



UNIVERSIDADE  
FEDERAL DO CEARÁ  
CAMPUS QUIXADÁ

# Criando uma app SPA c/ Vue.js e Pinia

QXD0193 - Projeto de Interfaces Web

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

# Agenda

- Introdução
- Pinia

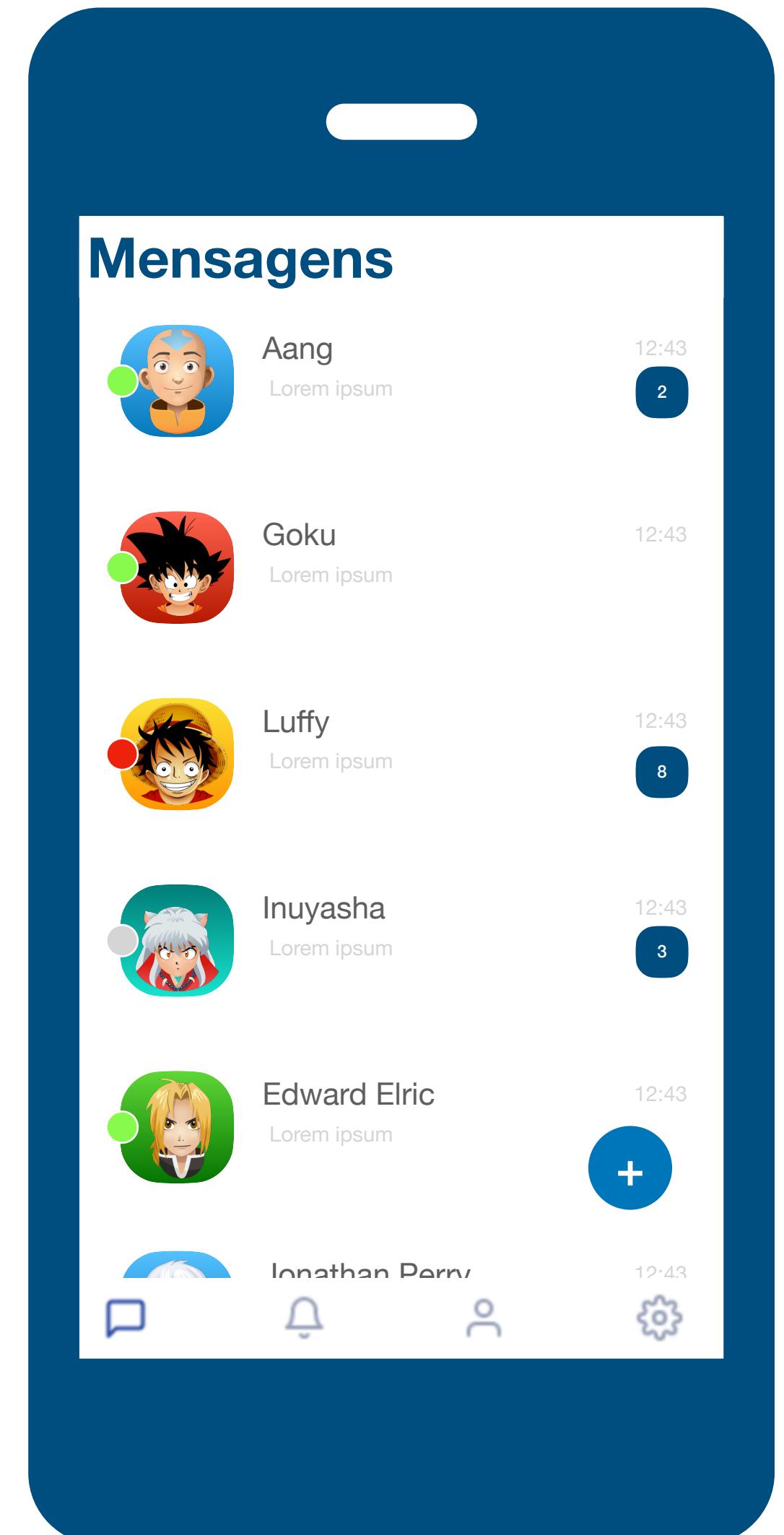
