



UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA

Ciclo III

## Desarrollo de Software



### Capa de Presentación: Vue

14

**Jeisson Andrés Vergara Vargas**

Departamento de Ingeniería de Sistemas e Industrial

<http://colswu.unal.edu.co/~javergarav/>  
[javergarav@unal.edu.co](mailto:javergarav@unal.edu.co)

2020

©

# Objetivo de Aprendizaje

**Identificar** la jerarquía de componentes y el concepto de router en Vue.

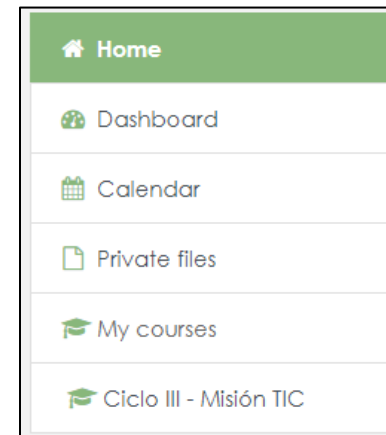
# Jerarquía de Componentes

## Componentes en Vue

En **Vue** las **aplicaciones** se dividen en **componentes**, un componente es una **pequeña parte** de la aplicación que por si misma implementa una **funcionalidad**.



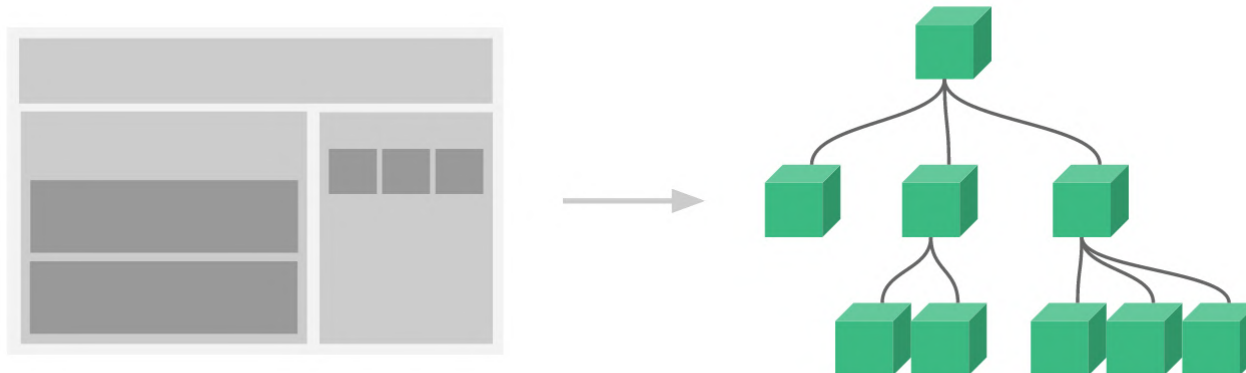
App



Componente

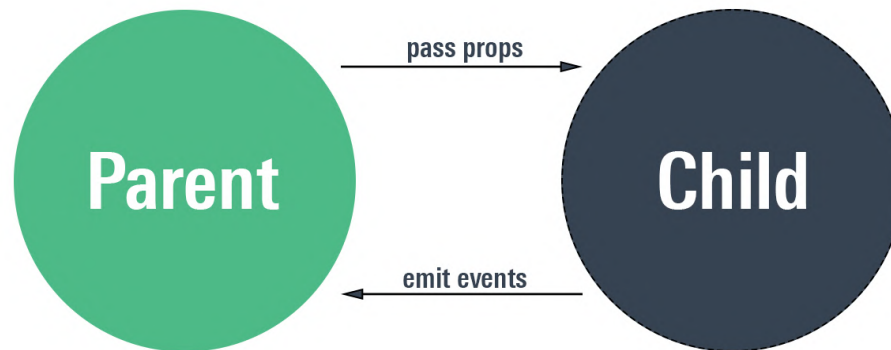
## Jerarquía de Componentes en Vue

Toda **aplicación** parte de un **componente** principal (**App.vue**), sobre este se **agregan** otros **componentes** (los cuales también tienen componentes agregados) y esto forma la **estructura** de la aplicación.



## Comunicación entre Componentes

Debido a la **estructura jerárquica** de los componentes, todas las relaciones serán del tipo **Padre-Hijo**, la comunicación entre estos se dará a través de **props** y **events**.



## Props

Los **props**, son **parámetros** que el componente **padre** le debe pasar al componente **hijo** al momento de **agregarlo**.

