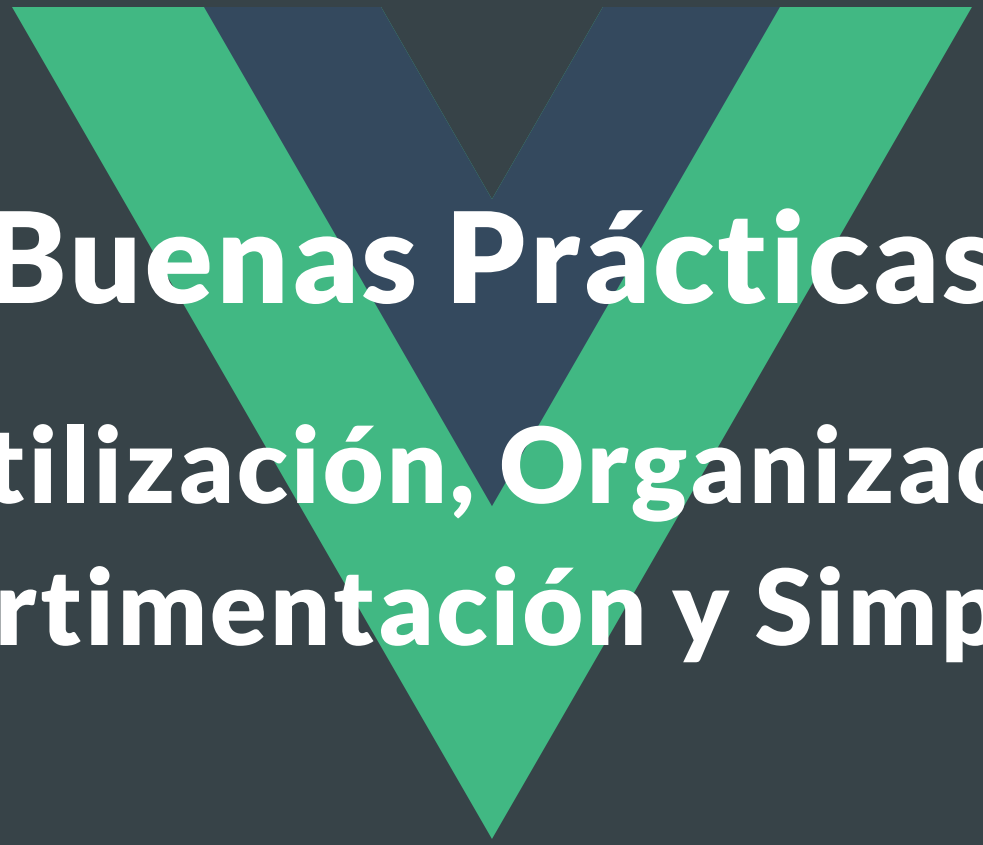

Buenas Prácticas

Reutilización, Organización, Compartimentación y Simplicidad

Introducción

- Vue destaca por: Simplicidad y pocas restricciones
- Puede parecer limitado para grandes proyectos
- Si no se siguen buenas prácticas se puede llegar al caos

Buenas prácticas

- Uso de Slots
- Componentes independientes. Principios F.I.R.S.T.
- Modularización del Store
- Testing

Slots

Uso de Slots

- Ideales en situaciones en las que tenemos un gran nº de componentes hijo para un padre.
- Evitamos tener que emitir un gran nº de eventos desde los componentes hijo al componente padre
- Slot: Fragmento de componente al cual se le puede inyectar contenido
- Permite el uso de los componentes de Vue de la misma forma cómo funciona un elemento de bloque en HTML, permitiendo anidar otros elem dentro de él.

Ejemplo: ContenedorDemo

```
<div class="container">
    <h2>Vue slots</h2>
    <slot></slot>
</div>
```

- Cuando se utilice nuestro componente, las etiquetas <slot></slot> se sustituirán por lo contenido dentro de las etiquetas del componente.