# Introdução



# 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



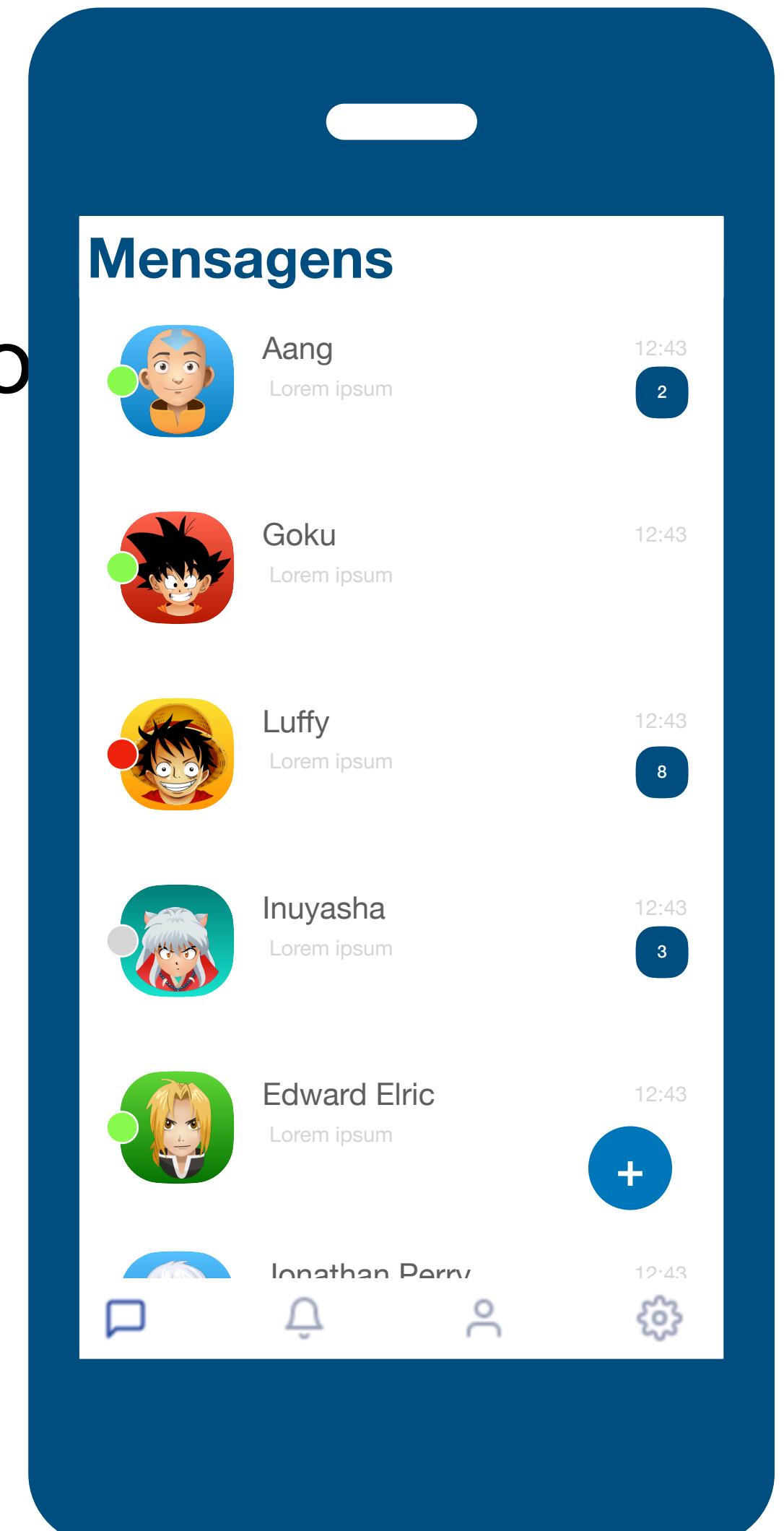
# Introdução ao Pinia

## Motivação

- A situação anterior “zombie notification” foi enfrentada pelos 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.”