## Emit Events

Un componente **hijo** se comunica con su componente **padre** **emitiendo eventos** que este le provee.

## ¿Cómo agregar Componentes?

Los **componentes** se pueden **agregar** de 2 maneras:

1. Agregando una etiqueta (con el formato **HTML tradicional**) en el **template** del componente **padre**, donde el nombre de la **etiqueta** es el nombre del componente **hijo**:

`<NameComponent> </NameComponent>`

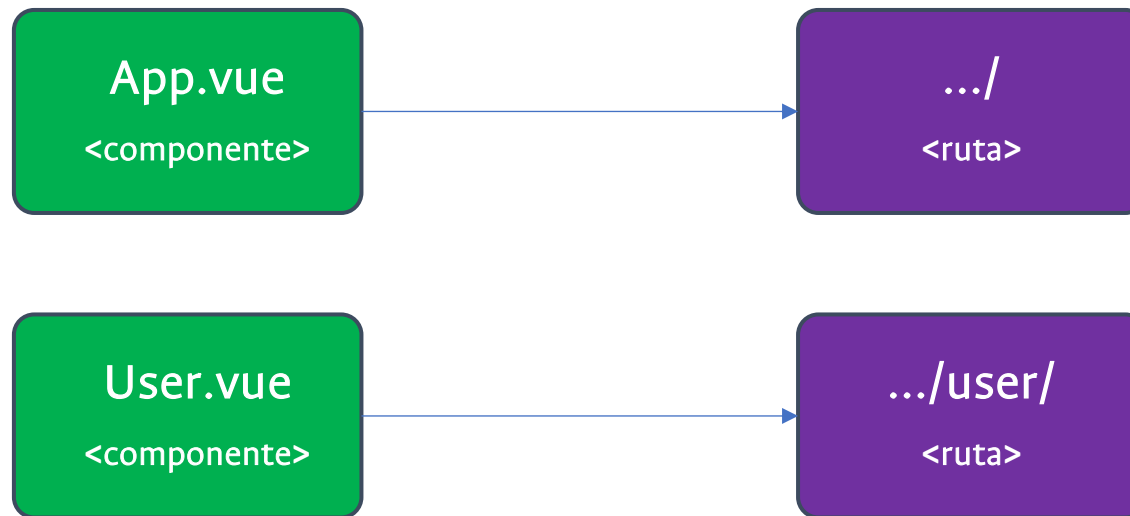
La **segunda** manera es con un **router** (se describirá más adelante).



# Routing

## ¿Qué es Routing?

El **Routing** es una herramienta que permite **asignarle una URL** a los **recursos** o **funcionalidades** (implementadas en **componentes**) de un app en **Vue**. Por ejemplo:



## ¿Para qué sirve el Routing?

El **Routing** tiene varios **beneficios**:

- Permite **describir** gran parte de la **estructura** jerárquica de la aplicación.
- **Facilita** la **navegación** y **acceso** a los recursos.
- Brinda **seguridad** sobre los **recursos**.
- **Maneja** de manera dinámica (basada en la ruta actual) las **inclusiones** de **componentes** hijos en los padre.

## ¿Como agregar Componentes?

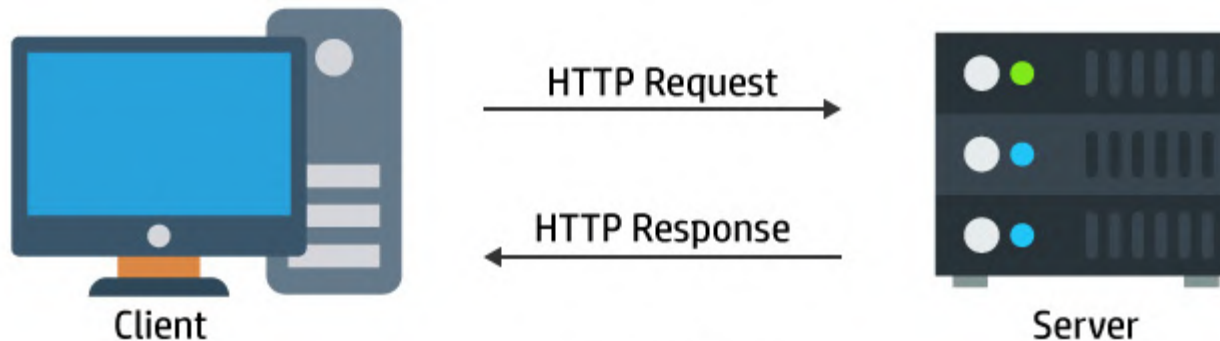
2. Implementando un **router**, el cual asigna a cada **componente** una **ruta**, y dependiendo de la **ruta actual**, carga el componente **hijo** en una etiqueta de la forma:

```
<router-view> </router-view>
```

# Comunicación con la Capa Lógica

## Comunicación con la API

En este punto se puede implementar la **comunicación** entre la **capa de presentación** y la **capa lógica**.



