VUE BEST PRACTICES AND SECURITY

Last Updated on May 23, 2022

Vue.js is a popular JavaScript open-source front-end building platform that is entirely capable of designing single-page applications. Using this language, Vue.js developers can use a single file component instance to build rich applications. For better performance, you can combine the codes with Vue.js. Vue.js framework offers advantages over architectures like Angular and React because of its lightweight nature and unique framework design principles.

Using our technical expertise and experience, we have curated some of the evolving best practices in this blog. Since Vue.js is becoming more and more favorable for developers in 2022, these quick tricks will help you to develop an effortless web application.

Vue.JS Best Practices

1. Always use key v-for with ": key" inside

in a list that can be easily updated. An example of this “Keys” is the most used for HTML lists, animations, and Vue transitions.As an example, we can say without the use of the “:key”, DOM will not update the UI properly. It will get confused about which record to update when duplicate records exist in the list. So, if we want to update the last item from the list, the code will always update the first item if we don’t use the “:key”.

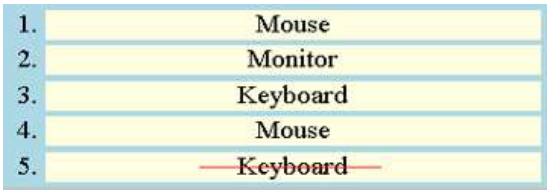
Let’s check it practically:

Example: Render List Without the Key

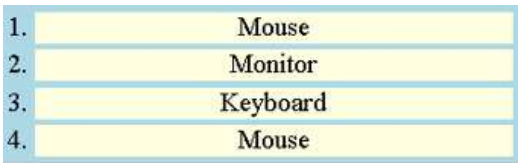
For the demonstration we made one list of computer parts with no IDs assigned. In the code, we have added one functionality where if you click on any part, it will delete it from the list.

```
<ul v-if="itemList && itemList.length > 0">
    <li v-for="item in itemList" @click="deleteItem(item)">{{ item }}
    </li>
</ul>
```

Once we run the above code, we will see a list printed on the browser as shown below. Now let’s try to delete the last item (highlighted one from the below image) of the list.



When we click on that item, instead of deleting the last item from the list, it will delete the third item from the list. As both are duplicate records. So, it will delete the records that come first in the list. So, the output will be like the below.



Example: Render list with the key

```
<ul v-if="itemList && itemList.length > 0">
    <li>
        v-for="(item, index) in itemList"
        :key="index"
        @click="deleteItem(index)">
        {{ item }}
    </li>
</ul>
```

Once we click on the last item to delete, it will actually delete the last item with the key.

4.	Keyboard

2. Always use kebab Casing for Events

It is advisable to use a kebab case where you need to emit custom events. This takes place at the time of using shared components where the same syntax is used for broadcast as well as to listen to the event.

when you use kebab casing for the custom events, it will be easy to identify which is a parent component.

Let see an example of kebab-case for event names which comes with component names:

Item Child Component:

```
// handle data and give it back to parent
this.$emit('value-update', this.localValue)
```

APP Parent Component:

```
<template>
  <div class="container">
    <item :value="value" @value-update="handleData"></item>
  </div>
</template>
```

3. Use Pascal Case or Use kebab case for Components

The best practices is to name a conventional component is by using a Pascal case or use kebab case. The most consistent attribute is the “Import attribute” which is irrespective of whichever project one chooses to work on.

```
# Avoid
itemcomponent.vue || itemComponent.vue || Itemcomponenrt.vue

# Good Practice
ItemComponent.vue
```

4. Keep npm Packages Updated

As per Vue Style guide base components can only contain HTML elements, 3rd party UI components and other additional based components.

Try to regularly update npm packages to avoid any dependency errors and to use the latest/updated features provided by individual packages.

As an example, if you have configured vuetify the design patterns in the VueJS project, vuetify regularly updates its packages to provide the best UI including some breaking changes sometimes. So, it’s better to update NPM packages regularly to avoid bulky or breaking changes at a time of need. We also know about visual studio code- an inbuilt feature that supports base components.

5. Manage Global File for Shared Variable

To make the system more generic and easier to update anytime, manage global configurations (i.e. API URLs. Third-party URLs if any, keys can be used in any integrated tool, theme settings) or in a separate file (i.e environment.json). It will be helpful once your website is live to update any global configurations without any re-deployments.

6. Used Suitable Data Type Assign for Variable

Continuously use proper data types instead of “any” to minimize the casting time and other casting errors.

Avoid any casting inside the loops.

In a case of two data types assigned to the same property, implement type casting using both the types and using conditions.

Example:

```
// Wrong
const age: any = 0; [hard to identify response type at the places this variable used and also hard to track errors on as

// Right
const age: number = 0; [easy to track and identify response type]
```

7. Data Property Initialization

It is recommended to initialize all the data properties that need to be reactive in advance in the data option.

Vue.js constantly observes the change in data by recursively running through data parts.

