



B5 - Application Development

B-DEV-500

Road to Instagram

Create your own Instagram with Nuxt



1.0



WHAT IS VUE JS ?

Application software development is one of the most popular businesses being practiced both at individual as well as enterprise levels. Different tools and techniques are being used by the developers for launching successful applications.

A lot of software technologies are also being used by the developers to make the applications faster, more attractive and user-friendly.

Vue.js is one of those new software technologies that are being widely used across the world for web development.

Vue.js is actually a JavaScript framework with various optional tools for building user interfaces.

See [Vue js documentation](#) for more informations.

To simplify even more our app's development, we are going to use Nuxt.js.

With Nuxt.js, your application will be optimized out of the box. Nuxt builds performant applications by utilizing Vue.js and Node.js best practices. To squeeze every unnecessary bit out of your app Nuxt includes a bundle analyzer and lots of opportunities to fine-tune your app.

See [Nuxt js documentation](#) for more informations.





INSTALLATION

Knowing about a tech is something, installing it is another.. Let's try without spending more than 15 minutes on it, if we can !

If you already have yarn or npm, great. If not, please install yarn the following way :

```
Terminal
~/B-DEV-500> curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key add -
~/B-DEV-500> echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee /etc/apt/sources.list.d/yarn.list
~/B-DEV-500> sudo apt update
~/B-DEV-500> sudo apt install yarn
```

Then, install vue and nuxt. Ask any question if something is wrong.

```
Terminal
~/B-DEV-500> yarn add global vue
~/B-DEV-500> yarn add global @vue/cli @vue/cli-init
```



Replace yarn with npm if you don't have yarn. But be careful ! Syntax is a bit different.

Now that we are set, we can start the project.

Launch the following command line to initiate a nuxt project, and give it a name.

```
Terminal
~/B-DEV-500> yarn create nuxt-app ProjectName
```

For project name, project description and author name, press enter. It will automatically fill the form based on your project name.

You then have to choose between yarn and npm, depending on which you are using. For the UI framework, please choose Vuetify.js.

We won't need a server framework so choose the recommended option (none).

Since we want to deploy our application on phones, we have to implement a progressive web app (web app that can appear like a native app) so choose that option for the Nuxt.js modules by pressing space on it and enter to validate.

Since we don't need a linting tool, just press enter on the next option.

No test framework is needed and universal rendering mode is fine.

To simplify development, add jsconfig.json.

Finally, install the vuex-persist plugin, you'll need it later.



```
Terminal
~/B-DEV-500> yarn add vuex-persist
```

Our project is installed ! Finally.



PREPARATION

First thing first, let's clean our files so we can start on a white page.

Open the project file in your favorite text editor, open the **nuxt.config.js** file and remove the line `dark = true` in the **vuetify** description (around line 53).

Inside the **layouts** directory, find the **default.vue** file and clean it.

```
<template>
  <v-app>
    <v-content>
      <nuxt />
    </v-content>
  </v-app>
</template>

<script>
export default {}
</script>
```

Last but not least, go to the **pages/index.vue** and clean that file to.

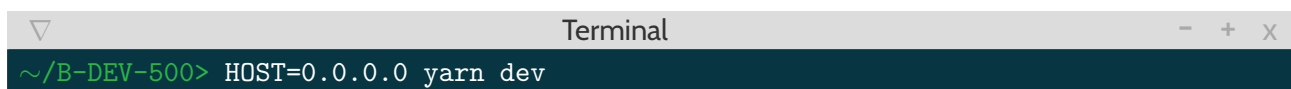
```
<template>
  <div>
  </div>
</template>

<script>
export default {}
</script>

<style>
</style>
```

Choose three pictures on internet and download them into the **assets** directory of your project. We will need them later.

We are ready to go ! To launch an app, you can either use `yarn dev` or `npm run dev`. But today we want to develop our app on our phones, so we are going to use the IP address, and then add our web app to your home screen (like a native app). Your phone has to be on the same wifi then your computer ! First, launch the project :



```
Terminal
~/B-DEV-500> HOST=0.0.0.0 yarn dev
```

Take the given **IP address** and go to it on your phone : you should have a white page. Depending on which phone you have, go to the menu and add your web app to your home screen (on an Iphone, open the safari's "share" menu and choose "add to home screen").

