Cute cats web app (Django + VueJs) — Create VueJS views

Jrmy Follow

May 6, 2018 · 8 min read

Summary

Part I—Settings

Part II—Create VueJs views

Part III—Create Rest API with DRF

Part IV—Get datas from API with VueJs

Introduction

Previous part was about how to create and setting a new project with Python, django and VueJs. At the end of the post, we were able to display the generic template of vue-cli on the django localhost port.

By now, we will custom those template and display customs views.

For this post, assuming you are already familiar with VueJs.

Let's go !

Recap

Our current config is the following. The #app tap will contain every VueJs templates loading thanks to render_bundle. Render_bundle is tracking frontend built files

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    ...
  </head>
  <body>
    {% load render_bundle from webpack_loader %}

<div id="app">
       <app> </app>
    </div>

{% render_bundle 'main' %}
  </body>
</html>
```

Our VueJs view where App is currently the vue-cli generic template

Main.js

```
import Vue from 'vue'
import App from './App.vue'

new Vue({
  el: '#app',
  components: {
    'app': App
  }
})
```

Update current *src folder*

```
django_project
|_ ...
|_ src
   |_ assets
   |_ components
   |_ router
   |_ views
   |_ App.vue
   |_ Main.js
```

Before going further, we create a *views* folder in addition to the *components* folder. *Views folder* will contain all templates of our application. *Components folder* will contain all component whose compose views.

So, let's personalize the VueJs template.

Templates

See a little mockup of our useless application. Because in our connected world, more and more people using smartphone to consult website, I'm always styling template for mobile first.

The application will counts :

- One image of a cute cat. Choose your favorite
- two views *Home.vue* and *CuteCat.vue*
- three components : an header, a footer and a button

Before going further, we have to setting an optional thing. To styling my apps, I enjoy to use Sass in place of CSS. To make it possible, first install

node-sass and *sass-loader*.

```
//In the package.json directory
npm install --save-dev node-sass sass-loader
```

Then, in *build/webpack.base.conf.js* paste the code below in the *module.rules array.*

```
{
 test: /\.(css|sass)$/,
 loader: ['node-sass', 'sass-loader']
}
```

Now, we can start.

Components

Components compose our views. It's seems logical to begin by building them. Header and Footer will appear on each views not like the button.

Header

In this header we want two links, one to the home and an other to the cute cat page. Let's create the *header.vue* file in *components* folder.

— How to use *<router-link>*, a *vue-router* element

- *tag="li"* attribute is used to encapsulate the *<a>* tag in a *<li>* tag
- *:to="{name : <your-router-name>}"* attribute is define the destination of the link. This name is the one define in the *router/index.js* file.

```
<template>
  <header>
    <div id="logo">UA</div>
    <nav>
      <ul>
        <router-link tag='li' :to="{ name: 'home'}">
          <a>Home</a>
        </router-link>
        <router-link tag='li' :to="{ name:
'cutecat'}">
```

```
          <a>CuteCat</a>
        </router-link>
      </ul>
    </nav>
  </header>
</template>

<script>
export default{
  data () {
    return {

}
  }
}
</script>

<style scoped lang="sass">
  @import './../assets/styles/colors'

header
    display: flex
    justify-content: space-around
    align-items: center
    background-color: $green
    border-bottom: 2px solid darken($green, 30%)
    color: $white
    div#logo
      font-size: 200%
      width: 33%
    nav
      flex-grow: 2
      ul
        padding: 0
        display: flex
        justify-content: space-around
        list-style: none
        a
          text-decoration: none
          color: $white
</style>
```

Footer

We create a footer component to display, in this case, a sentence and a sitemap

```
<template>
  <footer>
    <p>Created by a guy who wants to stay anonymous
because this app is very useless</p>
    <div id="sitemaps">
      <h5>Sitemap</h5>
```

```
      <ul>
        <router-link tag='li' :to="{ name: 'home'}">
          <a>Home</a>
        </router-link>
        <router-link tag='li' :to="{ name:
'cutecat'}">
          <a>CuteCat</a>
        </router-link>
      </ul>
    </div>
  </footer>
</template>
```

```
<script>
export default{
  data () {
    return {
```

```
}
  }
}
</script>
```

```
<style scoped lang="sass">
@import './../assets/styles/colors'

footer
  position: fixed
  bottom: 0
  background-color: $green
  border-top: 2px solid darken($green, 20%)
  padding: 0 15px
  color: $white
  ul
    padding: 0
    display: flex
    justify-content: space-around
    list-style: none
    a
      text-decoration: none
      color: $white
</style>
```

Because the header and the footer appear in each views we can add them into the *App.vue* template. Remember, this one is the main template called to rendering user interface. Take a break…

… NO ! break in coding to check what the result is.

Before running the app, we have to update *App.vue* file. First, in the script part, we import our two new components, MyHeader and MyFooter, and define them into the components object.

They can be easily use into the template, like all other tags.

App.vue