Use the Watch prop to get the latest values depending on another instead of creating multiple getters and setters. (Avoid the use of a watch inside the array of objects).

Avoid memory leaks – remove custom events, instances, intervals when base components are destroyed.

8. Used a $refs

Always used $refs to get results from DOM and try to minimize the use of JavaScript.

Avoid the use of jQuery, instead, use typescript because typescript allows code rendering faster and to catch and rectify multiple problems at development time.of projects.

9. Do Not Mix v-if and v-for

template to achieve the same.

Let see one example,

```
<ul>
    <li v-if="itemList && itemList.length > 0">
        v-for="(item, index) in itemList"
        :key="item.id"
        @click="deleteItem(index)">
        { item }}
    </li>
</ul>
```

In the above code, we will be able to check if the condition resides in v-if all time while rendering the list or not.

```
<ul v-if="itemList && itemList.length > 0">
     <li v-if="itemList && itemList.length > 0" v-for="(item, index) in itemList" :key="item.id" @click="deleteItem(inde
            {{ item }}
    </li>
 </ul>
```

Here, v-if applied to ul tag, so the v-if condition will be checked only once before rendering v-for items.

Some of the undiscovered advantages of Vue.js rendering are

Rendering is an efficient process because there is no looping for each item repeatedly.

Every time you change the dependency, the filtered list gets re-evaluated.

Rendering helps you differentiate the component logic from the template, which makes the component more readable in the user interface.

10. Vue Component Reusability & Communication

From the Vuex store, you have access to all the reusable components and reusable code of Vue.js. Vue's reusable component is very flexible to use. Try to make a common component with maximum required props and reuse it in all other pages. As an example, you can create common components for the confirmation messages. (You can pass dynamic confirmation text, button text, icons name, etc.). Same way confirmation you can create one component for add/edit functionality with the same model and bindings using the Vue component library.

Data communication between parent and child components in Vue using "props" and "event emitter" is explained below. Once created, we can reuse the object across the single page application by adding required props and code in the component.

Example:

```
<template>
  <div class="container">
    <item :value="value" @onvaluechange="handleData"></item>
    <h3>Value(emitted from child-component): {{value}}</h3>
  </div>
</template>
<style scoped="">
.container{text-align: center;}
</style>

<script lang="ts">
import { Component, Vue } from 'vue-property-decorator';
import Item from "@/components/ItemComponent.vue";

@Component({
  components: { Item },
})
export default class App extends Vue {
  private value: string = "Vue.js Application";

  private handleData(data: string) {
    // get the data after child dealing
    this.value = data;
  }
}
</script>
```

In Child Component (Item.vue):

```
<template>
  <div>
    <label>Enter Value: </label><input v-model="localValue" type="text">
    <button class="save-btn" @click="submitValue">Save</button>
  </div>
</template>

<style scoped="">
.save-btn{margin: 10px;}
</style>

<script lang="ts">
import { Component, Vue, Prop } from "vue-property-decorator";

@Component
export default class ChildComponent extends Vue {
  @Prop({default: ""}) private value: string;

  private localValue: string = "";

  private mounted() {
    this.localValue = this.value // save props data to itself's data and deal with it
  }

  private submitValue() {
    this.$emit('onValueChange', this.localValue) // handle data and give it back to parent
  }
}
</script>
```

For the **custom components**, we can use "**v-model**" to create two-way data binding without using props and event emitters.

Use "**eventBus**" in **unrelated components** for data communication.

11. Data should Always Return Function

Example:

In the below Vue example, the functional data is not returning any output. So, you cannot access properties outside the data function. It will generate a runtime error.

```
 data: {
    value: "Vue.js Application",
    itemList: [],
    counter: 0
  },
```

Right Way:

```
data: function () {
    return {
      value: "Vue.js Application",
      itemList: [],
      counter: 0
    };
  },
```

12. Template Expressions should only have Basic JavaScript Expressions

In the Vue template, you applied more complex logic and some formatting variables like (date formatting). Let's see the first example, we can see more complex logic in the template. What kind of issue we will face in the future.

```
<ul v-if="itemList && itemList.length > 0">
    <li>
        v-for="(item, index) in itemList"
        :key="index"
        @click="deleteItem(index)">
        {{
             item.split(' ').map(function (word) {
               return word[0].toUpperCase() + word.slice(1)
             }).join(' ')
        }}
   </li>
</ul>
```

In the above example, some formatting is applied to the item value. As you can see in the list tag, it will increase the LOC (line of code). At a time of some complex formatting or functionality, these LOC will increase in the template which will be difficult to manage and will be more confusing. So, it's better to separate the bindings by keeping formatting or extra login in some function in code.

Creating a separate function for formatting.

```
<ul v-if="itemList && itemList.length > 0">
   <li>
        v-for="(item, index) in itemList"
        :key="index"
        @click="deleteItem(index)">
        {{
```

```
private normalizedItemValue(value: string) {
    return value
      .split(" ")
      .map(word => {
        return word[0].toUpperCase() + word.slice(1);
      })
      .join(" ");
  }
```

In this “update” function example, we have added a new Vue function (normalizedItemValue) to manage UI formatting to make UI clean, easy to understand for developers, and easy to update.

