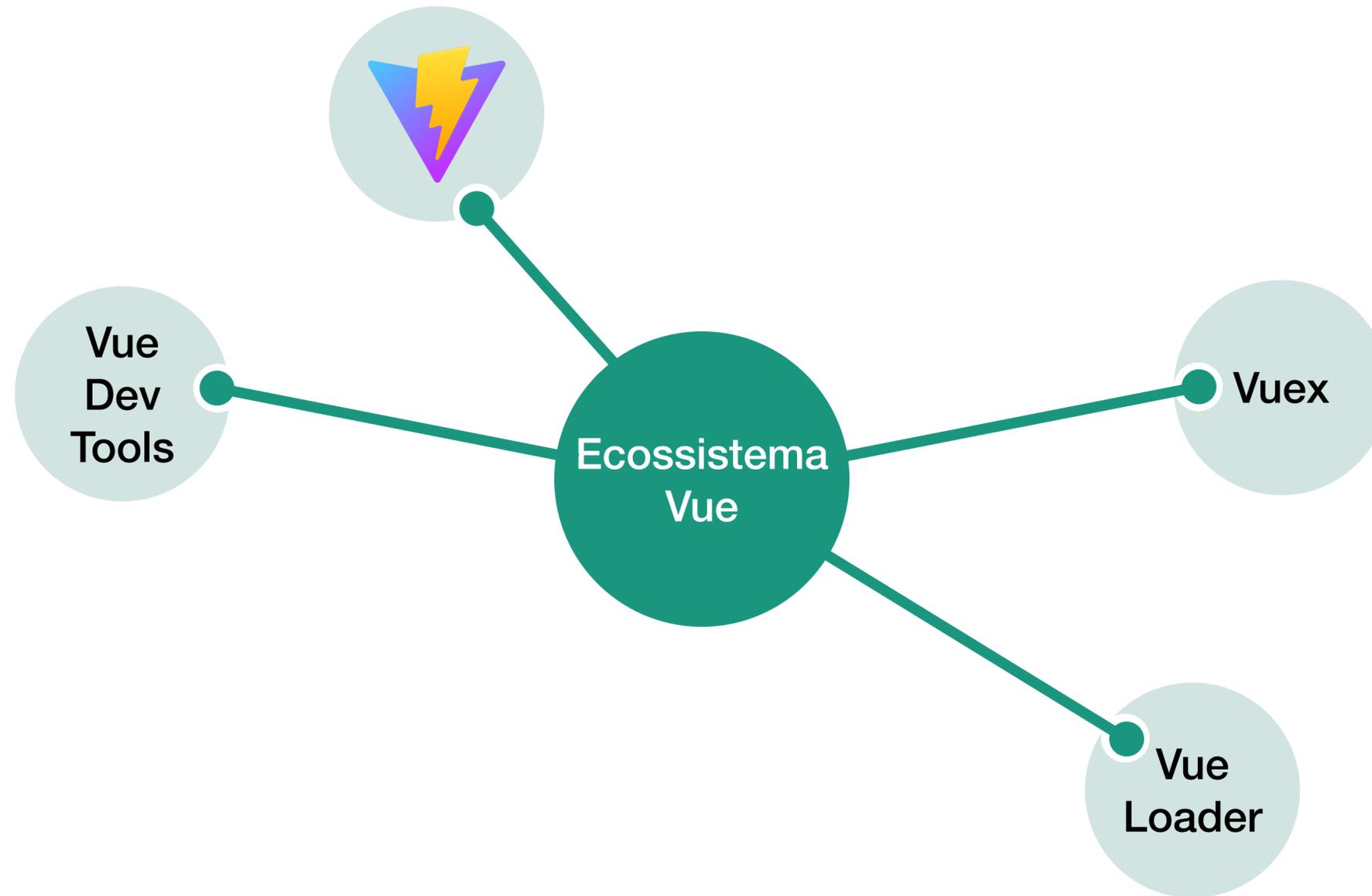
# Introdução ao Vuex

## Estrutura de uma aplicação

```
├── index.html
├── main.js
├── api
│   └── ... # abstractions for making API requests
├── components
│   ├── App.vue
│   └── ...
├── store
│   ├── index.js # where we assemble modules and export the store
│   ├── actions.js # root actions
│   ├── mutations.js # root mutations
│   └── modules
│       ├── cart.js # cart module
│       └── products.js # products module
```