```
<template>
  <div id="app">
    <MyHeader></MyHeader>
    <h1> Some content... </h1>
    <MyFooter></MyFooter>
  </div>
</template>
```

```
<script>
import MyHeader from '@/components/Header'
import MyFooter from '@/components/Footer'

export default {

components: {
    MyHeader,
    MyFooter
  }
}
</script>
```

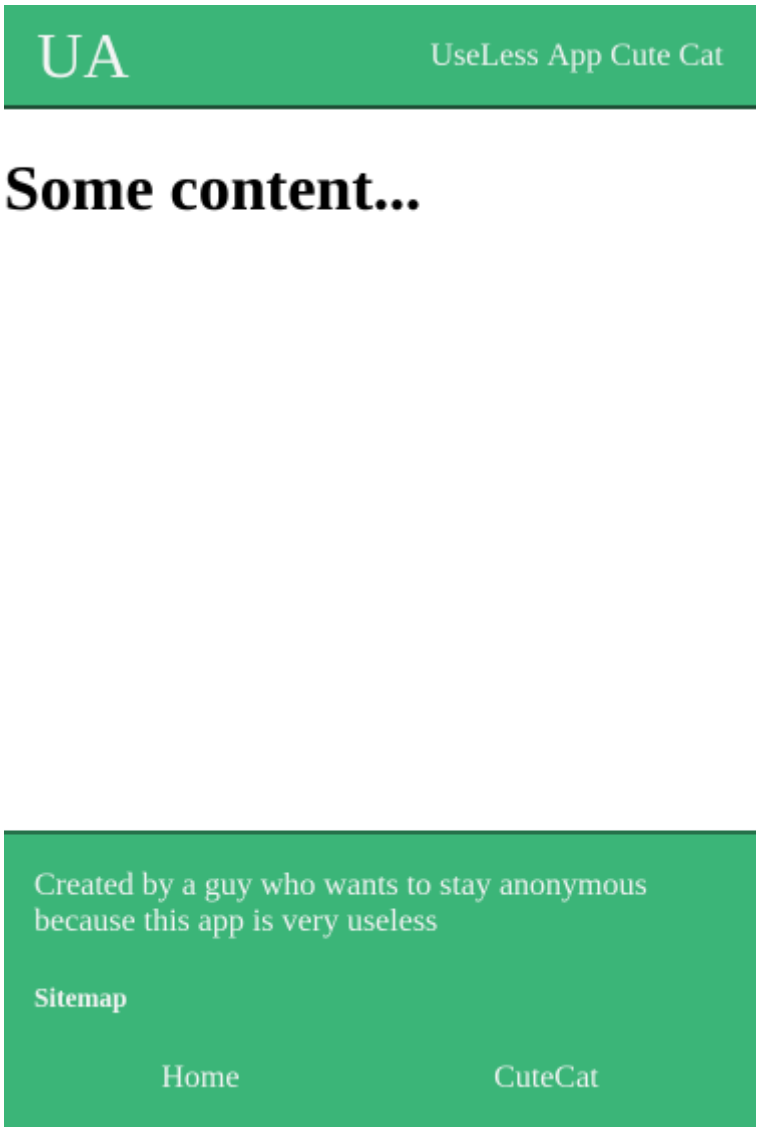
```
<style scoped lang="sass">
html, body
  margin: 0
  min-height: 100vh
</style>
```

We rebuild our frontend application and see what the python server render.

```
// rebuild
npm run install
```

```
// collect static files
python manage.py collectstatic
```

```
// lauch server
python manage.py runserver
```



As expected the page is containing the header and the footer styled as we defined. (*Be carefull, indentation is important in Sass. If you don't respect this rule, it will fail at the building step.)*

Unfortunately, the home and cute cat links aren't working yet, because we don't define routers yet. But before, we are going to create the button component.

Button component

This button must appaer in *Home.vue* but not in *CuteCat.vue,* so we can't call it in *App.vue* like we did before.

To be more generic we will define a prop named pathname. By now, each time we are using this component, we have to specify a pathname value. Pathname prop must be a string and his value, the name of the router we want to reach.

We also define a click event handler to execute a *goTo* method.

This one will redirect the user to the “pathname” route thanks to *this.$router.push({name: pathname})*

redirect.vue