- Sobre a solução se tornar 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 um sequência e do tempo de execução de eventos imprevisíveis

# Pinia



# Introdução ao Pinia

## Pinia

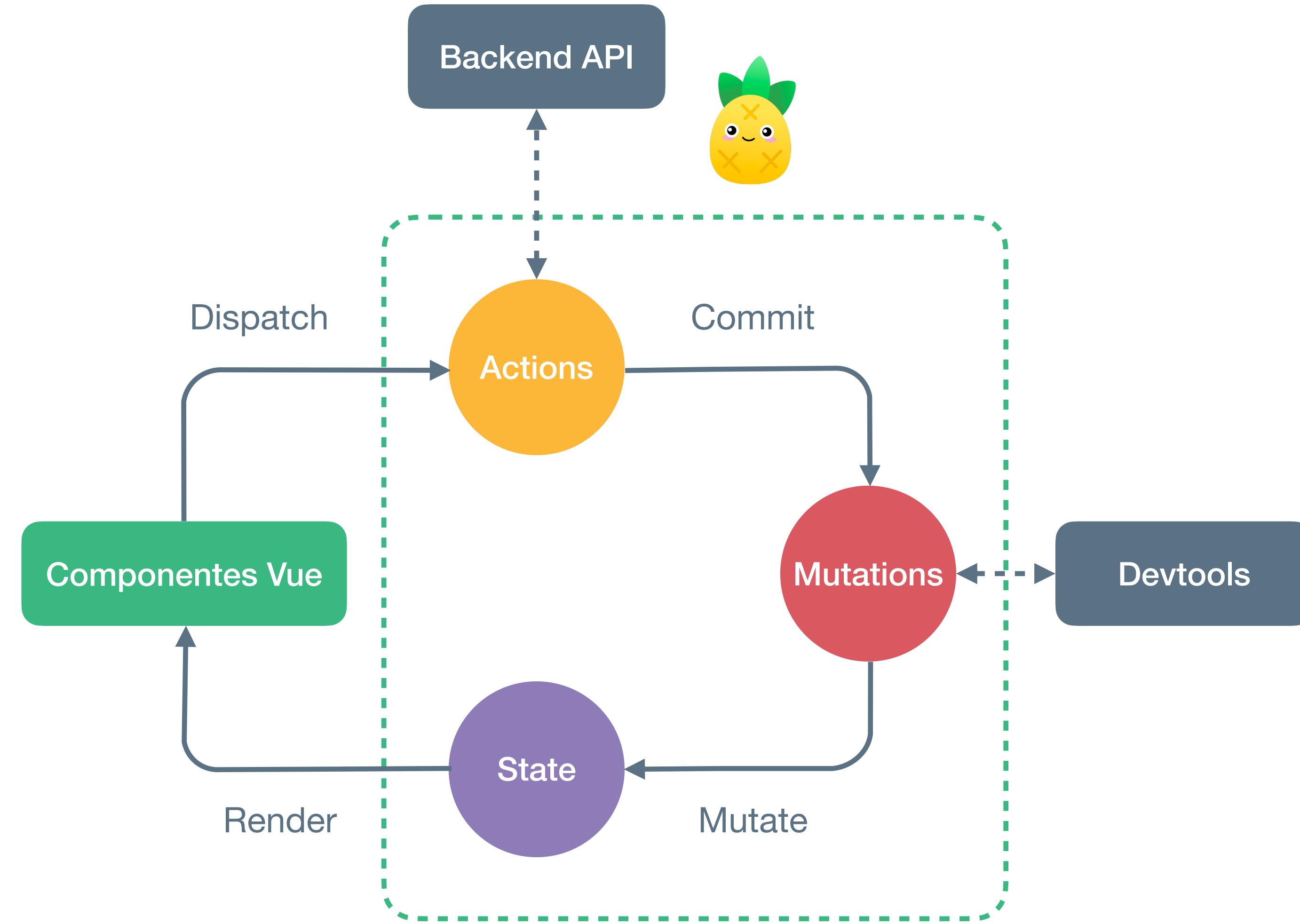
- Biblioteca que facilita a implementação da arquitetura Flux
  - State Management Pattern + library
  - Armazena os dados de forma centralizada garantindo que os estados só podem ser mudados de uma forma previsível
  - Iniciou como um experimento de *redesign* do Vuex 5 usando a composition API
  -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