# Introdução ao VueJS

## Ecosystema



# Introdução ao Vue router



# Introdução ao Vue router

## Vue Router

- Roteador oficial do Vue.js
  - Criado pelo autor do Vue, Evan You
- Ajuda na atualização da *view* em SPA (*Single page application*)

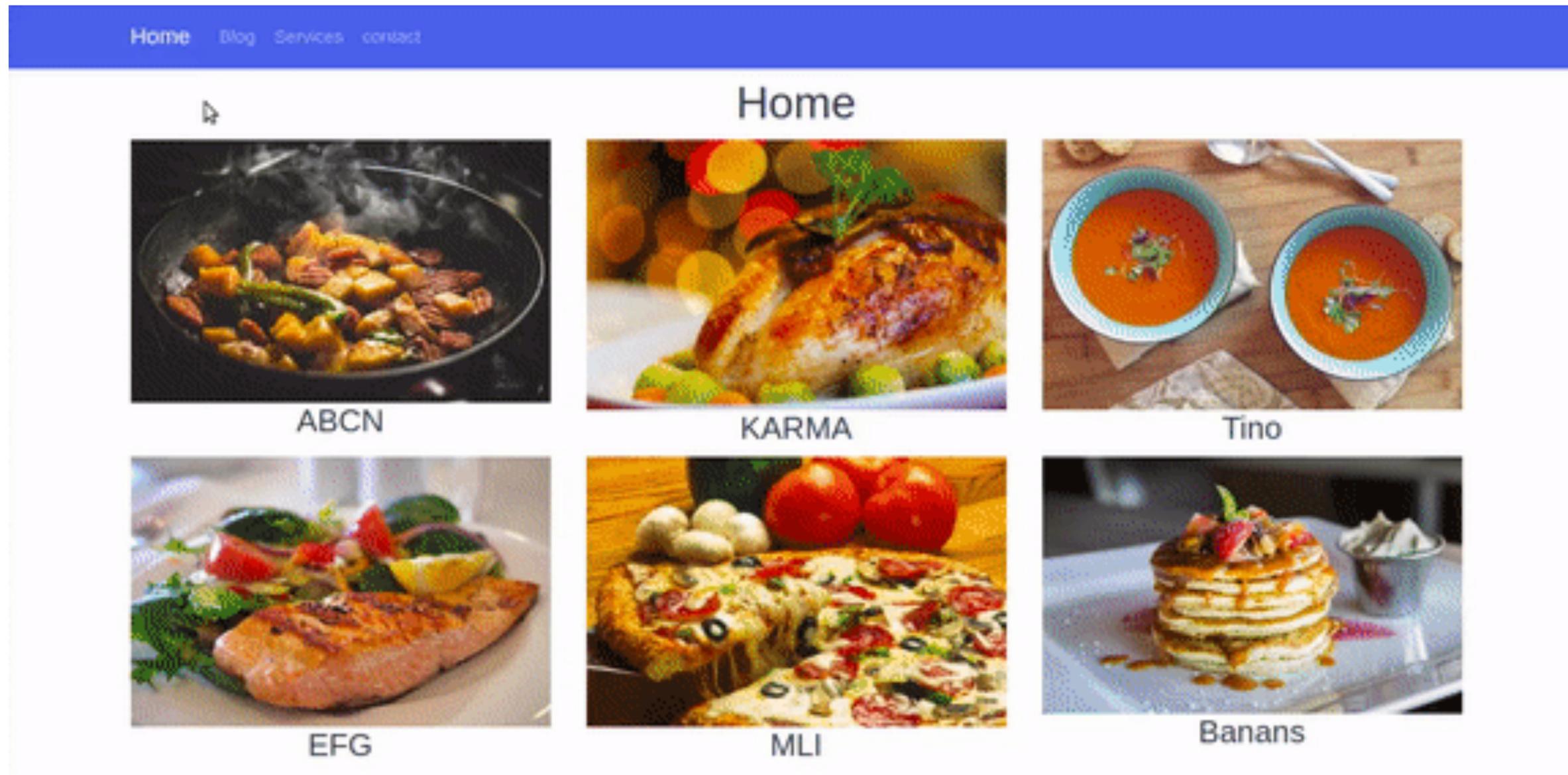
# Introdução ao Vue router

## Principais funcionalidades

- Mapeamento de rotas aninhadas
- Roteamento dinâmico
- Configuração modular baseada em componentes
- Rotas com parâmetros, query, wildcards
- Efeito de transição entre *Views*
- Ajuste fino do controle de navegação
- Diferentes modos de histórico de navegação
- Comportamento de rolagem customizável

