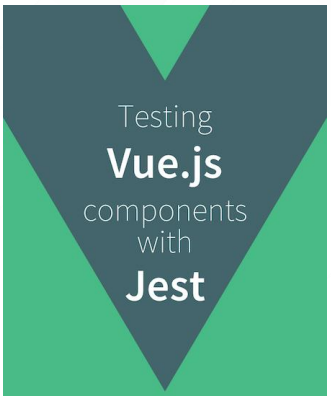


TESTING VUE CON JEST

Jest es un marco de prueba de JavaScript popular que viene con muchas ventajas para los desarrolladores. https://jestjs.io/

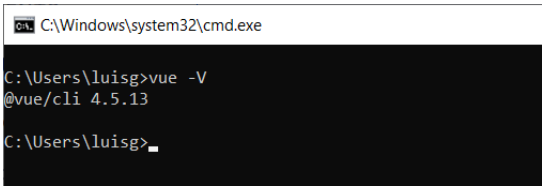
Prerrequisitos:

- Vue CLI. https://www.npmjs.com/package/jest & npm install -g @vue/cli

Para desarrollar el ejercicio debe cerciorarse de tener instalado la CLI de Vue, entonces, habrá una consola y tecle el siguiente comando

```
vue -V
```

En consecuencia, debe aparecer la siguiente información por consola

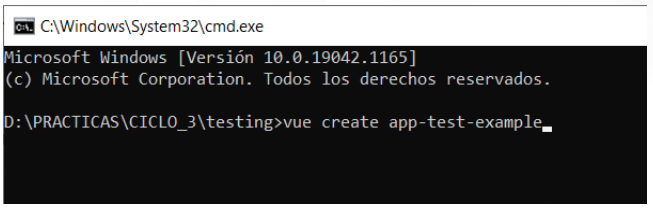


A continuación, vamos a realizar la actividad paso a paso.

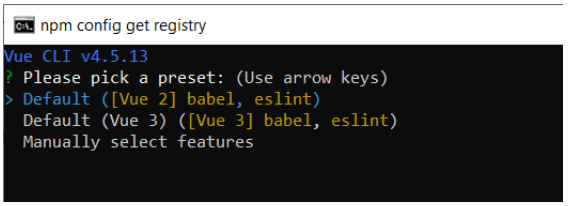
Configuración e instalación de dependencias

Inicialmente, crearemos una nueva app Vue llamada app-test-example. Para ellos nos ubicaremos en una carpeta que hallamos destinado para el proyecto y ejecutamos el siguiente comando en la consola, a saber:

```
vue create app-test-example
```



Seguidamente, elija el ajuste preestablecido predeterminado cuando se le solicite (presione la tecla Intro).



Después de eso, ejecute el siguiente comando para agregar nuestras dependencias de prueba (@vue/cli-plugin-unit-jesty @vue/test-utils):

```
npm install @vue/cli-plugin-unit-jest @vue/test-utils
```

A continuación, abra su proyecto desde su editor de código y modifique el archivo “package.json” de su proyecto para tener una entrada en la “scripts” que diga "test": "jest". Su “package.json” se verá de la siguiente forma:

Luego, cree el archivo “jest.config.js” en la ruta del proyecto /app-test-example/ y agregue dentro del archivo el siguiente contenido:

```
module.exports = {
  preset: "@vue/cli-plugin-unit-jest",
};
```

Codificar una aplicación simple

A continuación, desarrollaremos una pequeña aplicación para llevar a cabo pruebas. Mientras, eliminemos el directorio, src/components.