The image shown below is an output of the entered item name that is displayed as the first character in the upper case.

13. Clean Code and Refactoring

Use a shared/separate file for the static functions/properties for re-usability. It will be helpful to keep a shared/static code at the same file in the entire solution.

Use eslint or tslint analysis tools to maintain code quality.

In Vue, you can reduce the line of code by narrowing down the UI into smaller components.

Use keywords (Const/Let) provided by typescript smartly instead of the var keyword of javascript.

Example:

```
Bad:
var test: number = 0;
 “Identifier 'test' is never reassigned; use 'const' instead of 'var'”

You will get the above error if tslint is configured properly when you run the project

Good:
const test: number = 0; // if static, use const
let test: number = 0; // if updateable, use let
```

Let’s see now common security best practices for Vue.js Application for pro developers

Undoubtedly, the most crucial best practices to follow are for Vuejs. If you ask me what its significance is then the answer is there are many reasons for this. It essentially protects the future you from the current you. It's easy to lose track of the exact format, type, and other conventions you used for a prop when working on a huge project.

If you're part of a larger development team, then you must communicate well, so as to make sure they understand how to use your components. So, please just provide prop validations. You need to avoid challenges and meticulously track all your components to ascertain a prop's formatting. Check out this example from the Vue documentation.

```
props: {
  status: {
    type: String,
    required: true,
    validator: function (value) {
      return [
        'syncing',
        'synced',
        'version-conflict',
        'error'
      ].indexOf(value) !== -1
    }
  }
}
```

15. Components declared and used ONCE should have the prefix "The"

If you don't know what Single instance components are then let's understand it as those used only once per page and do not accept props. These types of components have their own name scheme, similar to base components. These are components that are unique to your programme, such as a header, sidebar, or footer. For every component, there should only be one active instance.

```
TheHeader.vue
TheFooter.vue
TheSidebar.vue
ThePopup.vue
```

16. Consistency using directive shorthand

The usage of shorthand for directives is a predominant method among Vue developers.

For example, @ stands for v-on; : colon- stands for v-bind; and similar other definitions

Using these shorthands in your Vue project is a wonderful idea.

However, if you want to establish a project-wide standard, you should either use them all the time or never at all. Your project will be more unified and readable as a result of this.

For no reason, if you call a method that is created and then monitored is a common mistake Vue developers make (or maybe it was just me). The idea is that the watch hook should be called as soon as a component is initialised.

BAD!

```
created: () {
  this.handlePropertyChange()
},
methods: {
  handlePropertyChange() {
    // stuff happens
  }
},
watch () {
  property() {
    this.handlePropertyChange()
  }
}
```

Vue, on the other hand, offers a predefined in-built solution for this. It's a feature of Vue viewers that we often overlook.

All we need to do now is reorganize our watcher and declare two properties:

1. handler (newVal, oldVal) – here is the actual observer method:
2. true – when our instance is created, our handler is called.

GOOD!

```
methods: {0
  handlePropertyChange() {
    // stuff happens
  }
},
watch () {
  property {
    immediate: true
    handler() {
      this.handlePropertyChange()
    }
  }
}
```

18. Use Actions: Commit the Data and Make API Call

Use Vuex actions to make the majority of my API calls because they make it so much easier to get data and provide a level of reusability and encapsulation. If I need to fetch the same web page from two separate locations, I can use the dispatcher and the appropriate parameters to fetch, commit, and return with no additional code than the dispatcher.

and the code base is simple to maintain.

19. Name Commits using single convention method

As the program reaches stability and matures, we'll all be forced to look over the component's history. If you or any of your team members do not use the same naming convention for their commits, it will be difficult to distinguish between them and comprehend what they do.

To make commits easier when browsing through the project history, follow the criteria stated below as one of the Vue js best practices.

Commit Message Format

```
< type >( <  scope > ): < subject >
< BLANK LINE >
< body >
< BLANK LINE >
< footer >
```

20. Multiple V-Condition

Using numerous v-if conditions to render multiple elements from a Vue component's render method is not recommended. Wrap the elements with and the extra ones with <template >.

```
< template v-if="true" >
 < p >Paragraph 1 < / p >
 < p >Paragraph 2 < / p >
 < p >Paragraph 3 < / p >
< / template >
```

21. Code Splitting

Everyone knows that performance is of utmost importance. As it turns out to be more and more important, you must find efficient ways of code splitting. Async components like

```
Vue.component('async-component', (resolve) => {
  resolve({
    template: '< div >Async Component< / div >',
    props: [ 'mypropVariable' ]
  });
});
```

Single file components

```
< template >
  < div >A sync Component < / div >
< / template >
< script >
  export default {
    props: [ 'mypropVariable' ]
  }
```

```
  components: {
    AsyncComponent: () => import('./AsyncComponent.vue')
  }
});
Dynamic Module Loading
export default {
  template: '< div >Async Component< /div >',
  props: [ 'mypropVariable' ]
}
You may try something like this;
import AsyncComponent from './AsyncComponent.js'`;
Vue.component('async-component', AsyncComponent);
```

22. Routing

Client-side routing is the most prevalent method for creating SPAs.Vue does not come with built-in routing because it has an official plugin called VueRouter that is extremely straightforward to use and includes all of the features you'll need to create a sophisticated application.If you're using the VueCLI, you can add it to your app without using the npm-install Vue-router command.

