



UNIVERSIDADE
FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Criando uma app SPA c/ Vue.js

QXD0279 - Desenvolvimento de Software para Web 2

Prof. Bruno Góis Mateus (brunomateus@ufc.br)

Agenda

- Introdução
- Single File Components
- Introdução ao Pinia
- 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



ESLint



Introdução

Webpack

- O problema da dependências em Js
 - O HTML não possui 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

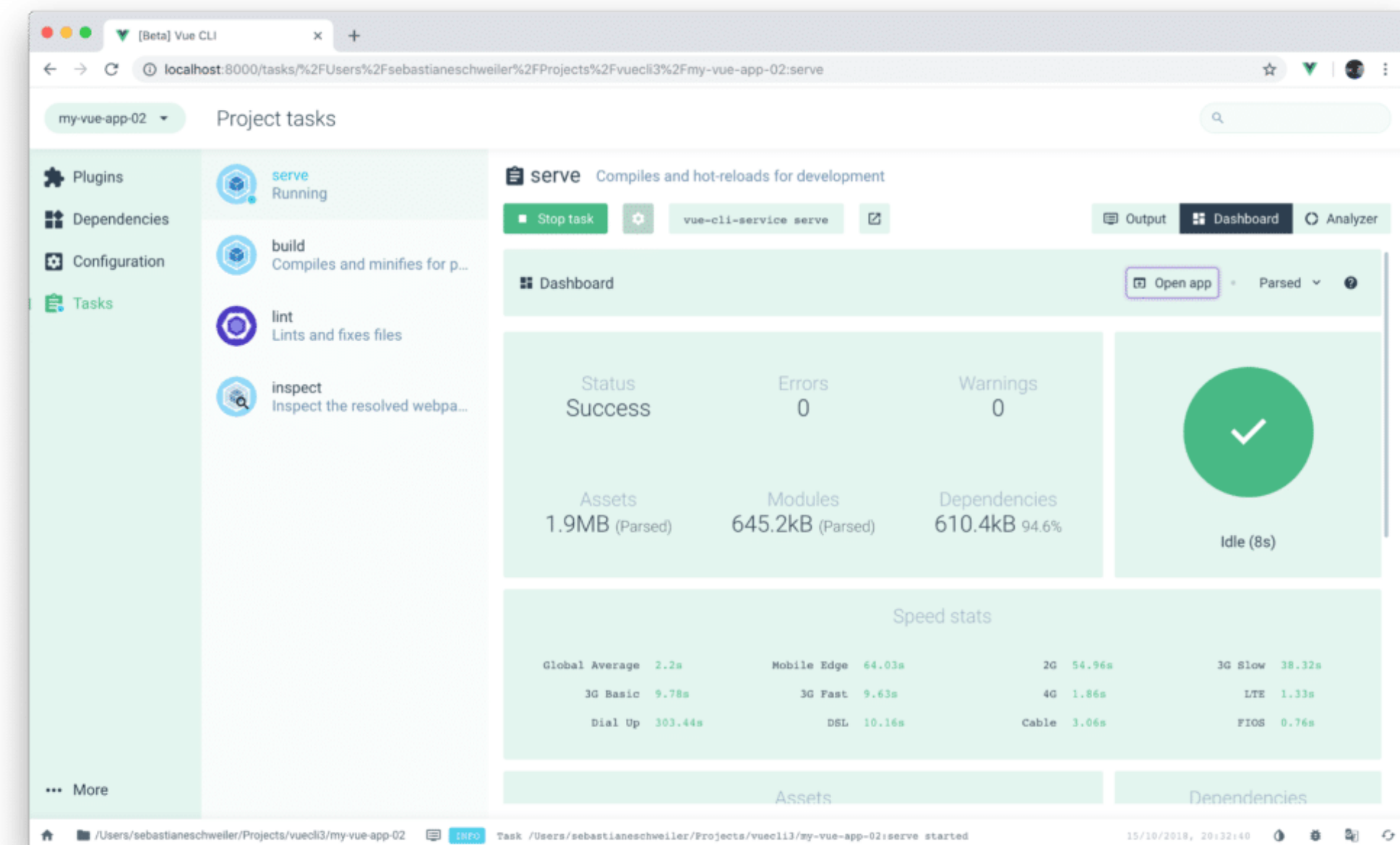
Webpack

- 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

- 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 transpila 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

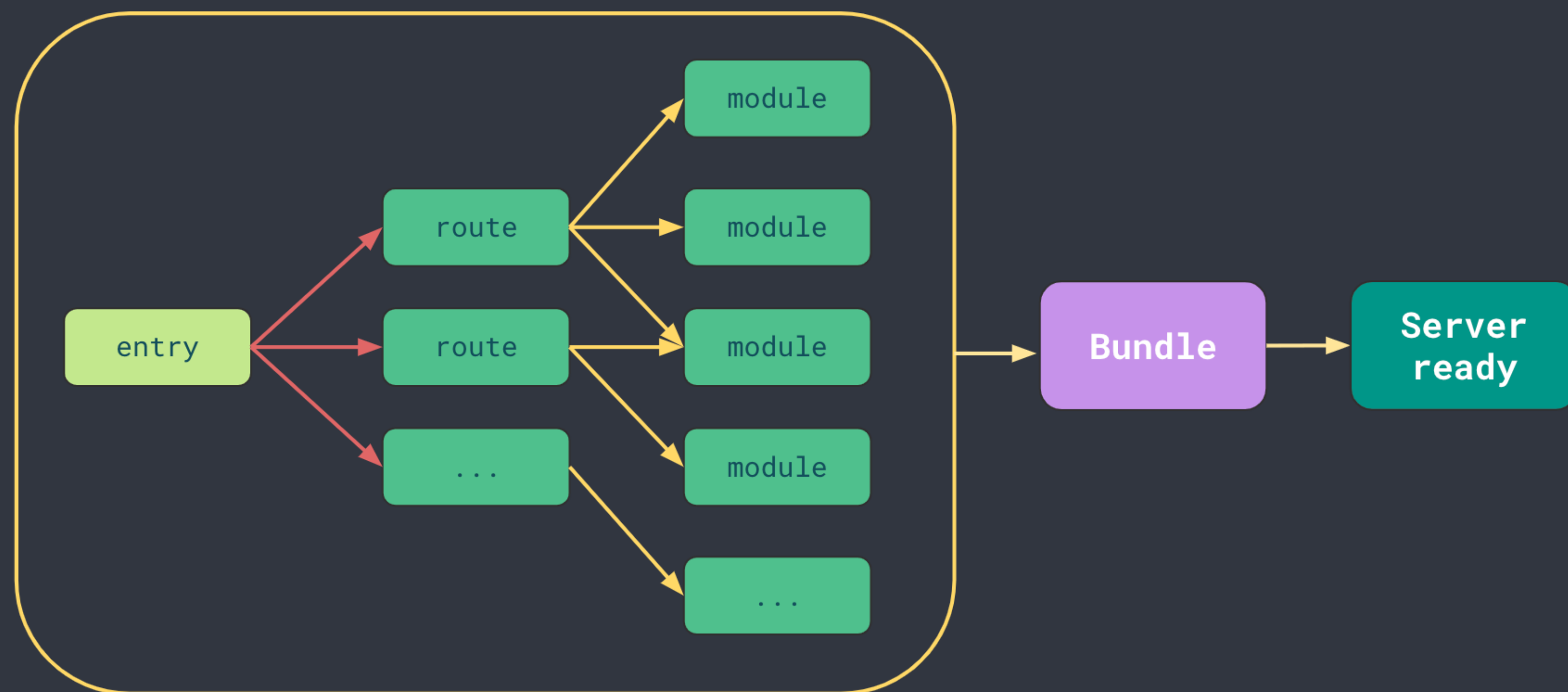
O projeto não é construído durante o desenvolvimento.
O tempo de startup e compilação é reduzido.