Your phone should now have a **vue js** icon on its home screen : it is your app, it will update in real time.

FIRST PAGE

We can't have an app without a Navbar. That's the first thing we are gonning to implement. To do so, we are going to use a component of Vuetify. A component is a **functionnal and styled element** (like a button, a menu, a form..) you can use from a specific library. For this project we are using the Vuetify library (which you add during the start up of the project), so you'll be able to access its components. We are now going to add the Navbar component of Vuetify. Go to the **index.vue** file. If you don't already know HTML, just know that it works with balistics, like

```
<div></div>
```

(the main one). So you can add a div balistic just inside the first one, in which we will add our Navbar component.

```
<div>
  <v-app-bar></v-app-bar>
</div>
```

So, we have a "navbar". Well, just a rectangle actually. Let's add a title to it. Since you know how to add a balistic, add a **v-toolbar-title** inside the **v-app-bar** balistic. Add a title by writing it between the opening < and the closing one.

```
<balise>TITRE</balise>
```

So the navbar doesn't move when we scroll, we can add a key word **fixed** inside the balistic. Key word allows us to specify some options on the component, here :

```
<v-app-bar fixed dark>
```

The "dark" key word specify the theme, it will allow us to change the color by adding it next :

```
<v-app-bar fixed style="background-color:rosybrown" dark>
```

The style key word is an easy way (when it is just for one or two specifications) to add CSS to an element. Feel free to change the color !

Okay, so we have something like a navbar now. Let's add content to our page !

We are now going to add the main feature of our application : pictures. To do so, we are going to add a **Card component** from Vuetify. Follow the next few steps :

Since you now know how and why we use balistics, you won't be surprised the next thing is adding a div balistic under the one containing the Navbar. It will be our main container, containing our app's body. Before adding the Card component, we are going to talk about CSS for a bit. Right now, our body's width is the screen width : everything we add in it will be the size of the screen. We want it to breathe !

To do so, we are going to give it a **class** with the name **container**. Like style, a class is added like `class="container"` on the div balistic.

To specify which CSS style element we want our div to have, we have to create the class in the style balistic, at the end of the document. A class is declared that way :

```
.container {
```



```
margin-top: 70px;
display: flex;
flex-direction: column;
align-items: center;
}
```

The margin-top line tells the div to start under the navbar. (approximatively, change as you want). The three other lines are for centering the div : a bit complicated to explain, I will just give you the code for this time.

So we now have a body set as we want. Let's add the Card component.

Like the navbar component (which has its main ballistic and then one for the title), we will add an image, a title and a description inside the main one. Something like that (I added the correct width) :

```
<v-card width="350px">
  <v-img :src="require('source')"></v-img>
  <v-card-title>TITRE</v-card-title>
  <v-card-text>DESC</v-card-text>
</v-card>
```

Replace the source with the source of one of your previously downloaded images, and add a title and a description. Great, you should have a card on your screen.

If we are going to recreate Instagram, we have to be able to like our pictures. Let's add it to our card component :

```
<v-card-actions>
  <v-btn icon>
    <v-icon>mdi-heart</v-icon>
  </v-btn>
</v-card-actions>
```

You can add a v-spacer ballistic to put your button to the right.

If we want our button to do something, it needs a function when clicked. The function will change the state of the button, which we will declare inside the data field of the script.

Here's the function and the data :

```
data() {
  return {
    liked: false
  },
  methods: {
    likeItem() {
      this.liked = !this.liked;
    }
  }
}
```

Find a way to call that function whenever the button is clicked. Hint : @click !

Once you have done that, you can see that the button doesn't change (oups !). We have to tell the icon to change color depending on its state.

How about a if / else ? Complete the code below !

```
<v-icon v-if="" color="pink darken-1">mdi-heart</v-icon>
<v-icon v-else>mdi-heart</v-icon>
```

Your button should now change colors whether you liked the picture or not. Great !