## Comunicación con el API

Esta **comunicación** se realizará a través de un **canal HTTP**, provisto por la librería de Node.js **AXIOS**.

The logo for the Axios library, featuring a stylized blue 'A' followed by the word 'XIOS' in a bold, dark grey sans-serif font.

# Políticas CORS



## ¿Qué son las Políticas CORS?

Las **CORS** son un **mecanismo** que controla el **acceso** a una aplicación, dependiendo del **origen de las peticiones**.

En este **contexto** se debe agregar algunas líneas de código a la **capa lógica** (Back-End desarrollada en FastAPI), para permitir el acceso de las peticiones desde la **capa de presentación**.

## Añadiendo Políticas CORS a la API

En el archivo `main.py` de nuestra **API REST** añadir las siguientes líneas de código:

(Luego de instanciar la api, **api = FastAPI()**):

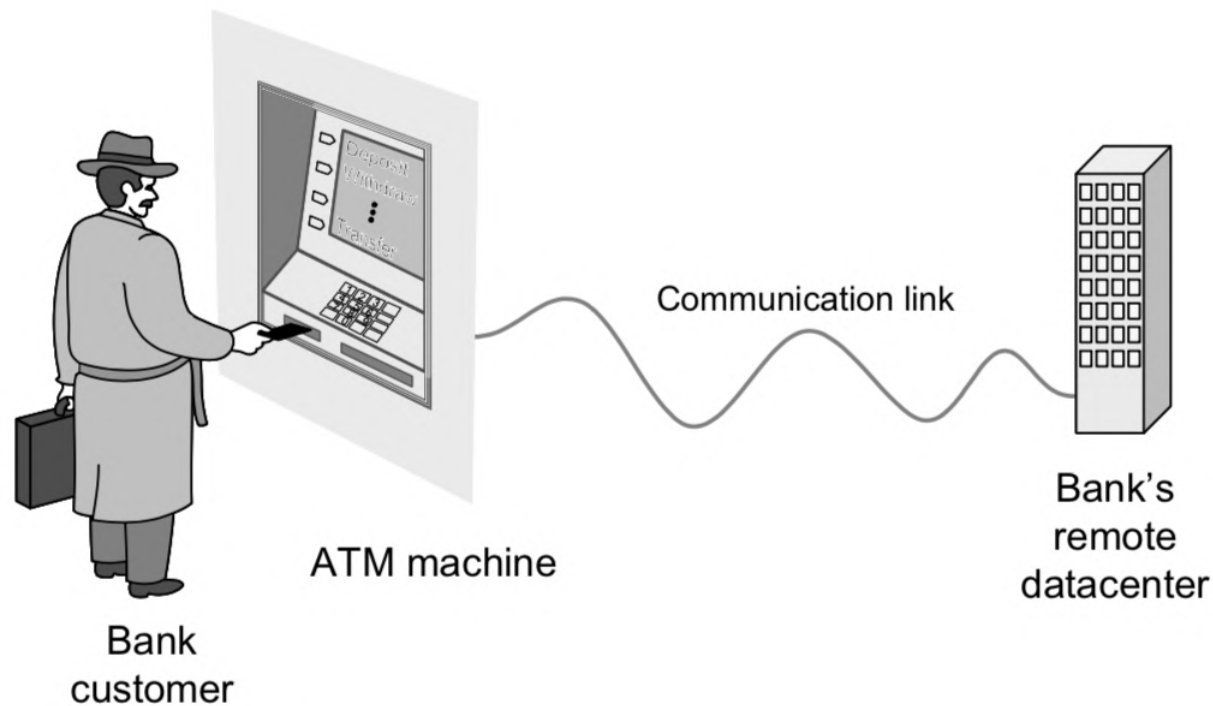
```
from fastapi.middleware.cors import CORSMiddleware

origins = [
    "http://localhost.tiangolo.com", "https://localhost.tiangolo.com",
    "http://localhost", "http://localhost:8080",
]

api.add_middleware(
    CORSMiddleware, allow_origins=origins,
    allow_credentials=True, allow_methods=["*"], allow_headers=["*"],
)
```

# Continuación del Ejemplo

## Software para un «ATM»



## Ejemplo

Para este acercamiento, se crearán **2 componentes** los cuales se añadirán al **componente principal** y serán controlados por un **router**.