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

Introdução

Vite

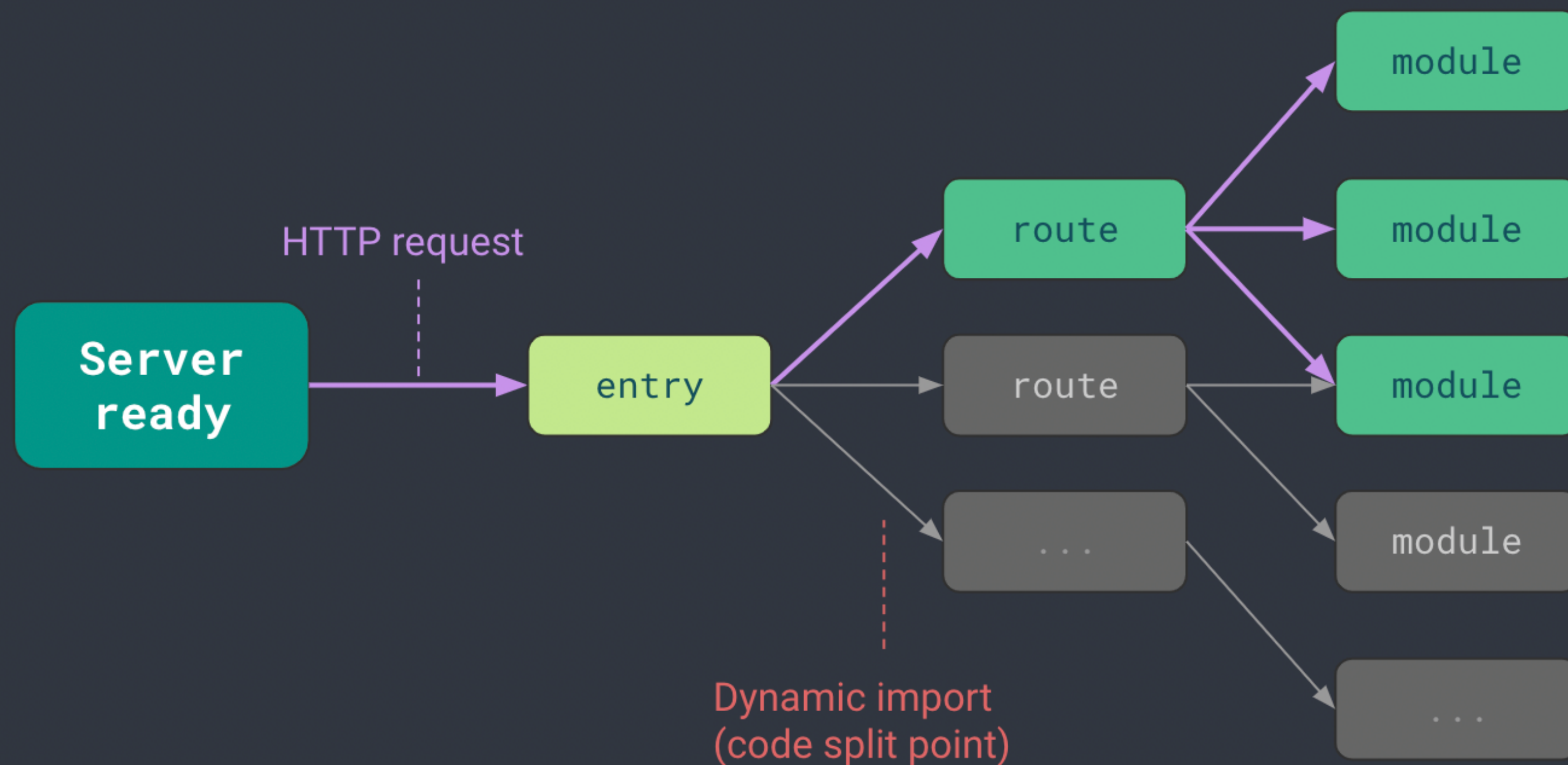
Bundle based dev server



Introdução

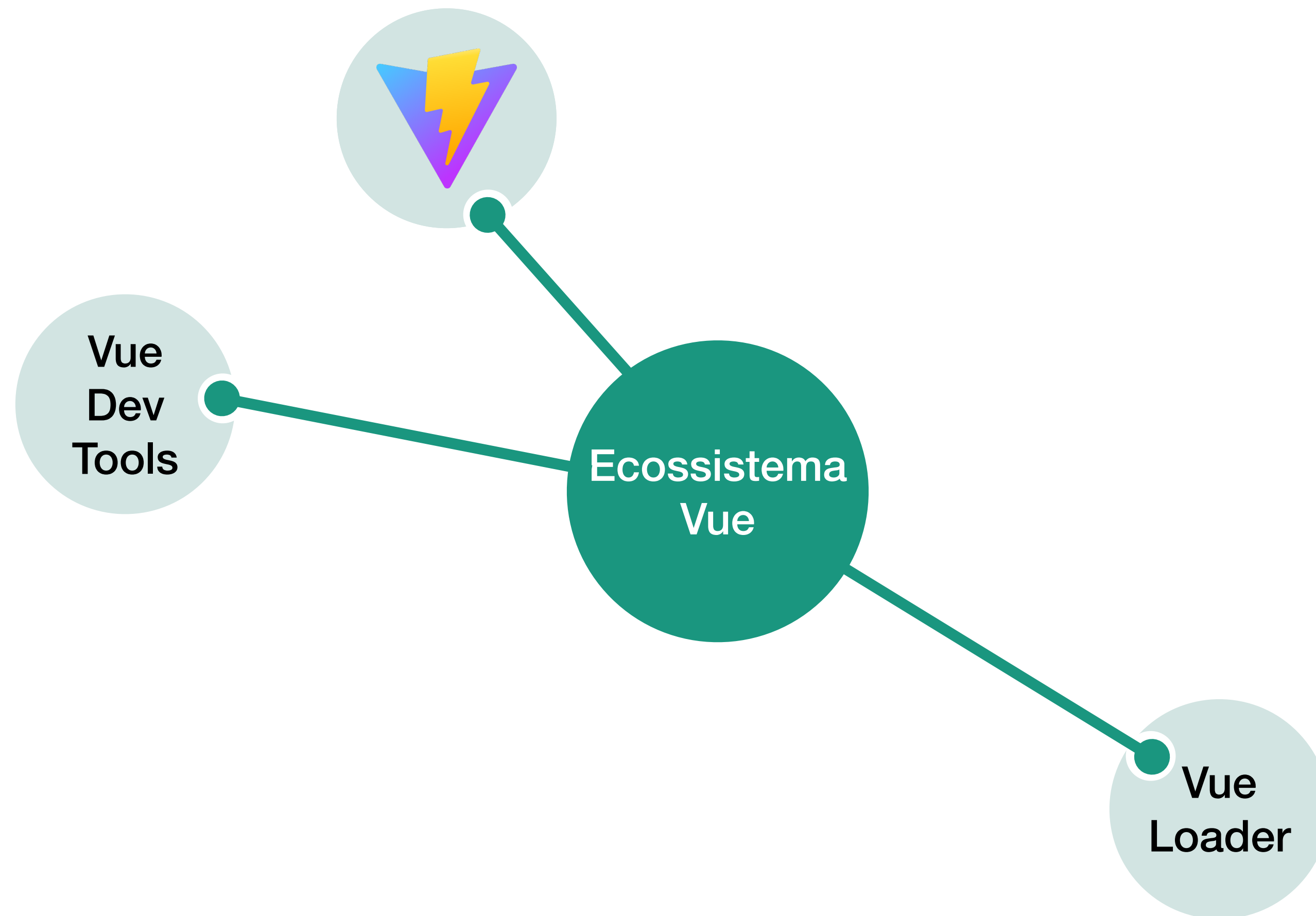
Vite

Native ESM based dev server



Introdução ao VueJS

Ecossistema



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 escalar 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>
import { ref } from 'vue'
export default {
  setup() {
    const msg = ref('Hello world!')
    return { msg }
  }
}
</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>
```

Diagram illustrating the migration from the Option API to the Composition API for a Single File Component.

The Option API (Left) shows a component with a `data` property, a `computed` property, and a `methods` property.

The Composition API (Right) shows the same component using `ref` and `computed` from 'vue' to manage state and logic.

Key differences and mappings:

- `data() { return { name: '' } }` is replaced by `const name = ref('')`.
- `computed: { isNamePresent() { return this.name.length > 0 } }` is replaced by `const isNamePresent = computed(() => name.value.length > 0)`.
- `methods: { submitName() { console.log(this.name) } }` is replaced by a `function submitName() { console.log(name.value) }`.
- The `return` statement in the Composition API returns the reactive variables and methods: `return { name, isNamePresent, submitName }`.

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>
```

