Nayan

Start

I started of by exploring the code of the project. And figuring out how things work. I am positively surprised how the project was put together, but I had some questions:

- Why would you consider creating your own boiler code project VS tools like NuxtJS & create-Vue
- I see async/await & promises being used in the same file (App.js)
- Would you consider using web components over using HTML with ‘data-vue-component’ attribute.
- Imagine you have multiple “routes”, how would you approach dynamically loading the components only needed for that route?
- How will you work with nesting components?

These questions are NOT about how I would do it differently, I’m just trying to start a conversation here😊

Countdown

I started creating the Countdown component, where I wanted to pass the countdown date as unix timestamp as a prop in milliseconds. The first goal was to render the days, hours, minutes, seconds on mount of the component. I noticed we were using Vue2, but we are able to use the functions from the Composition API (ref & computed). In order to show the counter immediately we call the helper function “updateCountdown”. In order to update the counter every second, we can use “setInterval” with 1000ms. The setInterval is cleared when unmounting the component.

I also saw in the design there are leading zeros. There are multiple ways to add leading zeros:

- In the template with a ternary operator
- Directly add zeros to days, hours, minutes, seconds
- Use computed, with new variables (formattedDays, formattedHours, formattedMinutes, formattedSeconds).

I did not use template, because the template will look cluttered. I preferred using computed in this case because in the future we could do a calculation with days, hours, minutes, seconds.

“countdownEnd” is a prop you can pass to Countdown component in milliseconds. So you can re-use this component to count down to other dates as well.

Header

I wanted to create a new Vue component, but not found a way to nest components. That’s why I added the HTML in the index.html (and CSS in main.scss) directly. Made the header more mobile friendly by adding media queries.

Info

This component renders a list of "info" items. I created a Vue component for this, because the text could be coming from a CMS + I prefer creating multiple small components rather than creating big HTML templates. The items are static for this POC, which is a list of objects with fields: id, name, description.

I wanted to insert HTML in the "description" field, so I used "v-html" to render the 24 November 2023 in a "b" tag to make it bold. I am aware of the XSS vulnerability using v-html. There are a 2 (or more?) solutions we could do:

- Don't make it dynamic, just add the items in the template itself (without v-for)
- Use a library like (e.g. dompurify) to sanitize the HTML, where you whitelist the HTML tags you want.

Improvement I would have done: Split up the InfoItem component in a separate component.

Newsletter

This component contains a form, which will do a POST request to /api/newsletter with the email address in the payload. The template consists of title, description and a form.

I've used axios to do the POST request. I still prefer using axios over fetch, because it automatically converts your payload to JSON (which is 90%+ of the cases). Axios has interceptors, where you can add a response/request handler. Those interceptors can come in handy if you want error handling to be consistent throughout your application.

I disabled the submit button & removed focus of the email input, if the subscription was successful. Imagine, I want to subscribe with another email address, I want to quickly do this without refreshing the page; So I made it possible when refocusing the input field, you can submit again (by enabling the button). When inserting existinguser@nayan.be, an error gets shown to the user. If the subscription was successful, there also is an (general) error message shown to the user.

A remark I have with the unhappy flow, when the user already exists: I would use the 409 status code. Now it looks like the subscription was successful. Now you have to do a frontend check if(success === true). IMO it's better to only return status code 200 if the action really was successful, else return 4xx (or 5xx) codes. Remark on this remark: it all depends on the conventions of the team/company of course 😊

Emarsys

First of all I looked into the Emarsys API documentation, before starting the implementation. This way I understand the available endpoints, parameters, request methods and response format.

To make sure I can do authenticated requests, I'll have to make sure we have an account set up on Emarsys + make sure we have a user name and secret for the API. The API uses the WSSE format. The X-WSSE header is generated from the user name and secret and consists of the following mandatory elements: UsernameToken, Username, PasswordDigest, Nonce, Created. To compute the

PasswordDigest, follow this: https://dev.emarsys.com/docs/emarsys-api/ZG9jOjI0ODk5NzAx-authentication#example -> This will create a helper function getWsseHeader

After setting up the authentication we can start building the API request.

Before implementing, we need to make sure there's an existing email campaign for our newsletter. This can be done by doing a POST call to: https://api.emarsys.net/api/v2/email with example body, that contains fields: name, administrator, language, subject, fromname, fromemail, email_category, html_source, text_source, browse, text_only, unsubscribe, filter (+ X-WSSE header obviously)

When we have an existing email campaign, we can start implementing the subscribe functionality. We replace the /api/newsletter url by the following:

1/

First of there is a "Send a Test Email" POST endpoint: https://api.emarsys.net/api/v2/email/{emailId}/sendtestmail

With emailID (= email campaign which you set up previously) and payload, with fields: subject & recipientlist.

2/

When the test was successful, we can do the actual "launch" of the email campaign.

Replace the url by: https://api.emarsys.net/api/v2/email/{emailId}/launch

emailId stays the same

The payload contains fields: emailId, schedule, timezone, features

In the actual API calls, we'll use the getWsseHeader helper function. We'll add the X-WSSE header, and the value will be getWsseHeader(user, secret)

The success/error handling is similar as previously implemented.

Follow up on the Emarsys dashboard about the email campaigns 😊