# Introdução ao Pinia

## Vantages 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>
```

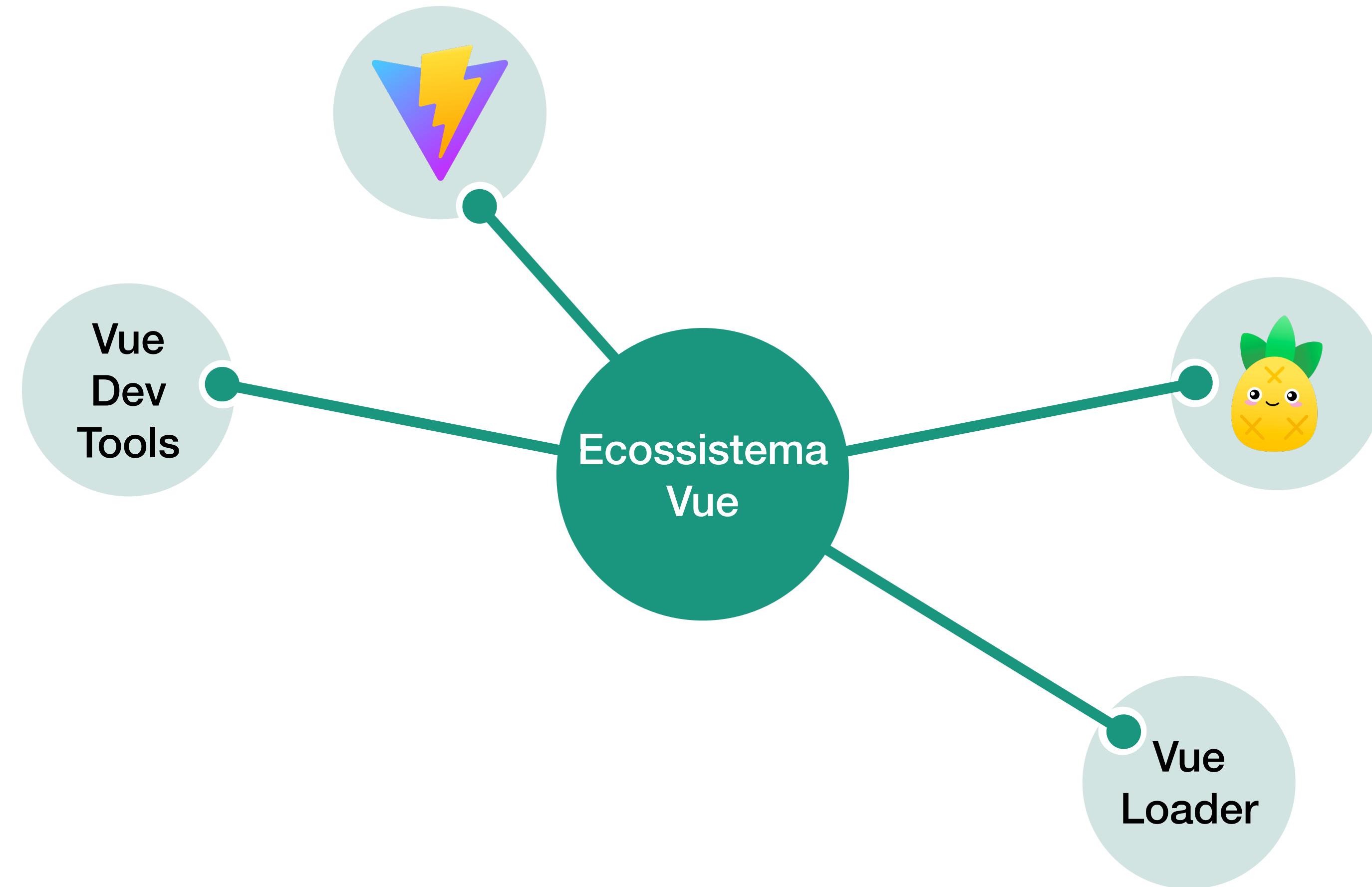
# Introdução ao Pinia

## 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