Single File Components

- Assim como outros frameworks modernos, Vue permite que os usuários criem componentes isolados em suas aplicações
- São importantes dentre outros motivos por favorecer a **reusabilidade** e a **manutenabilidade**
- São auto-contidos agrupando HTML, JS e CSS
 - Facilitando a manutenção especialmente quando a aplicação escala

Single File Components

Props: Passando dados para um componente filho

- Permite o envio de dados aos components filhos
- São atributos customizáveis registrados por um componente
 - Dever ser explicitamente declarados no componente filho
 - Um valor deve ser dado pelo componente pai/mãe
- São **unidirecionais**, sempre no sentido **pai/mãe -> filho**

Single File Components

Passando dados para um componente filho

```
const app = createApp({
  setup() {
    const posts = ref([
      { id: 1, title: 'My journey with Vue' },
      { id: 2, title: 'Bloggng with Vue' },
      { id: 3, title: 'Why Vue is so fun' }
    ])
    return { posts }
  }
})
```

```
app.component('blog-post', {
  props: ['title'],
  template: `<h4>{{ title }}</h4>`
})
```

```
<div>
  <blog-post
    v-for="post in posts"
    :key="post.id"
    :title="post.title"
  ></blog-post>
</div>
```

within them. This allows for easier
s grow much larger in scale.

lication. As a result, we'll break apart the
ities:

My journey with Vue

Bloggng with Vue

Why Vue is so fun

•

Single File Components

defineModel - Vue 3.4

- Em algumas situações precisamos atualizar o componente pai, a cada atualização de um prop do componente filho
- A partir do Vue 3.4, a maneira recomendada é a seguinte:

```
<script setup>
const model = defineModel()

function update() {
  model.value++
}
</script>

<template>
  <div>Parent bound v-model is: {{ model }}</div>
  <button @click="update">Increment</button>
</template>
```

```
<!-- Parent.vue -->
<Child v-model="countModel" />
```

Single File Components

defineModel - Vue 3.4

- É possível adicionar argumentos ao v-model associado ao defineModel

```
<script setup>
const title = defineModel('title')
</script>

<template>
  <input type="text" v-model="title" />
</template>
```

```
<!-- Parent.vue -->
<Child v-model:title="bookTitle" />
```

Single File Components

Escutando eventos do componentes filhos

- Em diversas situações é necessário que haja comunicação entre filho e pai
- Essa comunicação é feita por meio de eventos customizados
 - São iniciados quando um componente executa a instrução `$emit('nome do evento')`
 - Um componente que está escutando pelo é evento é notificado na instrução `$on('nome do evento')`
 - Dados podem ser enviados

Single File Components

Escutando eventos do componentes filhos

```
const app = createApp({
  setup() {
    const posts = ref([
      { id: 1, title: 'My journey with Vue' },
      { id: 2, title: 'Blogging with Vue' },
      { id: 3, title: 'Why Vue is so fun' }
    ])
    const postFontSize = ref(1)
    return { posts, fontSize }
  }
})
```

```
app.component('blog-post', {
  props: ['title'],
  emits: ['enlargeText'],
  template: `
    <div class="blog-post">
      <h4>{{ title }}</h4>
      <button @click="$emit('enlargeText')">
        Enlarge text
      </button>
    </div>
  `
})
```

```
<div :style="{ fontSize: fontSize + 'em' }">
  <blog-post v-for="post in posts"
    :key="post.id"
    :title="post.title"
    @enlarge-text="fontSize += 0.1"
  ></blog-post>
</div>
```

Single File Components

Escutando eventos do componentes filhos

```
const app = createApp({
  setup() {
    const posts = ref([
      { id: 1, title: 'My journey with Vue' },
      { id: 2, title: 'Blogging with Vue' },
      { id: 3, title: 'Why Vue is so fun' }
    ])
    const fontSize = ref(1.0)
    function onEnlargeText(enlargeAmount) {
      fontSize.value += Number(enlargeAmount)
    }
    return { posts, fontSize, onEnlargeText }
  }
})
```

```
app.component('blog-post', {
  props: ['title'],
  emits: ['enlargeText'],
  template: `
    <div class="blog-post">
      <h4>{{ title }}</h4>
      <button @click="$emit('enlargeText', 0.1)">
        Enlarge text
      </button>
    </div>
  `
})
```

```
<div :style="{ fontSize: fontSize + 'em' }">
  <blog-post v-for="post in posts"
    :key="post.id"
    :title="post.title"
    @enlarge-text="onEnlargeText"
  ></blog-post>
</div>
```

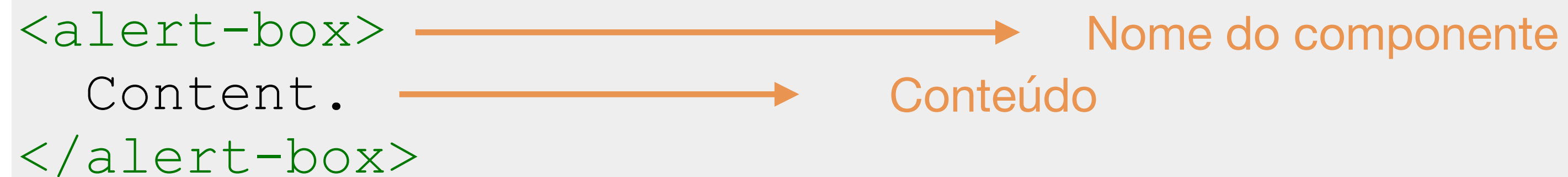
1. Ao clicar no botão o evento **enlargeText** é emitido com o valor **0.1** como argumento
2. O evento é tratado no blog-post
 - O valor **0.1**, é o **valor do parâmetro** recebido
3. o método **onEnlargeText** do **componente pai** é invocado
4. A propriedade do componente pai é atualizada
5. O style do div é atualizado como consequência

Valor enviado

Single File Components

Slots

- Assim com elemento HTML, algumas vezes é útil passar o conteúdo para o componente da seguinte forma



The diagram shows a code snippet for a single file component. The first line is `<alert-box>`, with an orange arrow pointing from it to the text "Nome do componente". The second line is `Content.`, with an orange arrow pointing from it to the text "Conteúdo". The third line is `</alert-box>`.

```
<alert-box>
  Content.
</alert-box>
```

- Isto pode ser feito ao utilizar o elemento `<slot>`

Single File Components

Distribuição de componentes com slots

```
app.component('todo-button', {  
  template: `  
    <button class="btn-primary">  
      <slot></slot>  
    </button>  
  `,  
})
```

```
<todo-button>  
  Add todo  
</todo-button>
```

```
<todo-button>  
  <i class="fas fa-plus"></i>  
  Add todo  
</todo-button>
```

```
<todo-button>  
  <font-awesome-icon name="plus"></font-  
  awesome-icon>  
  Add todo  
</todo-button>
```