23. Eliminate the DOM access directly

Avoid attempting to access the DOM directly, when programming on the Vue application at all costs. Instead, you should use $refs, I find it an ideal way to access the DOM, and it's more maintainable and you are no more required to depend on specific class names.

24. Add multiple classes to an element

One of the best things about Vue is it makes so convenient to add dynamic class to an element

```
//Add class red if isValidationError is true
< div :class="{'red': isValidationError}" >< / div >
```

The common yet a unique approach

```
// Add to classes if two properties return true
< div :class="{'red': isValidationError, 'text-bold': isSuccess }" >< / div >
```

25. Use Selectors

The codes will better explain the concept

```
// We have this selector
export const language = (state) => state.userConfig.language;// In one of our actions, we need language:
// Bad
[GET_LANGUAGE]({ commit, rootState }) {
   const languageSelector = rootState.userConfig.language;
   // Do stuff...
}
// Good
[GET_LANGUAGE]({ commit, rootState }) {
   const languageSelector = language(rootState);
   // Do stuff...
}
```

While developing an application using Vue.js, we're mostly concerned about performance, SEO, API calls, and UI/UX, so the security of the existing application is often overlooked by the developers. There are thousands of malicious attacks that can happen from the frontend side, some of which are mentioned below:

- Unrestricted File Upload
- Clickjacking
- XSS Attack
- SQL injection
- A denial-of-service attack (DoS attack)
- Session hijacking

So to overcome this the best practices is to have collection of Vue.js best practices while developing web applications.

26. Enable XSS Protection Mode

In the case where an attacker hacks and inserts a malicious code in the user input, we can enable the **"X-XSS-Protection": "1; mode=block" header** by preventing the response. The advanced browsers are well equipped with XSS protection mode were adding an X-XSS-Protection header is highly recommended. This gives an assurance to the old browsers on higher security concerns that do not support CSP headers.

27. Avoid Typical XSS Mistakes

An XSS attack is typically monitored in DOM API's inner HTML section. Say for an example.

```
document.querySelector('.application').innerHTML = value;
```

It is possible that the attacker can unknowingly insert a malicious code using the above line. So it is advisable not to set the inner HTML value-based on the user input. Instead of that, you can use textContent.

28. Use Captcha

You should always promote using captcha at the end-points where there is a larger audience to address for large-scale projects. Examples are login, registration or contact id. If you ask what is Captcha then A captcha is a computerized program or system that is designed to differentiate between humans and robots. It also prevents DoS (Denial Of Service) attacks.

29. Audit Dependencies Regularly

Run **npm audit** commands regularly to audit packages. This command shows all outdated versions of packages and vulnerable packages and suggests upgrading packages to the latest version.

30. Use of Third-party Packages

The more you use third-party packages in your project, there can be security issues rising as vulnerabilities in open-source libraries. This will instigate hackers to do more fraud. While selecting Vue.js components from the package to integrate with the application, the expert Vue.js developer needs to check if the library is open source or not as lack of transparency poses a security risk, security best practices is where the library is well-documented, recommended, meets specific requirements, and is actively supported by the author.

31. Disable our App to Load in an iframe

We should disable loading our app into the iframe. To restrict the rendering of a website in an iframe, you can enable the DENY option in X-Frame.

32. Displaying Generic Error Message for Authentication

While you are developing a user authentication form you must keep in mind certain factors. In your form, if the user enters the wrong password then a message is displayed saying “Your Password is incorrect”. This is not a good way because it helps an attacker to identify your email address or user id as valid and the password is wrong. Then the attacker starts a brute-force attack or dictionary attack to find the password for that user. That is no good for us. The solution is, you just display a message like ‘Incorrect login detail entered’.

Conclusion

With this, we have reached the end of the blog. We hope you found this insightful and will help you in maintaining best practices while developing apps using Vue.js. With our experience, we have curated these tricks and tactics that will help Vue.js developers to make code easily maintainable and accessible. Hopefully, these tips were useful to you because that is what we intended for your development project to be seamless and hassle-free. Cheers!