LOOPS

One photo is cool, but not great. Let's add some more.

We are going to talk about **loops**. It works the same way as a c loop, but the syntax is a bit different.

Since we want the card component to be duplicated, we have to add a div ballistic around it, just for the loop. In that div, add the following :

```
v-for="index in 3" :key="index"
```

Your card should be duplicated in three !

Okay, it is three times the same thing, but still !

To change that, you have to change your title/description/picture to array instead of direct text (like in C !). Just like you added a liked data, add three array containing :

- titles
- descriptions
- pictures name
- liked state (replace the liked data)

Since our loop's index starts at 1, we have to add '- 1' when using it (well yeah, we are working with arrays). Since it is possible you don't know how to access data from html, I will show you the two main ways :

```
<v-img :src="require('../assets/' + imgs[index - 1])"></v-img>  
<v-card-title>{{titles[index - 1]}}</v-card-title>
```

With that information, you can guess the rest !

Then, in **likedItem()**, add the line :

```
Vue.set(this.liked, index, this.liked[index]);
```

at the end.



Something else has to be changed in the likedItem() function.

You should now have three different cards ! Looking a bit more like Instagram right ? (Say yes or get out :))

NEW PAGE !

How about adding a page containing the pictures we liked ? Let's add a menu to navigate, and then let's add a new page !

Vuetify did well with its components, because it is really easy to add a menu ! I will spare you the harm of having to search for it, so here's the greatest component of all :

```
<v-menu offset-y>
  <template v-slot:activator="{ on }">
    <v-btn icon v-on="on">
      <v-app-bar-nav-icon></v-app-bar-nav-icon>
    </v-btn>
  </template>
</v-menu>
```

Add it inside the navbar component, just below the title line.

The activator tells the menu to open its list (we are going to add it next) when the button is clicked. Yay Vuetify !

Once again, the v-spacer ballistic can put your icon to the right, just saying...

We have an icon but we don't have a menu. Let's add the list we were talking about !

Just under the menu template, add a v-list, and a v-list-item ballistic (the second under the first). Then, add the following :

```
<nuxt-link :to=" " >
  <v-list-item-title>PageName</v-list-item-title>
</nuxt-link>
```

To have two pages, add a loop **index in 2** inside the v-list ballistic (you should know how to do that now).

You have Two spots. Links can't work since it doesn't have one !

Create two more arrays :

- PageNames
- PagesLinks

Our second page name will be **likes**.



link to the index.vue file is '/'

We have our links ! Likes doesn't work since the page is not created yet. Let's do that now !



LIKES

You have the main page, and the likes page will be really close from it. Here are the main features :

- navbar with menu
- list of cards (without the possibility to like them)

Since you wrote the main page, it will be easy for you to implement the likes page. You have everything you need !



STORE

We are now going to talk about the store. Right now, all the cards are on our likes page, whether we liked them or not.

To correct that, we need to know if a card is liked or not on any page.

That's where the store comes in.

Create an index.js file in the store directory :

```
import Vuex from 'vuex';

const store = () => {
  return new Vuex.Store({
    state: {},
    mutations: {}
  })
}

export default store
```

In the state, add a liked array, like you did in the main page.

```
liked: [false, false, false]
```

You then have to add two methods (or mutation in a store), one to add a like, one to delete a like. Syntax in a store is :

```
mutations: {
  addLike(state, index) {
    state.liked[index] = true;
  }
}
```



Don't forget to add the delLike() function in the store !

Now that our store is set, we are going to set our variables everytime we like or dislike an item. Go to your index.vue file, and go to your method.

Since we want to call the store in it, add the following line where you think it is best :

```
this.$store.commit('function', argument);
```

We now have our variables set in the store. To access it, change your liked data to the store data.

```
liked: this.$store.state.liked
```

We are set !

You need to change some things in the likes.vue page, but you should know what. Probably the same as the index.vue file.

All the cards are still on the likes page even if you didn't like them ? I know. Let's get to it.



It is really easy to only show a card if it is liked : remember the **if/else** we did earlier.



I will let you code this last part. You have everything you need ! But don't worry, you can ask any question you might have.