Outro componente

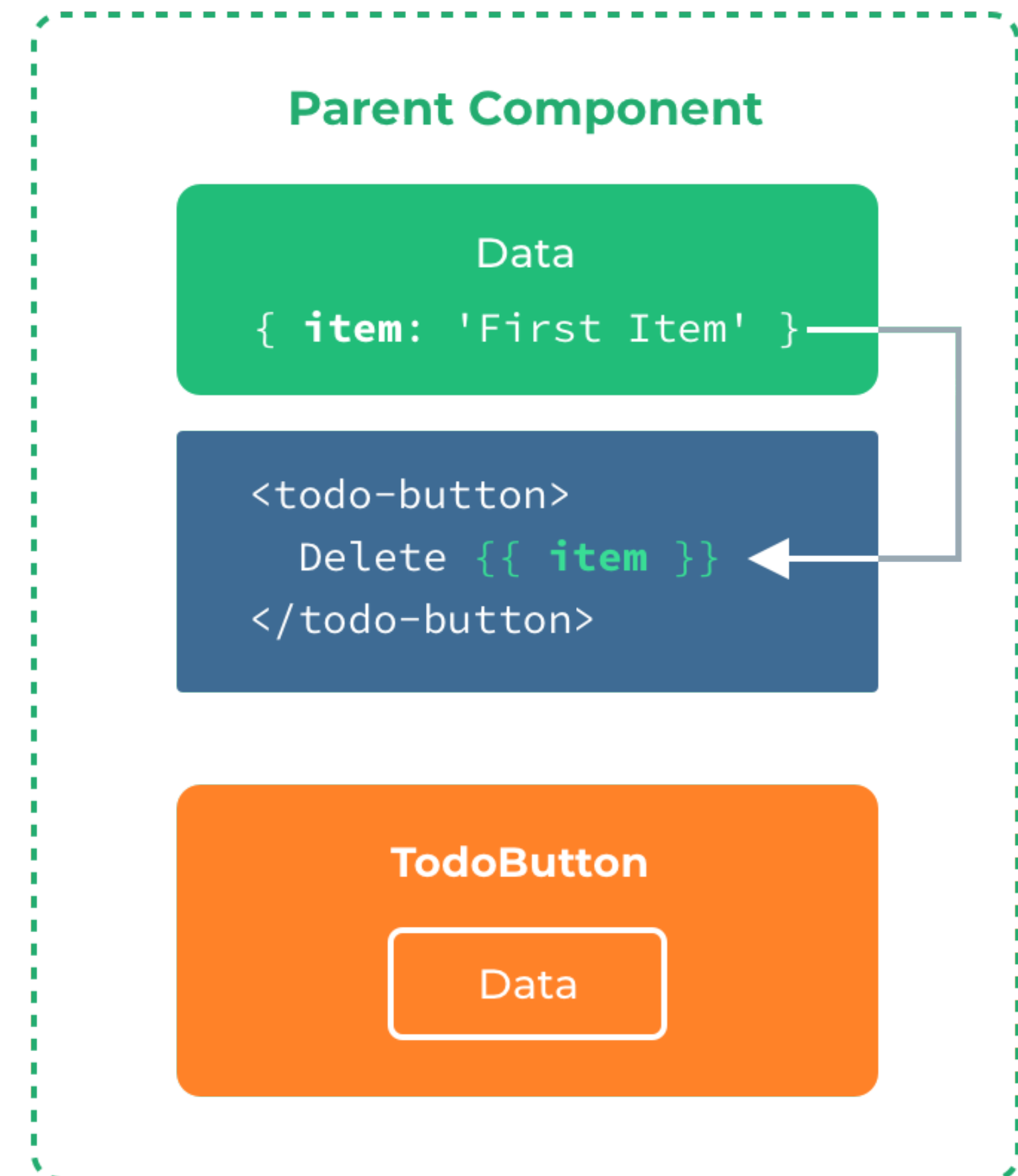
Single File Components

Distribuição de componentes com slots

```
app.component('todo-button', {  
  template: `  
    <button class="btn-primary">  
      <slot></slot>  
    </button>`  
})
```

```
<todo-button>  
  Delete a {{ item }}  
</todo-button>
```

Slots não tem acesso a dados do componente filho



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*)

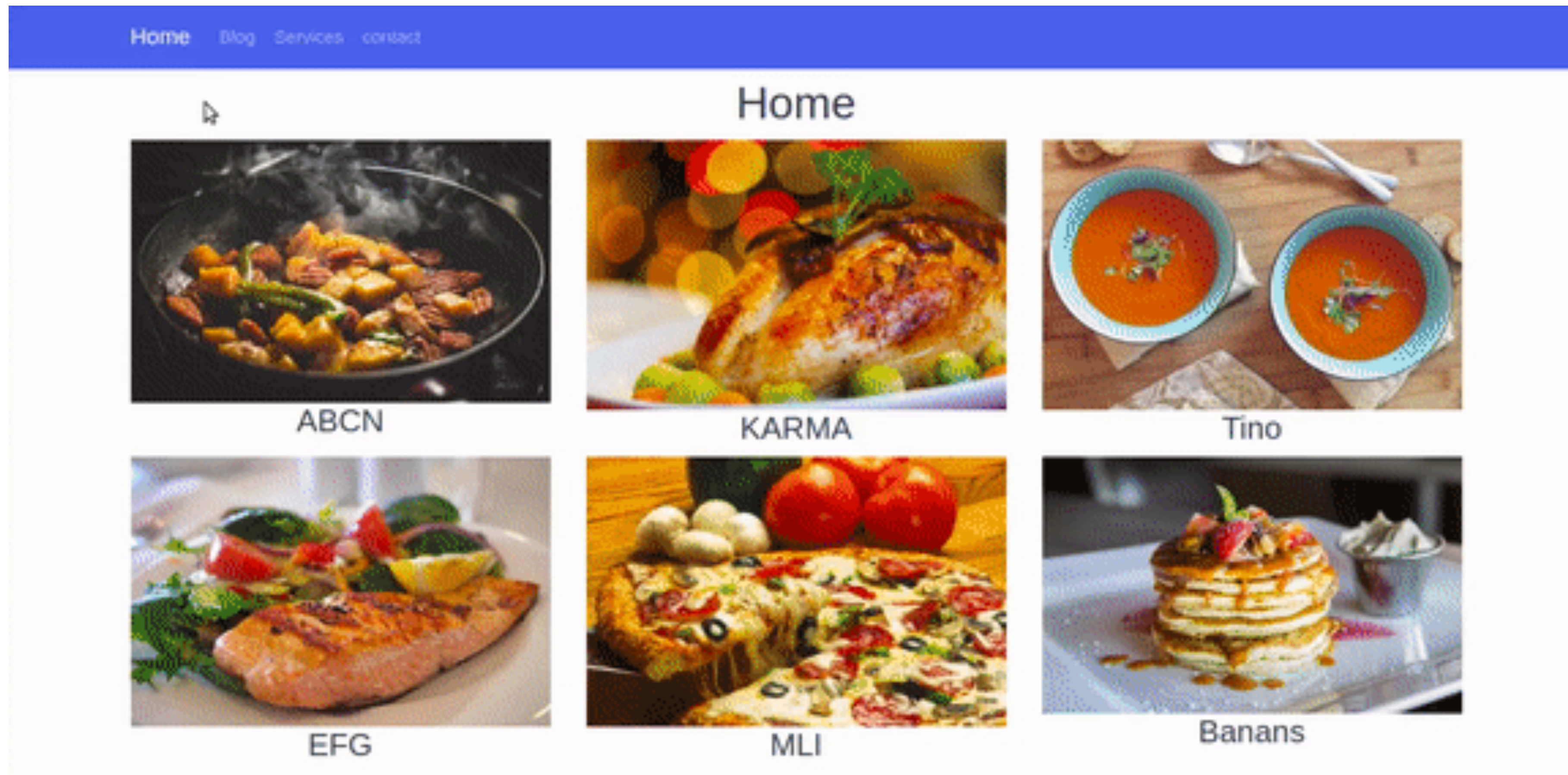
Introdução ao Vue router

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>
```

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

Local onde o componente associado a rota será renderizado

```
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)