Perform npm-update

Global Config for Shared Variables

- Regularly update npm packages to avoid any dependency errors.
- Some dependencies are being regularly updated, so it's always a good idea to do npm-update before starting work for the day.

recommended to have global configurations in a separate file, like **environment.json**

Never forget ":key" inside v-for directive

key is a vital argument to distinguish between the data records

```
<ul v-if="itemList && itemList.length > 0">
<li
    v-for="(item, index) in itemList"
    :key="index"
    @click="deleteItem(index)">
    {{ item }}
</li>
</ul>
```

How To

- Always declare : **key** argument in the **v-for** directive.
- Without : **key** argument, vue engine cannot recognize the uniqueness of the data record and leads to unintended behaviour in case of duplication of data records.

kebab-case for events

- While emitting custom events, always go with **kebab-case**.
- Kebab-case makes it easier to identify parent component.

```
'camelCase',
'lowercase',
'lower-kebab-case',
'lower_snake_case',
'pascalCase',
'Sentencecase',
'UPPERCASE',
'UPPER-KEBAB-CASE',
'UPPER_SNAKE_CASE'
```

PascalCase for components

- The best way to name a conventional component is by using **pascalCase** or **kebab-case.**
- It does not matter which one you choose for your project. 'Import' attribute is always consistent all the time.

Do not mix v-if and v-for

Avoid

- Don't use **v-if** on the same element as **v-for.**
- With this approach, the vue engine will check for **v-if** condition in every loop which can be resource consuming and impactful on the performance.

Do

- Instead we can apply **v-if** condition to the parent tag or template to achieve the same.
- **v-if** is applied to <ul> tag, so v-if condition will be checked only once before rendering v-for items.

```
    v-for="(item, index) in itemList"
    :key="item.id"
    @click="deleteItem(index)">
    { item }}
    </li>
</ul>
```

```
    v-for="(item, index) in itemList"
    :key="item.id"
    @click="deleteItem(index)">
    {{ item }}
    </li>
</ul>
```

Minimize use of "any" datatype

- It is a good practice to use proper data types instead of "**any**" to minimize the casting time and other casting errors.
- Avoid any casting inside the loop.
- In a case of two data types assigned to the same property, implement type casting using both the types and using condition.

Avoid

```
const age: any = 0;
```

This variable makes it hard to identify response types at different places and track their errors on the assignments.

Do

```
const age: number = 0;
```

Easy to track and identify the data type.

Use $refs

- Always use **Srefs** to get value from DOM and try to minimise use of JavaScript.
- Avoid use of jQuery, instead use typescript because typescript allows faster code rendering and makes it easier to catch and rectify more problems at development time.

Exmaple

```
const subs = this.$refs.item as Item;
```

Data initialization

- Initialize all the data properties that need to be reactive in advance in

Clean code and refactoring

- Use a shared/separate file for the static functions/properties for the re-usability. It will be helpful to keep a shared/static code in the same file for the entire solution.
- Use **eslint** or **tslint** analysis tools to maintain code quality.

- Reduce the line of code by narrowing down the UI into smaller components.
- Use keywords (**const** or **let**) provided by typescript instead of var keyword of javascript.

values depending on another instead of creating multiple getters and setters. **Avoid the use of a watch inside the array of objects.**

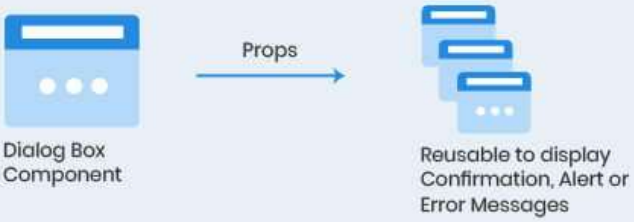
- Avoid memory leaks – remove custom events, instances, intervals when components are destroyed.

```
const test: number = 0;
// if static, use const

let test: number = 0;
// if not static, use let
```

Component reusability & communication

- Try to make a **common component** with maximum required props and reuse it in all other pages.
- Data communication between parent and child components using **"props"** and **"event emitter"**



- For the custom components, use **"v-model"** to create **two-way databinding** without using props and event emitters.
- Use **"eventBus"** in unrelated components for data communication.

Data should always return a function

- When you declare a component data, that aspect of data should return a value.
- In other cases when it does not return a value, that information of data will be shared at all instances of the component value.

Avoid

```
data: {
   value: "Vue.js Application",
   itemList: [],
   counter: 0
}
```