Seguidamente, modifiquemos el archivo “App.vue” agregando el siguiente código:

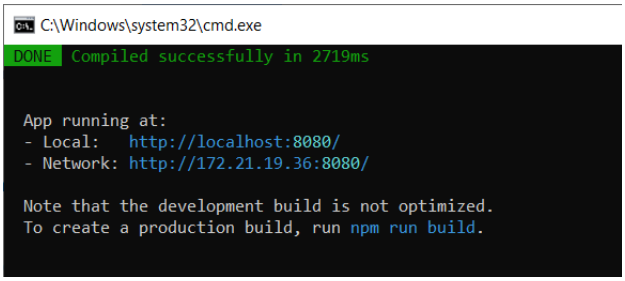
```
<template>
  <div id="app">
    <div>
      <h3>Let us test your arithmetic.</h3>
      <p>What is the sum of the two numbers?</p>
      <div class="inline">
        <p>{{ x1 }} + {{ x2 }} =</p>
        <input v-model="guess" />
        <button v-on:click="check">Check Answer</button>
      </div>
      <button v-on:click="refresh">Refresh</button>
      <p>{{ message }}</p>
    </div>
  </div>
</template>

<script>
export default {
  name: "App",
  data() {
    return {
      x1: Math.ceil(Math.random() * 100),
      x2: Math.ceil(Math.random() * 100),
      guess: "",
      message: "",
    };
  },
  methods: {
    check() {
      if (this.x1 + this.x2 === parseInt(this.guess)) {
        this.message = "SUCCESS!";
      } else {
        this.message = "TRY AGAIN";
      }
    },
    refresh() {
      this.x1 = Math.ceil(Math.random() * 100);
      this.x2 = Math.ceil(Math.random() * 100);
    },
  },
};
</script>

<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
```

```
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
.inline * {
  display: inline-block;
}
img {
  height: 350px;
}
</style>
```

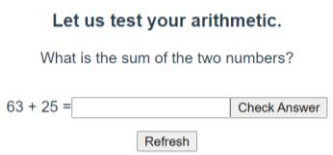
Luego, ejecute desde el directorio raíz de su proyecto el siguiente comando de consola:

```
npm run serve
```

Ahora, copie la dirección web (http://localhost:8080/) que le proporciona la consola para observar su aplicación desde el navegador web:



Su aplicación, debe verse de la siguiente forma:



Probando la aplicación con Jest

Para continuar, cree una carpeta llamada "__tests__" en el directorio raíz de su proyecto. Seguidamente, dentro "__tests__" cree un archivo llamado "app.spec.js". De forma predeterminada, jest capturará cualquier archivo de prueba (buscando de forma recursiva a través de carpetas) en su proyecto que tenga el nombre *.spec.jso *.test.js.

En la parte superior de “app.spec.js” vamos a importar lo siguiente “@vue/test-utils”, así como nuestro propio componente “App.js”

```
import { mount } from "@vue/test-utils";
import App from "./../src/App.vue";
```

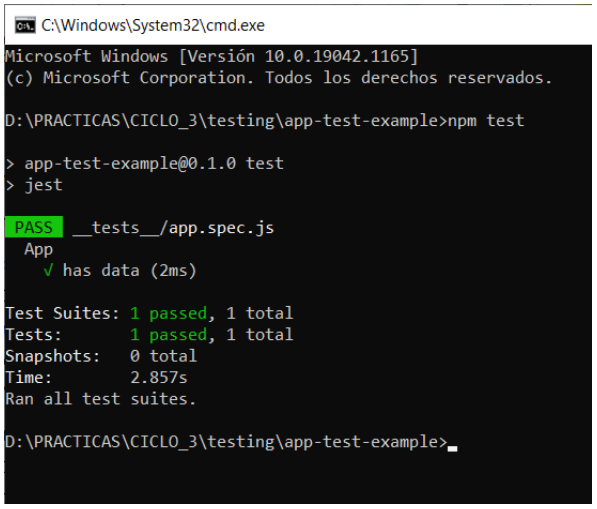
A continuación, escribamos nuestra primera prueba.

```
describe("App", () => {
  // Inspect the raw component options
  it("has data", () => {
    expect(typeof App.data).toBe("function");
  });
});
```

Seguidamente, ejecutemos el siguiente comando de consola:

```
npm test
```

Se debe observar la siguiente imagen:

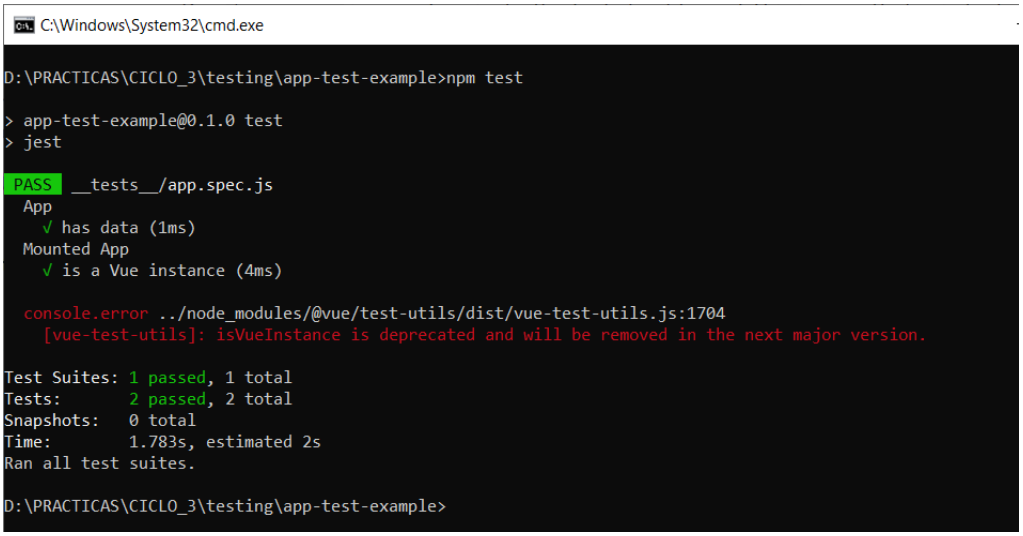


Esta es una prueba bastante básica y verifica si la data en nuestro componente es una función.

Ahora, agreguemos otro bloque “describe” a nuestro archivo de prueba.

```
describe("Mounted App", () => {
  const wrapper = mount(App);

  test("is a Vue instance", () => {
    expect(wrapper.isVueInstance()).toBeTruthy();
  });
});
```

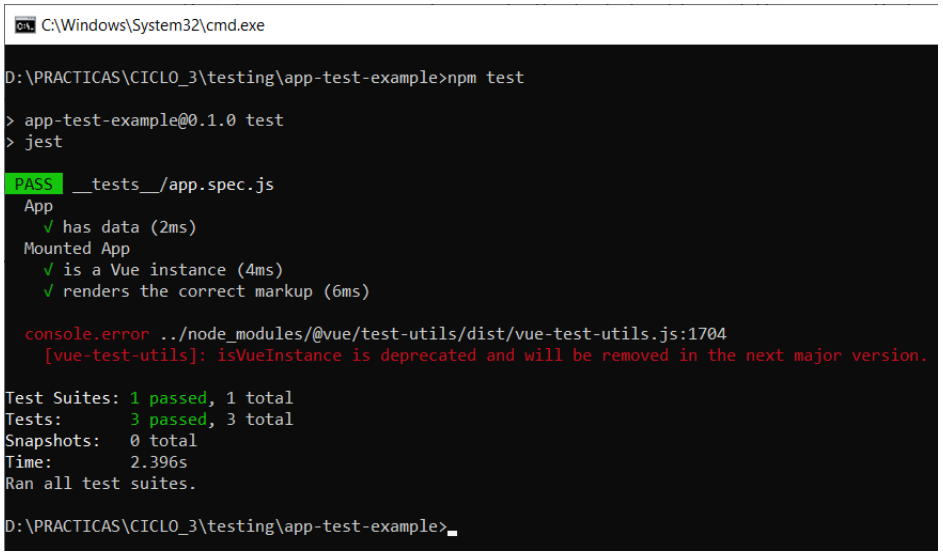
Esta vez estamos montando el componente, lo que nos devuelve un wrapper (Un contenedor es una instancia simulada de Vue). Podemos usarlo para validar si ciertos valores están presentes usando la función de Jest “expect”. El resultado de la prueba una vez escrito de nuevo el comando “npm test” es el siguiente:



Ahora, escribamos una prueba como esta:

```
describe("Mounted App", () => {
  const wrapper = mount(App);

  it("renders the correct markup", () => {
    expect(wrapper.html()).toContain("What is the sum of the two numbers?");
  });
});
```

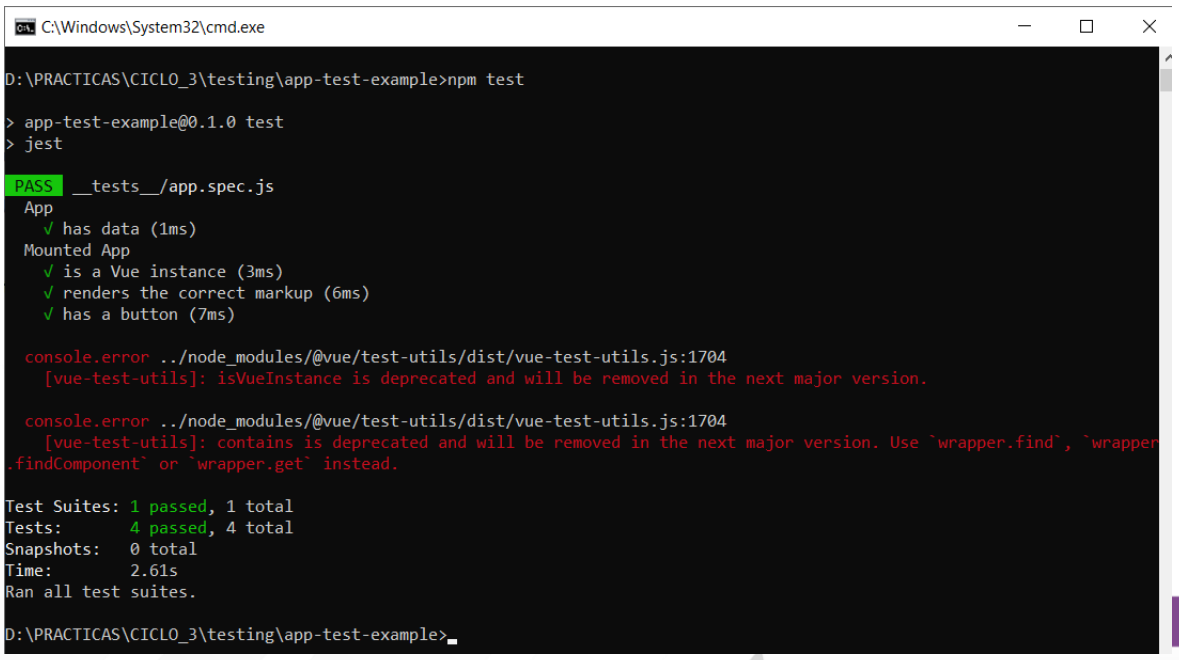
Veamos el resultado:



Y esta:

```
describe("Mounted App", () => {
  const wrapper = mount(App);

  it("has a button", () => {
    expect(wrapper.contains("button")).toBe(true);
  });
});
```