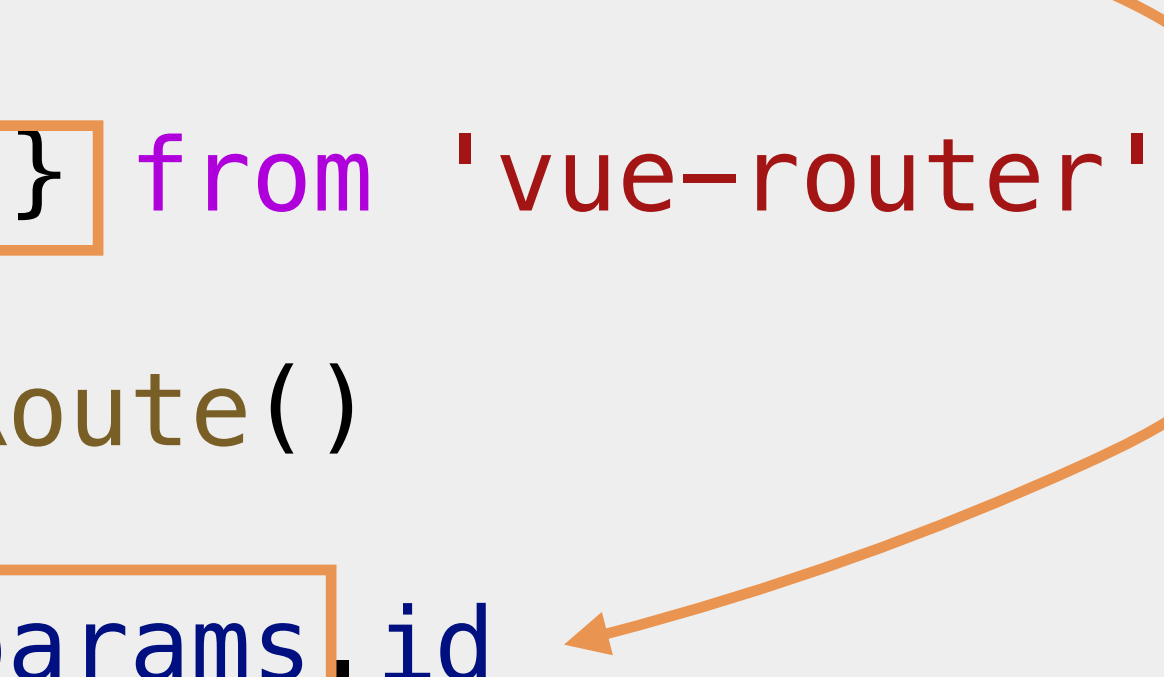
app.mount('#app')
```

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>
```



Introdução ao Vue router


Roteamento dinâmico

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

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' }]
```

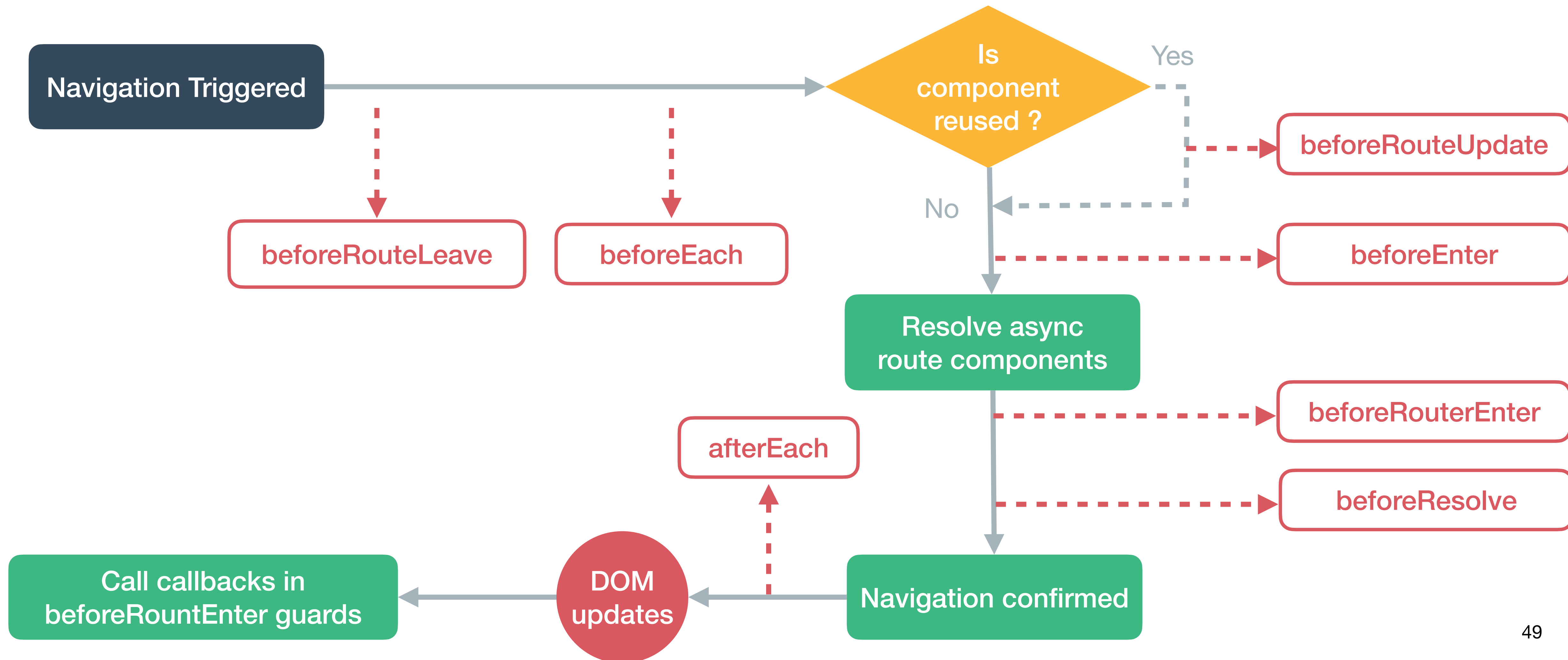
Introdução ao Vue router

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'  
})
```

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>

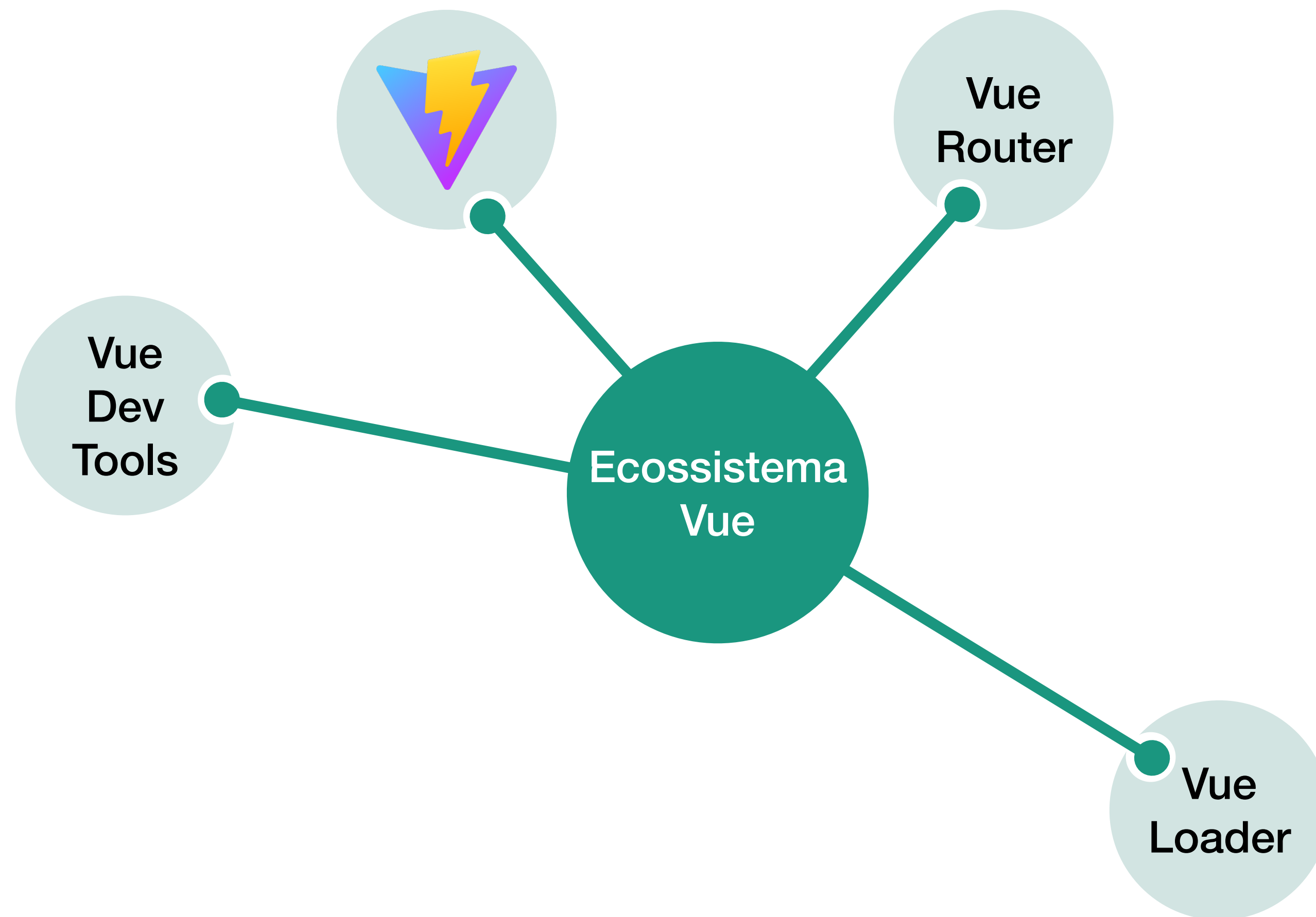
    <div v-if="error" class="error">{{ error }}</div>