Do

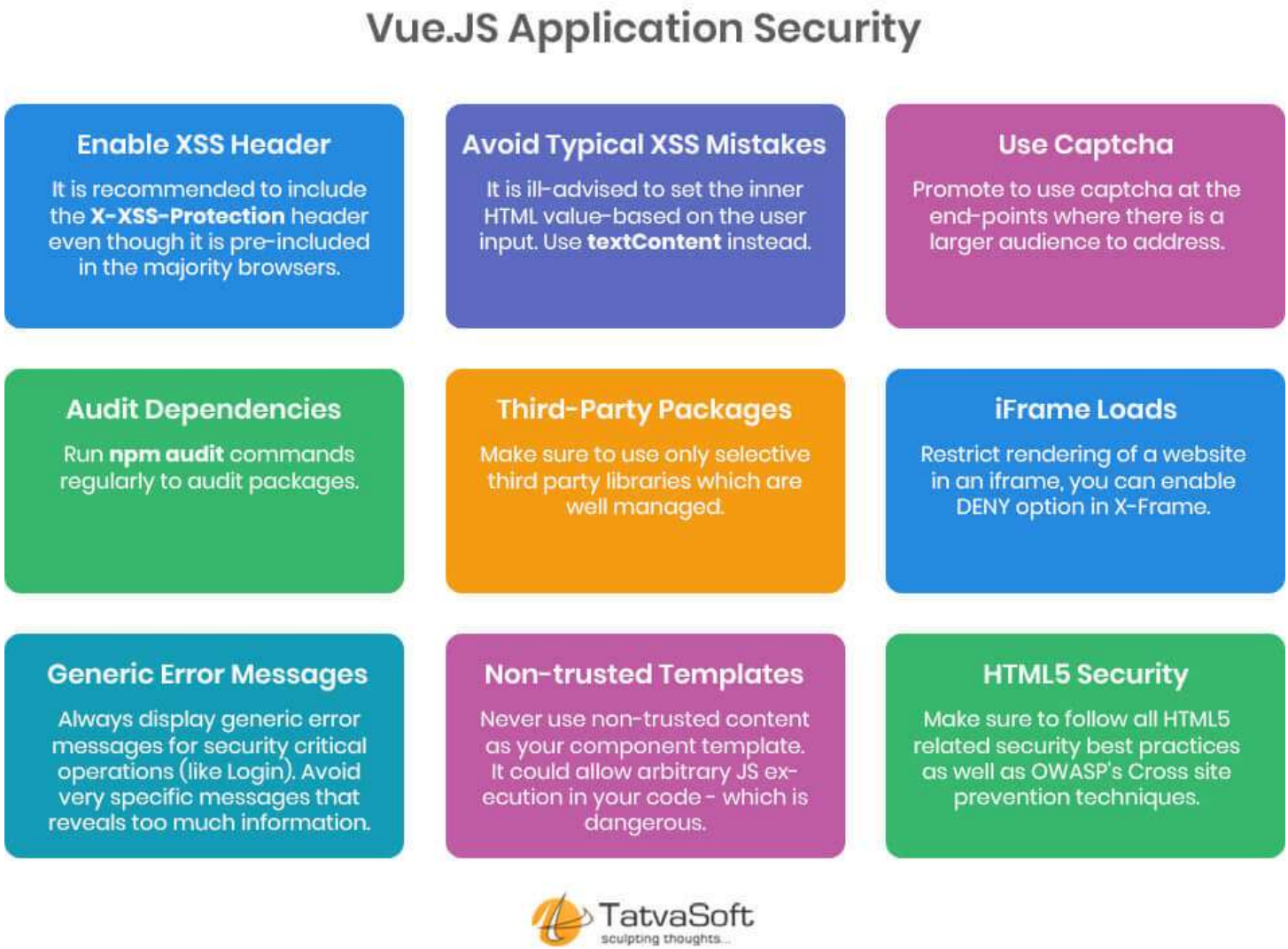
```
data: function () {
   return {
      value: "Vue.js Application",
      itemList: [],
      counter: 0
   };
}
```

Avoid lengthy javaScript in template expressions

- Adding complex logic to the template expressions can introduce unnecessary clutter in the code making it confusing to read and difficult to manage.
- It is advisable to separate the bindings by keeping complex logic in a separate function in the code.

```
<ul v-if="itemList && itemList.length > 0">
   <li
     v-for="(item, index) in itemList"
     :key="index"
     @click="deleteItem(index)">
     {{
        normalizedItemValue(item)
     }}
   </li>
</ul>
```

```
private normalizedItemValue(value: string) {
  return value
   .split(" ").map(word => {
   return word[0].toUpperCase()+ word.slice(1);
  })
  .join(" ");
}
```



Want to embed this image? Please cite source to TatvaSoft.com

Share this Image On Your Site

```
Please include attribution to TatvaSoft.com with this graphic.
<a href="https://www.tatvasoft.com/blog/vue-js-best-practices/"><img src="https://www.tatvasoft.com/blog/wp-content/uplc
```

Other Best Practices Blogs:

1. Angular Best Practices and Security
2. React Best Practices and Security
3. NodeJS Best Practices and Security

6. SharePoint Best Practices
7. .NET Core Best Practices

Vishal Shah

Vishal Shah has an extensive understanding of multiple application development frameworks and holds an upper hand with newer trends in order to strive and thrive in the dynamic market. He has nurtured his managerial growth in both technical and business aspects and gives his expertise through his blog posts.

Prev Post
AWS Lambda vs Azure Functions: Serverless Computing

Next Post
React Best Practices and Security

GET IN TOUCH

Comments

Leave a message...

MAHADAJI BHAGAT
Mon, Jun 20, 2022, 5:45PM

One of the major benefits of using vue is the reusability of code. By reusing the component developers are able to improve the code structure and can simplify maintenance and testing.