# Introdução ao Vue router

## Demonstração



# Introdução ao Vue router

```
<div id="app">
  <h1>Hello App!</h1>
  <p>
    <router-link to="/">
      Go to Home
    </router-link>
    <router-link to="/about">
      Go to About
    </router-link>
  </p>
  <router-view></router-view>
</div>
</html>
```

```
const routes = [
  { path: '/', component: Home },
  { path: '/about', component: About },
]

const router = VueRouter.createRouter({
  history: VueRouter.createWebHashHistory(),
  routes, // short for `routes: routes`
})

const app = Vue.createApp({})

app.use(router)

app.mount('#app')
```

Utilizado para criar um link. Substitui a tag <a>

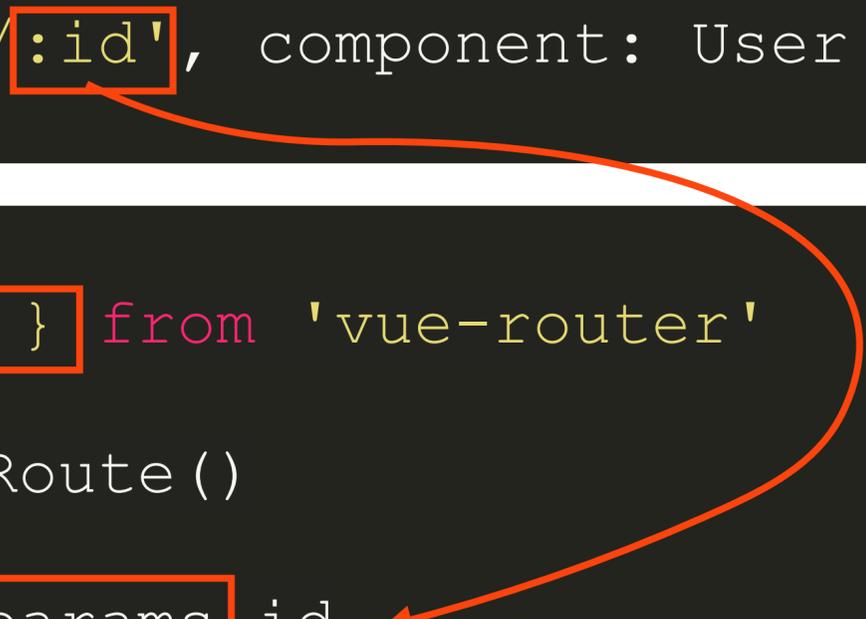
Local onde o componente associado a rota será renderizado

# Introdução ao Vue router

## Roteamento dinâmico

```
const routes = [  
  // dynamic segments start with a colon  
  { path: '/users/:id', component: User },  
]
```

```
<script setup>  
import { useRouter } from 'vue-router'  
  
const route = useRouter()  
  
const id = route.params.id  
</script>
```

An orange arrow originates from the ':id' segment in the first code block and points to the 'route.params.id' property access in the second code block, illustrating how the dynamic segment is resolved into a route parameter.

# Introdução ao Vue router

## Roteamento dinâmico

Padrão da URL	matched path	\$route.params
/users/:username	/users/eduardo	{ username: 'eduardo' }
/users/:username/posts/:postId	/users/eduardo/posts/123	{ username: 'eduardo', postId: '123' }

# Introdução ao Vue router

## Roteamento dinâmico - Named routes

```
const routes = [  
  {  
    path: '/user/:username',  
    name: 'user',  
    component: User  
  }  
]
```

```
<router-link :to="{ name: 'user', params: { username: 'erina' }}">  
  User  
</router-link>
```

# Introdução ao Vue router

## Navegação programática

```
import { useRouter } from 'vue-router'

const router = useRouter()

router.push('/users/eduardo')

router.push({ path: '/users/eduardo' })

router.push({ name: 'user', params: { username: 'maria' } })

router.push({ path: '/register', query: { plan: 'private' } })

router.push({ path: '/about', hash: '#team' })

router.push({ path: '/home', replace: true })

router.replace({ path: '/home' })
```