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

- Os dados são recuperados antes (`beforeRouteEnter`)
- A navegação é realizada

Introdução ao Vue router

Ecossistema



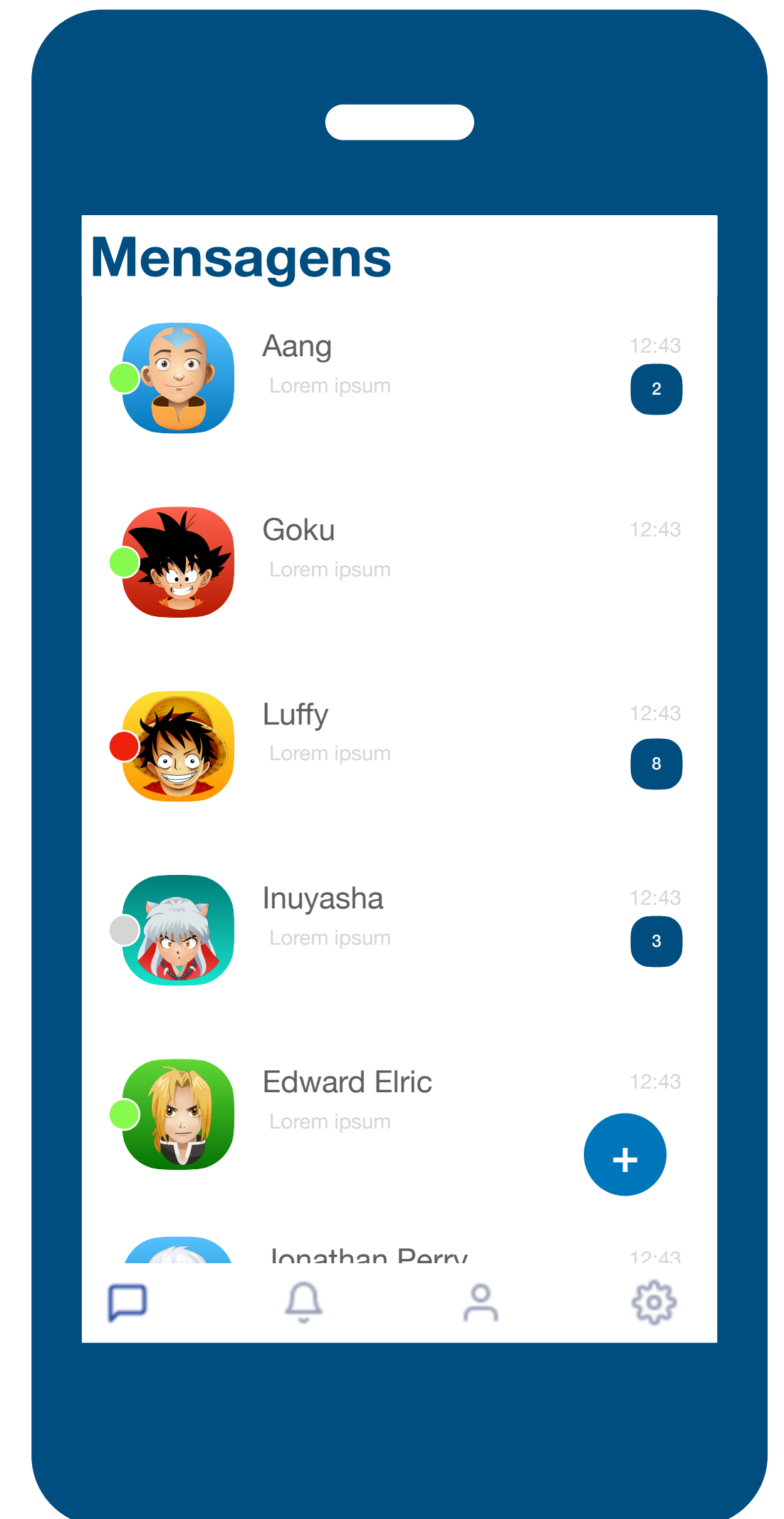
Introdução ao Pinia



Introdução ao Pinia

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



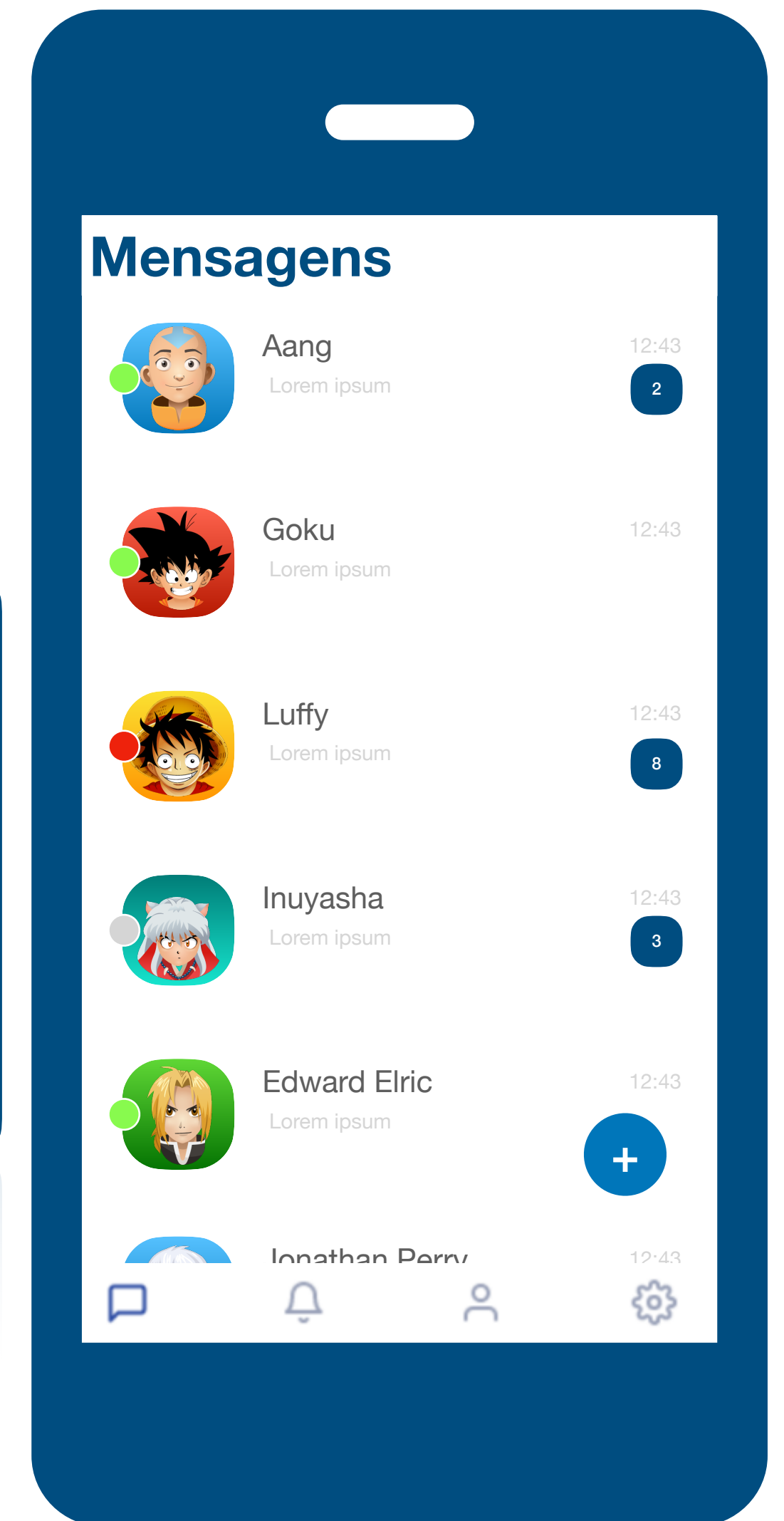
Introdução ao Pinia

Motivação

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

“Quando múltiplos componentes de uma aplicação compartilham os mesmos dados, a complexidade das interconexões irão aumentar até que não seja mais possível prever ou entender o estado dos dados. Consequentemente, 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 Pinia