```
<template>
  <button type="button" @click="goTo">My cute
cat</button>
</template>

<script>

export default {
  props:{
    pathname: {
      type: String,
      required: true
    }
  },
  methods: {
    goTo() {
      this.$router.push({name: this.pathname})
    }
  }
}
</script>

<style scoped lang="sass">
  @import './../../assets/styles/colors'

button
    width: 80%
    background-color: transparent
    color: $green
    border: 2px solid $green
    border-radius: 5px
    min-height: 50px
    text-transform: uppercase
    &:hover
      background-color: $green
      color: $white
      border-color: $white

</style>
```

We have created the three components. The header and the footer were placed in every template by adding them into *App.vue*. Now, we have to create two views and setting routers.

Views and Routers

In *views* folder create one file, named *Home.vue* and one, named *CuteCat.vue*. In *router* folder, update *index.js* to setting router views.

First, import the views Home and CuteCat. Then, we have to define routes. Routes is an array whose containing all route object.

To define a route, we must specify a

- Path *[string]*
- Name *[string]*—the one we used to specify the destination of <router-link>
- Component—the component to call when the path is reach

router/index.js

```
import Vue from 'vue'
import Router from 'vue-router'

import Home from '@/views/Home'
import CuteCat from '@/views/CuteCat'

Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '/',
      name: 'home',
      component: Home
    },
    {
      path: '/cutecate',
      name: 'cutecat',
      component: CuteCat
    }
  ]
})
```

With those setting, when path is:

- / → Home component will be display
- /cutecat → CuteCat component will be display

Time to add some content in our views.

Home.vue

According to the previous mockup, this view must contain a title and our custom button component.

First, import the button component and define him in the script part. Then, use the component name tag is the template to mount it. He specified the *pathname prop* as required, so, we have to assign a value. Just add an *pathname* attribute equal to “cutecat” (the route name to display the cute cat component)

Home.vue

```
<template>
  <section class="container">
    <h1> Welcome to your APP</h1>
    <redirectButton pathname="cutecat">
</redirectButton>
  </section>
</template>
```

```
<script>
  import redirectButton from
'@/components/buttons/redirect'


  export default {
    components: {
      redirectButton
    }
  }
</script>
```

```
<style scoped lang="sass">
  @import './../assets/styles/colors'
  @import './../assets/styles/global'
```

```
.container
    @include container-style
    h1
      color: $green
</style>
```

CuteCat.vue

Here, a photo of my cute cat ♥. It’s a very simple view !

CuteCat.vue

```
<template>
  <section class="container">
    <h1> Hi Guy </h1>
    <p>Welcome on an another page. Where lives the
cuttest cat</p>
    <img src="https://i.imgur.com/gdWIxn2.jpg"
alt="cute_cat"/>
  </section>
</template>
```

```
<script>
  export default {
    data () {
      return {
      }
    }
  }
</script>
```

```
<style scoped lang="sass">
  @import './../assets/styles/colors'
  @import './../assets/styles/global'
```

```
.container
    @include container-style
    h1
      color: $green
    p
      margin: 0
      color: lighten($green, 10%)
    img
      width: 100px
      height: 100px
</style>
```

Update App.vue

Now we have define our views and router we must update *App.vue* to synchronize content with path. It's really easy, we simply adding <router-view> tag.

App.vue

```
<template>
  <div id="app">
    <MyHeader></MyHeader>
    <router-view></router-view>
    <MyFooter></MyFooter>
```

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

```
<script>
import MyHeader from '@/components/Header'
import MyFooter from '@/components/Footer'
```

```
export default {
  name: 'App',
  components: {
    MyHeader,
    MyFooter
  }
}
</script>
```

To make it work, go to *Main.js* to define router.

Main.js

```
import Vue from 'vue'
import App from './App'
import router from '@/router'
```