/home

/register?plan=private

/users/maria

/users/eduardo

# Introdução ao Vue router

## Redirect e Alias

```
const routes = [{ path: '/home', redirect: '/' }]
```

```
const routes = [{ path: '/home', redirect: { name: 'homepage' } }]
```

```
const routes = [{ path: '/', component: Homepage, alias: '/home' }]
```

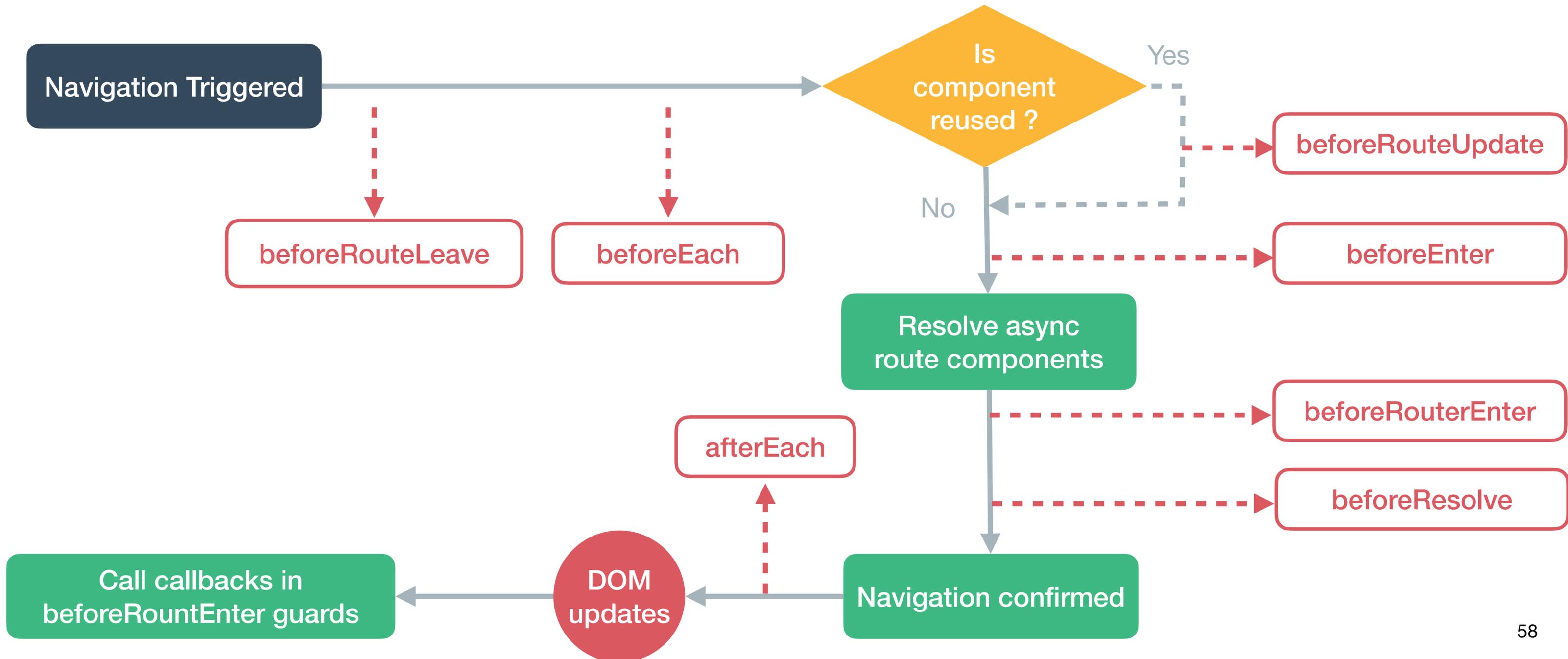
# Introdução ao Vue router

## Navigation Guards

- Pontos de interferência fornecidos pelo Vue router para customização do processo de navegação
  - Em geral, utilizado para redirecionar ou cancelar uma rota
- Existem 3 opções de “guardas”
  - Globais: `beforeEach`, `beforeResolve` e `afterEach`
  - Por rota: `beforeEnter`
  - Em componentes: `beforeRouteEnter`, `beforeRouteUpdate` e `beforeRouteLeave`

# Introdução ao Vue router

## Navigation Guards



# Introdução ao Vue router

## beforeEach

```
router.beforeEach(async (to, from) => {  
  // canUserAccess() returns `true` or `false`  
  const canAccess = await canUserAccess(to)  
  if (!canAccess) return '/login'  
})
```