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 Pinia

Princípios do FLUX - 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 Pinia

Princípios do FLUX - 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 Pinia

Princípios do FLUX - Mutations are synchronous

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

Introdução ao Pinia

Pinia

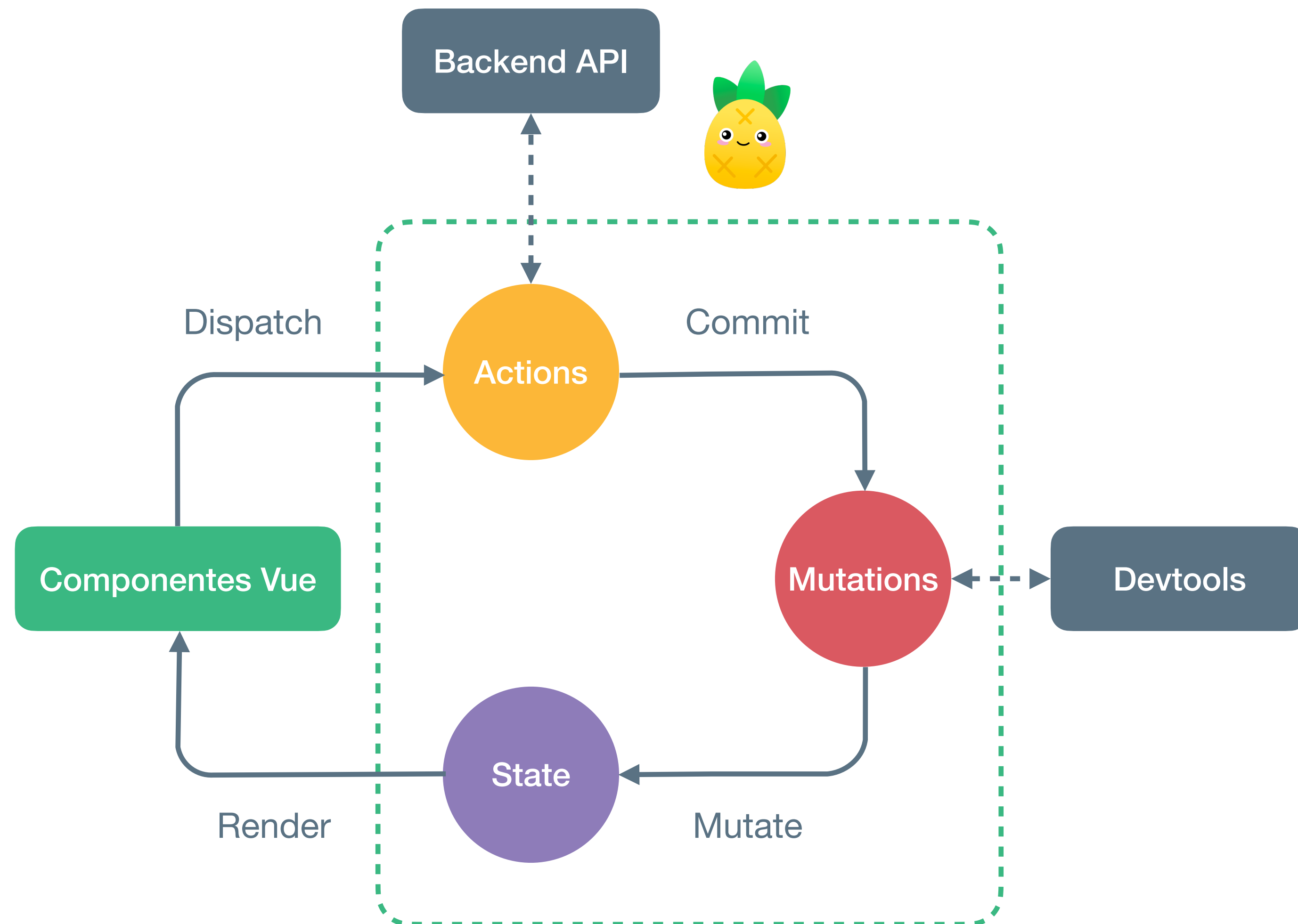
- Biblioteca que facilita a implementação da arquitetura Flux
 - State Management Pattern + library
- Armazena os dados de forma centralizada garantindo que os estados só podem ser mudados de uma forma previsível
- Iniciou como um experimento de *redesign* do Vuex 5 usando a composition API
- Prover uma API **mais simples comparada com o Vuex**
- Baseada em três conceitos principais: **state**, **getters** e **actions**

Introdução ao Pinia

Vantagens de usar o Pinia

- Devtools support
 - Rastrei ações e mutações
 - Viagem no tempo e debug facilitado
- Hot module replacement
 - É possível modificar as stores sem recarregar a página
- Plugins
- Suporte a TypeScript e autocompletion em JS
- Server Side Rendering suport

Introdução ao Pinia



Introdução ao Pinia

Core concepts - Store

- É uma entidade que armazena o estado e as lógica de negócios que não estão ligadas com a árvore de componentes
 - Armazena o estado global da aplicação
 - Podemos tratá-la como um componente que está sempre presente

Introdução ao Pinia

Quando usar Stores?

- Stores devem conter **informações** que devem ser **acessadas em toda parte da aplicação**
 - Dados usados em vários locais (Ex: Informações do usuário “logado”)
 - Dados que precisam ser preservados independente da navegação
- **Deve se evitar o armazenamento de dados que poderiam estar em um componente**
 - A visibilidade de um elemento do componente

Introdução ao Pinia

Core concepts - State

- É parte central das *stores*
- Pinia permite o uso de várias stores independentes (*single state tree*)
 - *Single source of truth*
 - Evita o compartilhamento dos dados com todos os componentes

Introdução ao Pinia


Criando uma Store

```
import { createPinia } from 'pinia'

app.use(createPinia())
```

```
import { defineStore } from 'pinia'
export const useCounterStore = defineStore('counter', () => {
  const count = ref(0)
  return { count }
})
```

```
<script setup>
import { counterStore } from '@/stores/counter'
const myCounter = useCounterStore()
myCounter.count++
</script>
```



Introdução ao Pinia

Core concepts - Getters

- Algumas vezes precisamos de um estado derivado do estado da store
- São o equivalente ao *computed values* só que aplicados a *states*
- Precisam ser *síncronos*

Introdução ao Pinia

Core concepts - Getters

```
import { ref, computed } from 'vue'
export const taskStore = defineStore('main', () => {
  const todos = ref([
    { id: 1, text: '...', done: true },
    { id: 2, text: '...', done: false }
  ])

  const doneTasks = computed(() =>
    todos.value.filter(todo => todo.done))

  const doneTasksCount = computed(() =>
    doneTodos.value.length)

  return { doneTasks, doneTasksCount }
})
```

```
<script setup>
  const store = taskStore()
</script>

<template>
  <p>
    #Done {{ store.doneTasksCount }}
  </p>
</template>
```