Ejemplo: ContenedorDemo

```
<contenedor-demo>
    <p>Este contenido se inyectará en el slot</p>
    <otro-componente></otro-componente>
</contenedor-demo>
```

- Resultado:

```
<div class="container">
    <h2>Vue slots</h2>
    <p>Este contenido se inyectará en el slot</p>
    /** ... **/
</div>
```

Componentes Independientes

Principios F.I.R.S.T.

Principios F.I.R.S.T.

- Principios que se han de seguir a la hora de crear y compartir componentes en nuestra aplicación Vue.
- Principios definidos por Addy Osmani

> Keep it (**F**)ocused.
> Keep it (**I**)ndependent.
> Keep it (**R**)eusable.
> Keep it (**S**)mall.
> Keep it (**T**)estable.
> or in short, **FIRST**.

- **F**ocused
  - Conciso. Resuelve 1 problema
- **I**ndependent
  - Independiente. Desacoplado
- **R**eusable
  - Reutilizable
- **S**mall
  - Pequeño. Simple
- **T**estable

Modularización del Store

Store bien organizada

El Store

- Aunque existen alternativas (Pinia), nos centramos en VueX
- Vuex es el *state management pattern* del ecosistema Vue
- Vuex permite centralizar la información de nuestra aplicación en una store con la cual pueden interactuar el resto de componentes.
- Funciona de manera reactiva
- No necesario en pequeñas apps, casi obligatorio en grandes proyectos.
- Nos permite ahorrar complejos "caminos" de eventos para propagar cambios

Componentes del Store

- 4 componentes principales
  - State: Se utiliza para mantener la info de la app
  - Getters: Utilizados para acceder a la info desde fuera de la store
  - Mutations: Utilizados para modificar el state
  - Actions: Utilizados para hacer commit de las mutaciones

Organización del Store

- Principios de organización:
  1. Mantener un estado a nivel de aplicación centralizado en la store
  2. Los estados siempre cambian cometiendo mutaciones
  3. La lógica asíncrona se encapsula y se usa con acciones
- Siguiendo estos principios nuestros proyectos siempre estarán bien estructurados y serán fácilmente escalables.

Modularización del VueX

- Al trabajar con VueX en proyectos extensos, a largo plazo, se premia la modularización del mismo.
- En vez de tener una estructura en la que todo el store se encuentre unificado:

```
├── index.html
├── main.js
├── ...
└── store/
    ├── index.js
    ├── actions.js
    ├── mutations.js
    └── modules
```

Modularización del VueX

- Modularizaremos el Store de la siguiente manera:

```
├── index.html
├── main.js
├── ...
└── store/
    ├── index.js
    └── modules/
        ├── module1.store.js
        ├── module2.store.js
        └── module3.store.js
```

- **Nota**: Otra forma válida es añadir el apartado store en el directorio del componente que exponga los datos.

Testing

Testing de componentes

- Las pruebas unitarias no solo evitan errores sino que aportan confianza sobre el código a los desarrolladores.
- Garantizan que pueden añadir cambios en el futuro sin temor a que rompa el proyecto.
- Detalles a tener en cuenta
  - El test debe proporcionar mensajes de error claros
  - Es nuestra responsabilidad utilizar una buena librería de testing
  - Se han de realizar pruebas simplificadas por cada componente

Testing. Apunte

- Además de unitarias, Vue admite test E2E y de integración.
- Es buena idea combinarlos con los unitarios para gozar de mayor robustez.
- Vuex es la parte con más dificultad de testing y se recomienda aplicar test de integración.

Conclusión

Conclusión

- Vue como framework puede utilizarse en diferentes contextos.
- Pequeñas interfaces de microservicios, plugins y demás, hasta complejas aplicaciones SaaS SPA con multitud de información a la vez.
- Hemos de ser conscientes de la complejidad del proyecto y debemos aplicar las mejores prácticas para tener siempre una evolución favorable cara un futuro.

Fin.