**Componente User:** mostrará un mensaje de bienvenida al usuario.

**Componente User Balance:** a través de una **petición** con AXIOS se consultará a la **capa lógica** el saldo del usuario **precargado** y se mostrará la **información**.

# Resultado Esperado

## Componente User:



# Resultado Esperado

## Componente User Balance:



# Instalaciones



## Instalaciones Necesarias

Antes de iniciar, se deben instalar 2 paquetes: **axios** y **router**. Ejecutar los siguientes comandos en la **raíz del proyecto**:

```
npm install --save axios
```

```
npm install --save vue-router
```

# Implementación del Router

## Implementación del Router

En la **carpeta src**, al mismo nivel de **App.vue**, crear un archivo con el nombre **router.js**. No preocuparse por los archivos faltantes:

En este **archivo** se debe agregar el siguiente **código** – **Parte 1**:

```
import vueRouter from 'vue-router'  
  
import User from './components/User'  
import UserBalance from './components/UserBalance'  
import App from './App'
```

## Implementación del Router

En el **archivo router.js** se debe agregar el siguiente **código** – **Parte 2**:

```
const router = new vueRouter({  
  mode: 'history',  
  base: __dirname,  
  routes: [  
    {  
      path: '/',  
      name: "root",  
      component: App  
    },  
  ],  
});
```

## Implementación del Router

En el **archivo router.js** se debe agregar el siguiente **código** – **Parte 3**:

```
{
  path: '/user/:username',
  name: "user",
  component: User
},
{
  path: '/user/balance/:username',
  name: "user_balance",
  component: UserBalance
},
]
})
export default router
```

## Añadiendo el Router

Ahora se debe añadir el **router** a la aplicación. Para esto se debe editar el código de **main.js**:

Código de **main.js** - Parte 1:

```
import Vue from 'vue'
import App from './App'
import vueRouter from 'vue-router'

import router from './router'
Vue.use(vueRouter)
```

## Añadiendo el Router

Código de **main.js** - Parte 2:

```
Vue.config.productionTip = false

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

# Implementación del Componente User



## Implementando el Componente User

En la carpeta **components** se construye un archivo **User.vue**, el cual tendrá el siguiente código:

**Código** del archivo **User.vue** – Parte 1 (Template):

```
<template>
  <div id="User">
    <h2>Hola <span> {{username}}, </span> ¡Bienvenido!</h2>
  </div>
</template>
```

## Implementando el Componente User

Código del archivo **User.vue** – Parte 2 (Script):

```
<script>
  export default {
    name: "User",
    data: function(){
      return {
        username: "none"
      }
    },
    created: function(){
      this.username = this.$route.params.username
    }
  }
</script>
```

## Implementando el Componente User

Código del archivo **User.vue** – Parte 3 (Style):

```
<style>
  #User{
    width: 100%;
    height: 100%;

    display: flex;
    justify-content: center;
    align-items: center;
  }
```

## Implementando el Componente User

Código del archivo **User.vue** – Parte 4 (Style):

```
#User h2{  
  font-size: 50px;  
  color: #283747;  
}  
  
#User span{  
  color: crimson;  
  font-weight: bold;  
}  
</style>
```

# Implementación del Componente UserBalance

## Implementando el Componente **UserBalance**

En la carpeta **components** se construye un archivo **UserBalance.vue**, el cual tendrá el siguiente código:

**Código** del archivo **UserBalance.vue** – Parte 1 (Template):

```
<template>
  <div id="UserBalance">
    <h2>{{username}}</h2>
    <h2>Tu saldo es: <span> {{balance}} COP </span> </h2>
  </div>
</template>
```

# Implementando el Componente **UserBalance**

Código del archivo **UserBalance.vue** – Parte 2 (Script):

```
<script>
import axios from 'axios';
export default {
  name: 'UserBalance',
  data: function () {
    return {
      username: "",
      balance: 0
    }
  },
}
```

## Implementando el Componente UserBalance

Código del archivo **UserBalance.vue** – Parte 3 (Script):

```
created: function(){  
  this.username = this.$route.params.username  
  let self = this  
  
  axios.get("http://127.0.0.1:8000/user/balance/" + this.username)  
    .then((result) => {  
      self.balance = result.data.balance  
    })  
    .catch((error) => {  
      alert("ERROR Servidor");  
    });  
}  
}  
</script>
```

En este punto se hace la conexión con la API



# Implementando el Componente **UserBalance**

Código del archivo **UserBalance.vue** – Parte 4 (Style):