Reply

SAMUEL BURNS
Mon, Jun 20, 2022, 5:44PM

One best practice is to refactor big code into smaller reusable code in order to make it reusable, saving your time, and making the project more reliable and maintainable.

Reply

RULDU BAJPAI
Mon, Jun 13, 2022, 5:34PM

Using third party packages in your development project is a common practice but it can cause security issues or there may be bugs that can cause error or slow down your development process. So try to audit these packages on a regular interval to ensure the best code quality.

Reply

JAMES LAWRENCE
Mon, Jun 13, 2022, 5:33PM

Many developers face the issues of not being able to keep the base component together. To solve this issue the best practices of naming base component can be adopted where by developers use a affix a like App, Base. It is completely up to you until you keep it

RAJIV MANGAL
Tue, May 10, 2022, 3:05PM

Is Vue.js SEO friendly? And can it be used for Mobile app development

Reply

DAYTON GILES
Thu, May 12, 2022, 10:19AM

Vue.JS by default is not SEO friendly but, as a developer you must be knowing about the modules of Vue.JS that are SEO friendly and make use of them during your development process. Talking about mobile support since Vue.js is a web framework, it cannot support Mobile app development at its core or on its own.

Reply

MADHUR PRASAD
Tue, May 10, 2022, 3:03PM

Who founded Vue and for what purpose? Was there really a need for Vue?

Reply

BROCK LEVINE
Thu, May 12, 2022, 10:16AM

Evan You is the person who founded Vue who is a former employee of Google. Evan liked Angular but the thing that he didn't like was Angular being a heavy framework. So he created Vue that had all the good of Angular but he built Vue to be extremely lightweight.

Reply

MITRA NAGAR
Wed, Apr 20, 2022, 11:04AM

What is Vue Good for?

Reply

CALEB HOWARD
Mon, Apr 25, 2022, 10:01AM

Vue is good for startups as well as for building large-scale applications. Vue is simple and easy to learn, offers high-grade production-ready apps for Android and IOS. It can also be used for building Single Page Applications.

Reply

I was not sure if I wanted to implement any of the best practices but today as Vue has grown and become widespread, these best practices have become a standard. You have to implement them in your project to improve your chances of getting an effective app.

Reply

ANTONIO HODGE
Fri, Mar 4, 2022, 6:05PM

I was relying a lot on third-party packages for my development project and I was not sure if I was doing the right thing. But after reading the security best practices I came to know that it is not recommended. Really appreciate your work.

Reply

HAYDEN YOUNG
Fri, Mar 4, 2022, 6:02PM

The Code splitting best practice is something I came across while I was reading the blog and it has helped me a lot in improving my site's overall performance. Thank you for this blog.

Reply

ENZO MURPHY
Tue, Feb 15, 2022, 12:03PM

It is a nice article, but I have a doubt on what are famous Development tools a developer should consider learning?

Reply

ANSHU DAVE
Wed, Dec 15, 2021, 3:49PM

Code Splitting is splitting of code into various components or bundles which can be loaded on demand. This helps in running the application smoothly and decreases the loading time of complex applications.

Reply

VINAY TAVADE
Wed, Dec 8, 2021, 4:22PM

What is Code Splitting?

Reply

NEIL LAWRENCE
Wed, Dec 8, 2021, 4:21PM

KIM DATTA
Wed, Dec 15, 2021, 3:44PM

Vue is a prgressive framework which is used for building user interfaces. Vue is designed to be very adaptive, easy to integrate with other libraries. Vue is capable to power complex SPA (Single page Applications) when combined with supporting libraries or tools.

Reply

GEORGE AUSTIN
Fri, Oct 29, 2021, 12:16PM

An XXS Vulnerability is also known as a Cross-Site Scripting Vulnerability, it is a type of malicious code injection Vulnerability. It usually happens due to an attacker who sends a harmful or malicious code to another end-user on the website, but the other end user's browser doesn't know that it is not trustable it executes the script, and after this attacker has access to the cookies of visitors.

Reply

ZACHARY GRAY
Tue, Oct 26, 2021, 10:03AM

There is a term used in the Vue Security section i.e. XSS Attack, what does it mean by that?

Reply

JENNIFER MILLS
Mon, Oct 11, 2021, 12:43PM

Hello, Henry here is your answer: Captcha is used by any website that would like to restrict usage by the bot, by making sure that each vote is entered by a human, reducing the bot registration or fake accounts, preventing the large tickets from getting bought by one user (usually bought in bulk for resale), controlling online harassment not allowing fake comments, message bots or bots providing reviews on sites.

Reply

HENRY RILEY
Fri, Oct 8, 2021, 2:38PM

What is the purpose of Captcha?

Reply

JOHN D
Thu, Sep 23, 2021, 1:00PM

or letters.

Reply

KENDRA N. BATTLE
Thu, Sep 23, 2021, 12:50PM

What is a Captcha?