Introdução ao Pinia

Core concepts - Actions

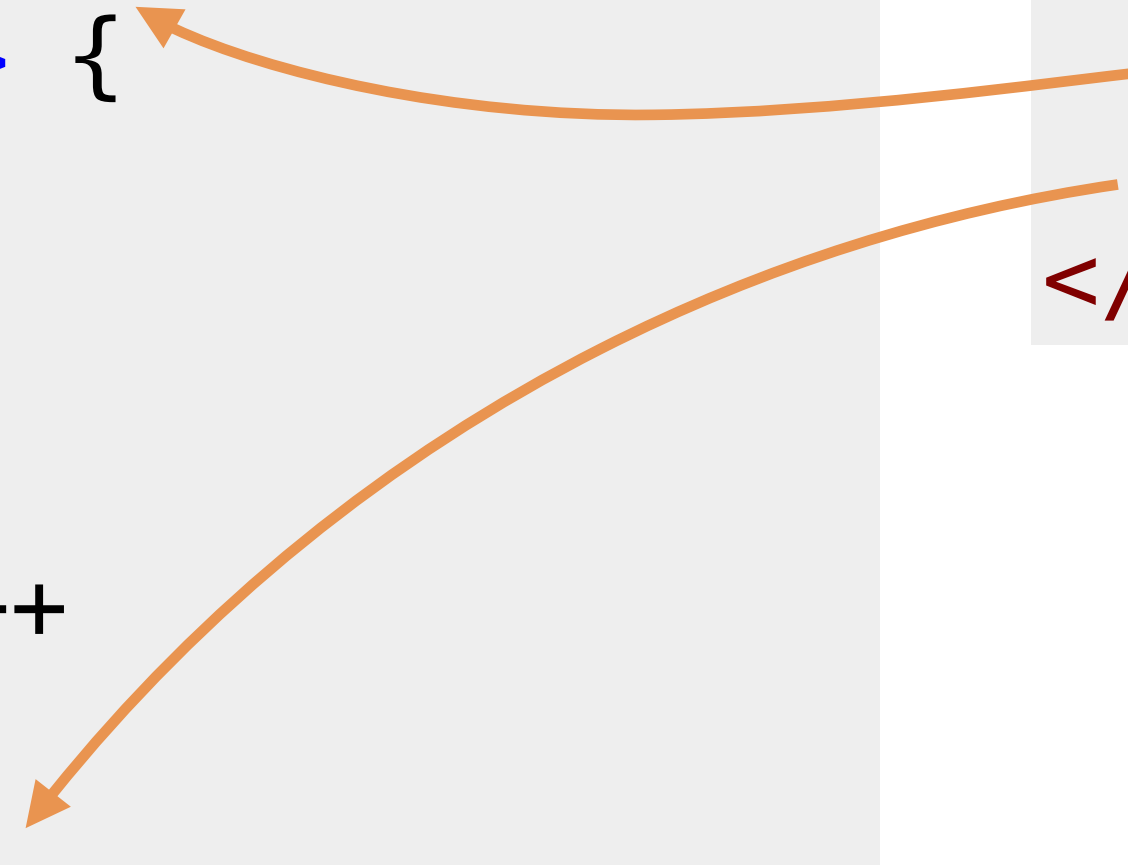
- Ações são o equivalente aos métodos porém aplicados em *stores*
- Diferentemente de *getters*, podem ser assíncronas
- Perfeitas para definir lógicas de negócios

Introdução ao Pinia

Core concepts - Action

```
export const useCounterStore =  
defineStore('main', () => {  
  const counter = ref(0)  
  
  function increment() {  
    return counter.value++  
  }  
  
  function randomizeCounter() {  
    counter.value = Math.round(100 *  
Math.random())  
  }  
  
  return { randomizeCounter }  
})
```

```
<script setup>  
  const store = useStore()  
  store = store.randomizeCounter()  
</script>
```



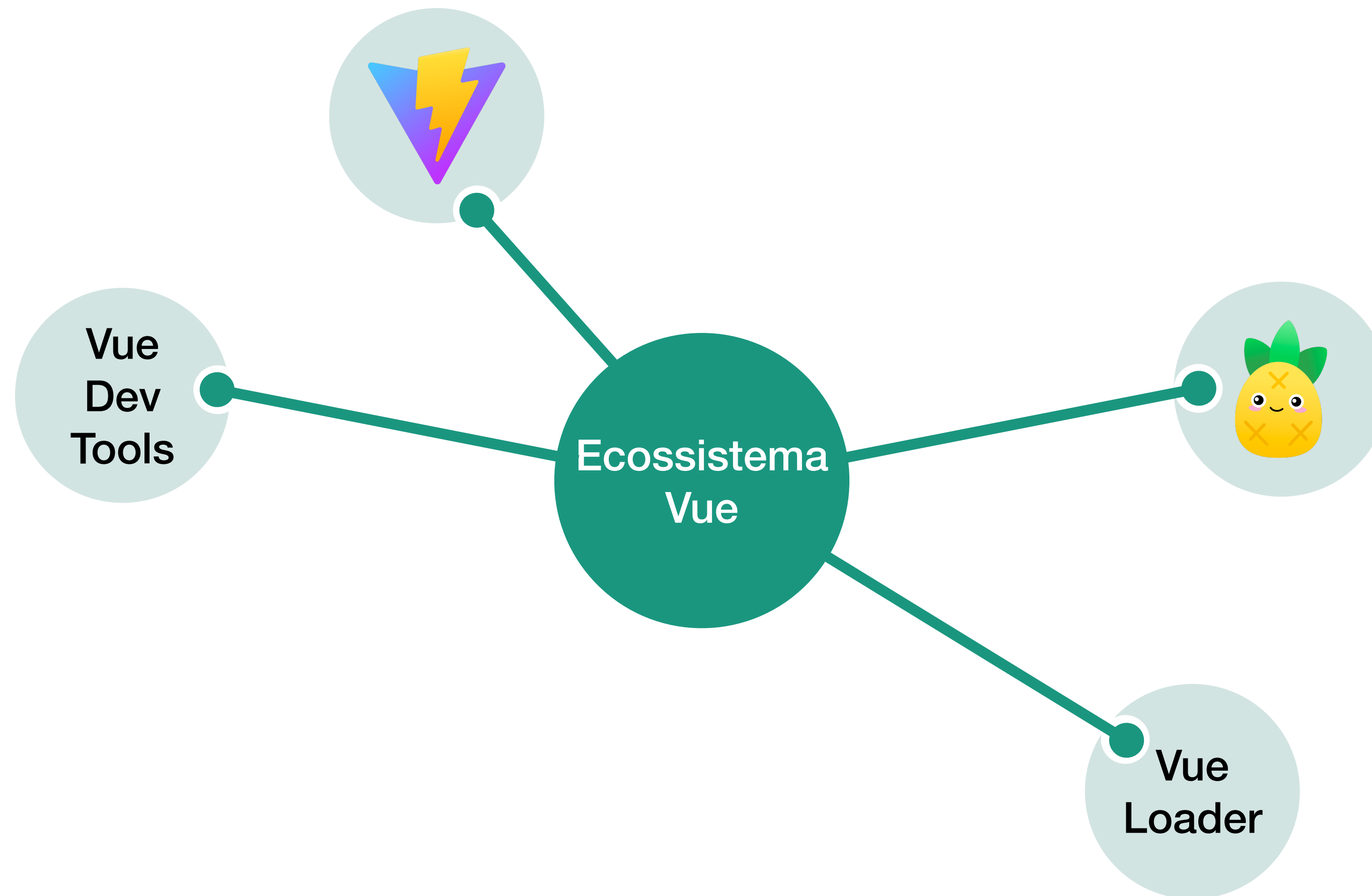
Introdução ao Pinia

Estrutura de uma aplicação

```
|— index.html
|— main.js
|— api
|   |— ... # abstractions for making API requests
|— components
|   |— App.vue
|   |— ...
|— stores
|   |— user.js
|   |— main.js
```

Introdução ao Pinia

Ecossistema



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](#)
- [Complex Vue 3 state management made easy with Pinia](#)
- <https://next.router.vuejs.org>

Por hoje é só