- Chamado sempre que a navegação é iniciada
- Retorno
  - false -> Cancela a navegação
  - Route Location -> Mesmo efeito de chamar route.push
  - Nada -> Navegação é confirmada

# Introdução ao Vue router

## beforeResolve

```
router.beforeResolve(async to => {
  if (to.meta.requiresCamera) {
    try {
      await askForCameraPermission()
    } catch (error) {
      if (error instanceof NotAllowedError) {
        return false
      } else {
        throw error
      }
    }
  }
})
```

# Introdução ao Vue router

## afterEach

```
router.afterEach((to, from) => {  
  sendToAnalytics(to.fullPath)  
})
```

# Introdução ao Vue router

## beforeEnter

```
const routes = [  
  {  
    path: '/users/:id',  
    component: UserDetails,  
    beforeEnter: (to, from) => {  
      // reject the navigation  
      return false  
    },  
  },  
],  
]
```

# Introdução ao Vue router

## beforeEnter

```
function removeQueryParams(to) {
  if (Object.keys(to.query).length)
    return { path: to.path, query: {}, hash: to.hash }
}
function removeHash(to) {
  if (to.hash) return { path: to.path, query: to.query, hash: '' }
}
const routes = [
  {
    path: '/users/:id', component: UserDetails,
    beforeEnter: [removeQueryParams, removeHash],
  },
  {
    path: '/about', component: UserDetails,
    beforeEnter: [removeQueryParams],
  },
]
```

# Introdução ao Vue router

## Em componentes

```
onBeforeRouteUpdate((to, from) => {  
  this.name = to.params.name  
})
```

```
onBeforeRouteLeave((to, from) => {  
  const answer = window.confirm('Do you really want to leave? you have  
  unsaved changes!')  
  if (!answer) return false  
})
```

# Introdução ao Vue router

## Data fetching

- Existem duas opções para busca/recuperação dados (*data fetching*)
  - Após a navegação
    1. A navegação é realizada
    2. O componente é renderizado
    3. Os dados são recuperados nos *hooks* (`created`) do componente
  - Antes da navegação
    - Os dados são recuperados antes (`beforeRouteEnter`)
    - A navegação é realizada

# Introdução ao Vue router

## Antes da navegação

```
beforeRouteEnter((to, from, next) => {
  getPost(to.params.id, (err, post) => {
    next(vm => vm.setData(err, post))
  })
})

beforeRouteUpdate(async(to, from) =>
  this.post = null
  try {
    this.post = await getPost(to.params.id)
  } catch (error) {
    this.error = error.toString()
  }
})
```