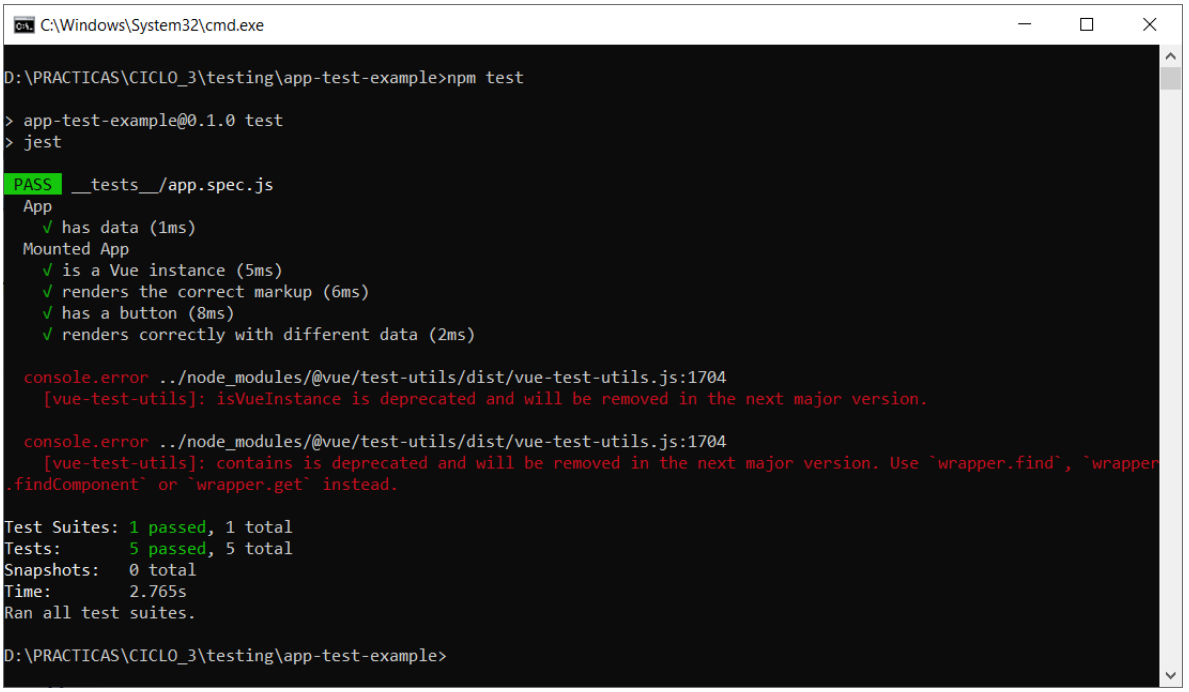
La salida será la siguiente:



Ahora, escribamos algunas pruebas para la funcionalidad más específica de Vue de la aplicación.

```
describe("Mounted App", () => {
  const wrapper = mount(App);

  it("renders correctly with different data", async () => {
    wrapper.setData({ x1: 5, x2: 10 });
    await wrapper.vm.$nextTick();
    expect(wrapper.text()).toContain("10");
  });
});
```

El resultado:



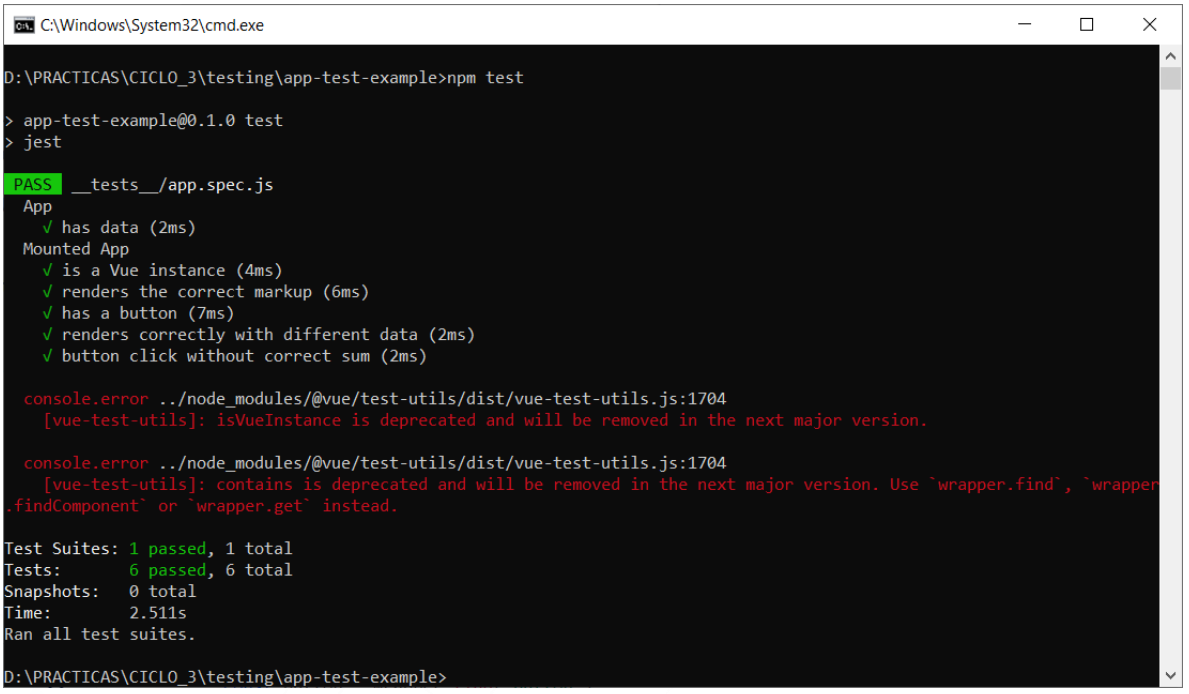
setDatale permite configurar los datos del componente. Dado que esas variables se inicializaron en data, son reactivas. Cuando estamos burlando nuestro componente, sin embargo, hay que llamamos “$nextTick()” en el “wrapper.vm”, que es el componente subyacente del “wrapper” . Entonces, podemos encontrar que nuestras propiedades reactivas se actualizan.

Finalmente, vamos a probar si nuestra aplicación da el resultado correcto de acuerdo con lo que pretendemos que haga: ¡probar la suma!

```
describe("Mounted App", () => {
  const wrapper = mount(App);

  it("button click without correct sum", () => {
    expect(wrapper.vm.message).toBe("");
    const button = wrapper.find("button");
    button.trigger("click");
    expect(wrapper.vm.message).toBe("TRY AGAIN");
  });
});
```

“wrapper.find” devuelve un “wrapper” para el elemento “botón” (aunque hay 2 botones en la página, el que se quiere probar es el primer botón de la página para que sea capturado). “x1” y “x2” se establecen a partir de nuestra prueba anterior. Pero “guess”, la variable que está conectada al elemento de entrada a través de “v-model”, no lo está. Entonces, cuando se hace clic en el botón de envío, no se ha ingresado la suma correcta. Por lo tanto, esperamos un “message” que sea “TRY AGAIN”.

La salida debe ser la siguiente:



Finalmente, el código de “app.spec.js” debe quedar de la siguiente manera:

```
import { mount } from "@vue/test-utils";
import App from "./../src/App.vue";

describe("App", () => {
  // Inspect the raw component options
  it("has data", () => {
    expect(typeof App.data).toBe("function");
  });
});

describe("Mounted App", () => {
  const wrapper = mount(App);

  test("is a Vue instance", () => {
    expect(wrapper.isVueInstance()).toBeTruthy();
  });
});

describe("Mounted App", () => {
  const wrapper = mount(App);

  it("renders the correct markup", () => {
    expect(wrapper.html()).toContain("What is the sum of the two numbers?");
  });
});

describe("Mounted App", () => {
  const wrapper = mount(App);

  it("has a button", () => {
    expect(wrapper.contains("button")).toBe(true);
  });
});

describe("Mounted App", () => {
  const wrapper = mount(App);

  it("renders correctly with different data", async () => {
    wrapper.setData({ x1: 5, x2: 10 });
    await wrapper.vm.$nextTick();
    expect(wrapper.text()).toContain("10");
  });
});

describe("Mounted App", () => {
  const wrapper = mount(App);

  it("button click without correct sum", () => {
    expect(wrapper.vm.message).toBe("");
    const button = wrapper.find("button");
    button.trigger("click");
    expect(wrapper.vm.message).toBe("TRY AGAIN");
  });
});
```