Reply

LARA BURKE
Fri, Sep 10, 2021, 3:37PM

I am really impressed with the infographic that you have presented at the bottom of the article. The infographic is like a long story short because you will get all the explanations described in a very unique way.

Reply

CARLY J
Wed, Aug 4, 2021, 2:32PM

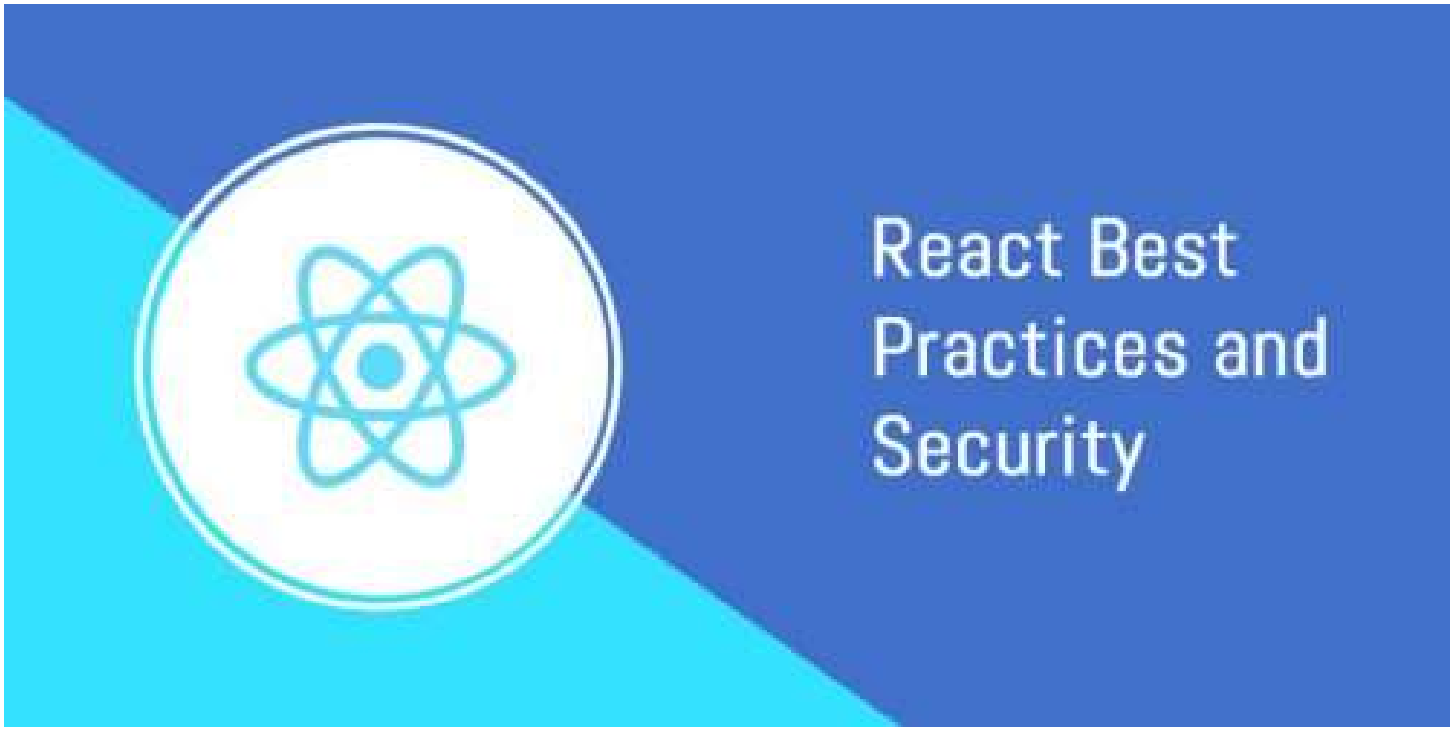
It is very rare to find a good subject on Vue due to its popularity when you compare it with other front-end frameworks like React and Angular. I found your article very subjective and you have maintained a well-defined process to explain all the best practices

Reply

Related Articles

Node js Best Practices and Security

Apr 28, 2022



React Best Practices and Security

Mar 7, 2022

Angular Best Practices and Security

May 23, 2022

Ready to build your custom application solution?

Send us requirements on **info@tatvasoft.com** or call **+1 (972)-202-6489**

REQUEST A PROPOSAL

United States

17304 Preston Road,
Suite 800, Dallas,
Texas, 75252

+1 518 282 4642

India

TatvaSoft House,
Rajpath Club Road,
Ahmedabad, Gujarat, 380054

+91 960 142 1472

United Kingdom

307B, Warnford Court,
29 Throgmorton Street,
London EC2N 2AT

+44 (0)207 947 4950

Sydney

Ground Floor, Level 3,
Suite 2, 828 Pacific Hwy,
Gordon NSW 2072

+61 2 9416 0440

Melbourne

Canada

+61 3 9581 2659

+1 647 978 7562