# Introdução ao Vue router

## Após a navegação

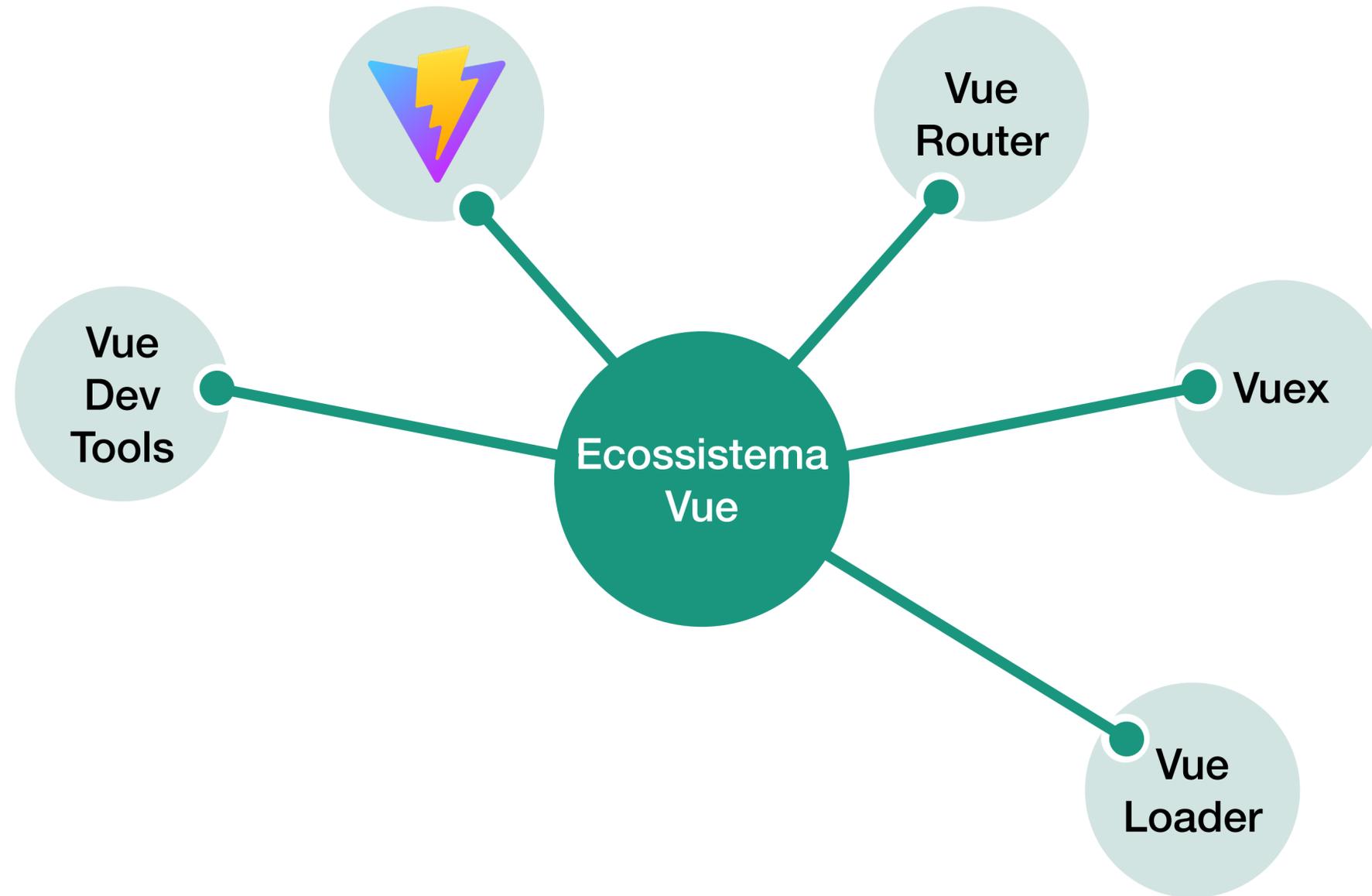
```
<template>
  <div class="post">
    <div v-if="loading" class="loading">Loading...</div>

    <div v-if="error" class="error">{{ error }}</div>

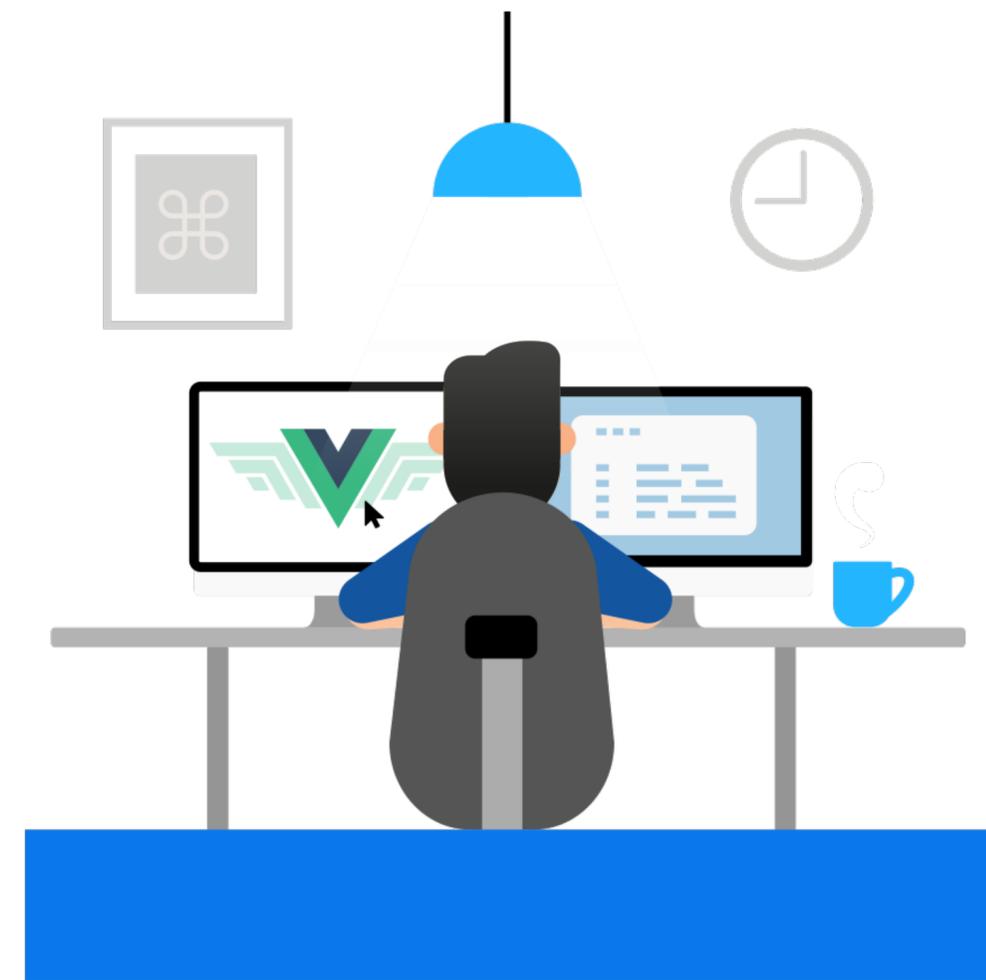
    <div v-if="post" class="content">
      <h2>{{ post.title }}</h2>
      <p>{{ post.body }}</p>
    </div>
  </div>
</template>
```