```
<style>
  #UserBalance{
    width: 100%;
    height: 100%;

    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
  }
```

# Implementando el Componente **UserBalance**

Código del archivo **UserBalance.vue** – Parte 5 (Style):

```
#UserBalance h2{  
    font-size: 50px;  
    color: #283747;  
}  
  
#UserBalance span{  
    color: crimson;  
    font-weight: bold;  
}  
</style>
```

# Integrando Componentes

## Integrando Componentes

Para Integrar los componentes se debe **editar** el archivo **App.vue**.

**Código** del archivo **App.vue** – Parte 1 (Template):

```
<div class="main-component">  
  <router-view></router-view>  
</div>
```

En el template solo se editará esta parte.

## Integrando Componentes

Código del archivo **App.vue** – Parte 2 (Script):

```
methods: {  
  
  init: function(){  
    if(this.$route.name !== "user"){  
      let username = localStorage.getItem("current_username")  
      this.$router.push({name: "user", params:{username:username}})  
    }  
  },  
}
```

En la parte de métodos se agrega el método **init** – función asociada al botón init

## Integrando Componentes

Código del archivo **App.vue** – Parte 3 (Script):

```
getBalance: function(){  
    if(this.$route.name !== "user_balance"){  
        let username = localStorage.getItem("current_username")  
        this.$router.push({ name:"user_balance",  
                           params:{username:username}  
                           })  
    }  
},  
},
```

En la parte de métodos se agrega el método **getBalance** – función asociada al botón Saldo.

## Integrando componetes Componentes

Código del archivo **App.vue** – Parte 4 (Script):

```
beforeCreate: function(){  
  localStorage.setItem('current_username', 'camilo24')  
  localStorage.setItem('isAuth', true)  
  
  this.$router.push({name:"user",params:{username:'camilo24'}})  
}
```

La parte de Style se mantiene.

# Ejecución



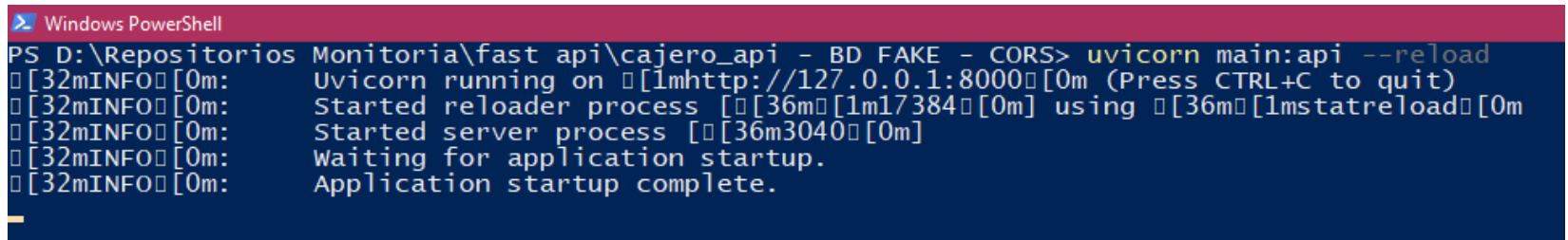
## Ejecutar la Capa Lógica

Recordando las clases pasada, ejecutar la **API** construida en **FastAPI**, con las modificaciones de las políticas **CORS** realizadas.

En la **raíz** del **proyecto** de FastAPI:

```
uvicorn main:api --reload
```

Resultado:



```
Windows PowerShell
PS D:\Repositorios\Monitoria\fast api\cajero_api - BD FAKE - CORS> uvicorn main:api --reload
[32mINFO[0m: Uvicorn running on [1mhttp://127.0.0.1:8000[0m (Press CTRL+C to quit)
[32mINFO[0m: Started reloader process [36m[1m17384[0m] using [36m[1mstatreload[0m]
[32mINFO[0m: Started server process [36m3040[0m]
[32mINFO[0m: Waiting for application startup.
[32mINFO[0m: Application startup complete.
```

## Ejecutar la Capa de Presentación

Recordando la clase pasada, ejecutar la **APP** construida en **Vue**:

En la **raíz** del **proyecto** de Vue:

```
npm run start
```

Resultado:

```
npm
DONE Compiled successfully in 181ms
I Your application is running here: http://localhost:8080
```

# Resultado

## Resultado

Dirigirse a <http://localhost:8080> y observar el resultado:



## Referencias

- [Vue.js] (2020). Retrieved 6 December 2020, from <https://es.vuejs.org>