```
Vue.config.productionTip = false
```

```
new Vue({
  el: '#app',
  router,
  template: '<App/>',
  components: { App }
})
```

Done !

We have defined three components, an header, a footer and a button. Then, created two views. *Home.vue*, to welcome users and *CuteCat.vue* to display a cute cat. We have linked everything together so our app should work properly. Let's see results.

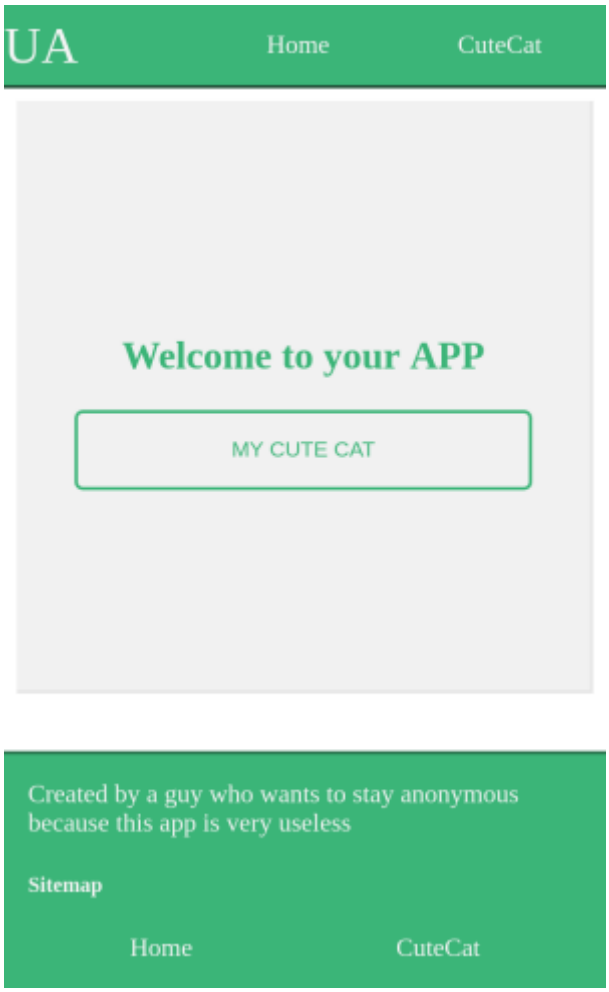
As usually, we have to rebuild the VueJs before to run Python server.

```
// rebuild
npm run install
```

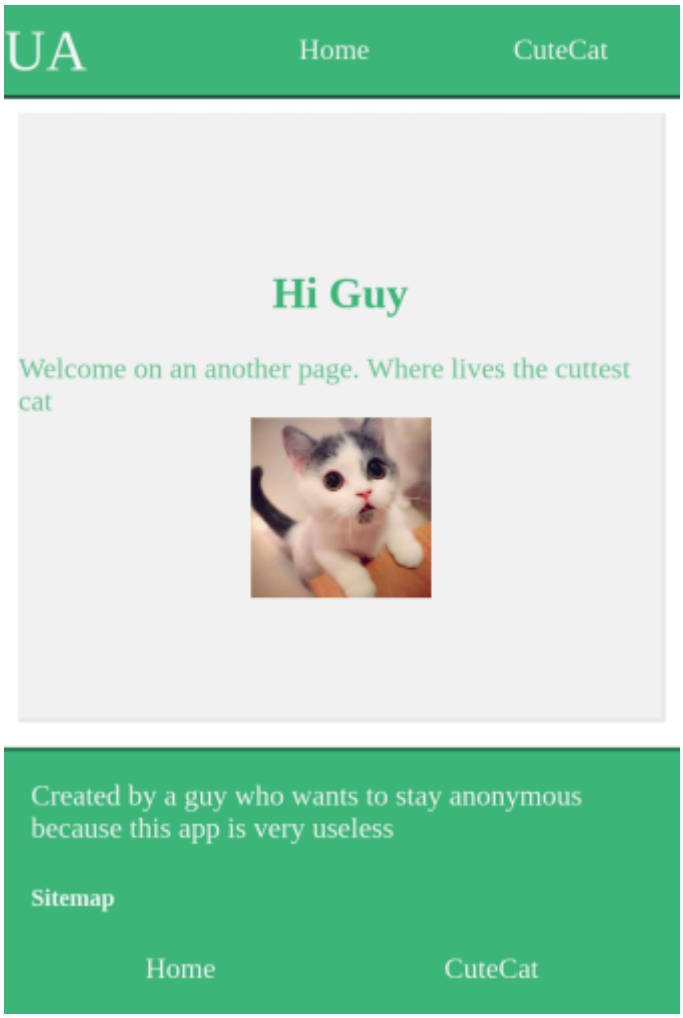
```
// collect static files
python manage.py collectstatic
```

```
// lauch server
python manage.py runserver
```

Open browser and go to localhost:8000, where the magic happen



If we click on the button my cute cat or on the link in the header, we will be directed to the *CuteCat* views.



TADAA !!

OMG he is so cute !

Good job every one, we have created a custom frontend. We know how to create and styling a component. We are able to use and call them in any views. We can navigate between views thanks to vue-router.

I agree, in this state our app is not very useful. Later, in a third part, we are going to create API endpoint to reach and render a cat list and see details about each cats by clicking on him.


Bye.