# Introdução ao VueJS

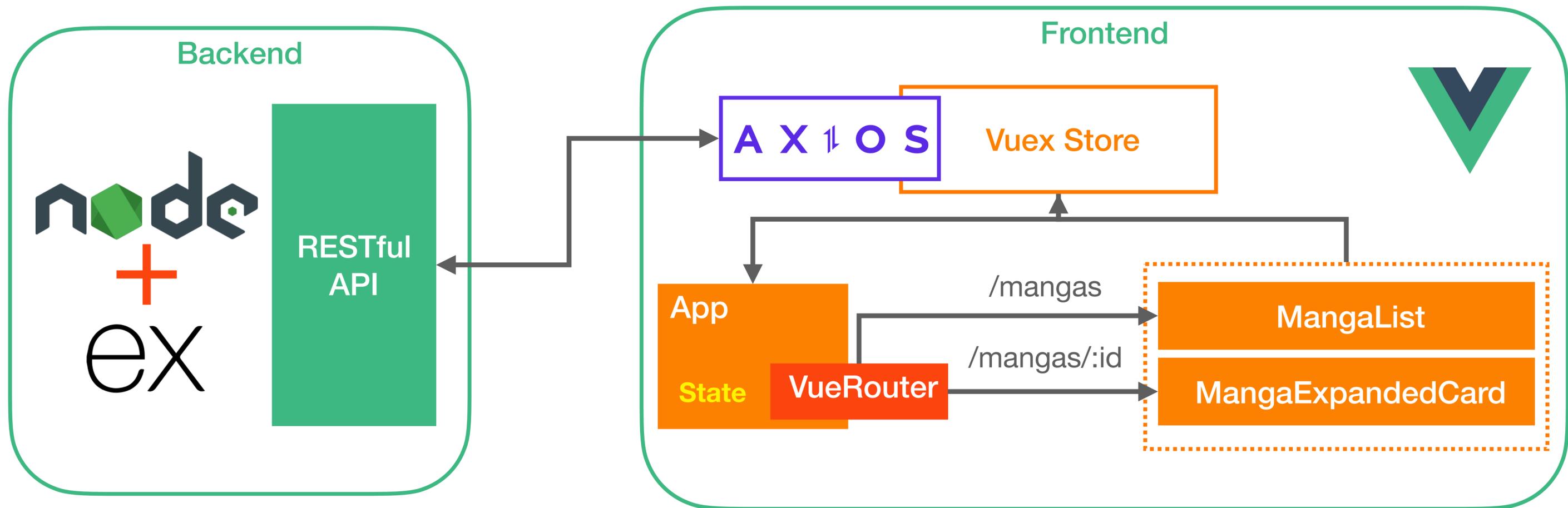
## Ecosystema



# Migrando nossa Manga Store



# Prática



# Referências

- [Why Vue CLI?](#)
- [Jargon-Free Webpack Intro For VueJS Users](#)
- [Introducing Vite: A Better Vue CLI?](#)
- [Has Vite Made Vue CLI Obsolete?](#)
- [Vue 3.2 - Using Composition API with Script Setup](#)
- [WTF is Vuex? A Beginner's Guide To Vuex 4](#)
- <https://next.router.vuejs.org>

Por hoje é só