nstallation

```
npm install --save vue-full-calendar
```

Or for Vue 1.x users

```
npm install --save vue-full-calendar@0.0.3
```

Installing the plugin will globally add the `full-calendar` component to your project.

```
//main.js

import FullCalendar from 'vue-full-calendar'

Vue.use(FullCalendar)
```

But you can also import the standalone component to add locally or for more complex installations.

```
// foo.vue

import { FullCalendar } from 'vue-full-calendar'

export default {

  components: {

    FullCalendar,

  },

}
```

jQuery

Please note that fullcalendar depends on jQuery, but you won't need to add it to your project manually, fullcalendar will handle this for you automatically if jQuery is not detected.

CSS

As of version 2.0, we have removed the automatic import of the fullcalendar.css, you will need to explicitly import this css file in your project.

```
import 'fullcalendar/dist/fullcalendar.css'
```

Example App

I have created a simple Vue 2 webpack application as an example/playgroundhttps://github.com/BrockReece/vue-fullcalendar-example

or try out this Code Sandbox

Scheduler

For those wanting to use the scheduler plugin, this Code Sandbox shows you a full working example.

Basic Usage

You can pass an array of fullclendar objects through the props

```
<full-calendar :events="events"></full-calendar>

...

<script>

...

  data() {

    return {

      events: [

        {

            title  : 'event1',

            start  : '2010-01-01',

        },
```

```
        {
            title  : 'event2',
            start  : '2010-01-05',
            end    : '2010-01-07',
        },
        {
            title  : 'event3',
            start  : '2010-01-09T12:30:00',
            allDay : false,
        },
      ]
    }
  }
...
</script>
```

More event options can be found
at http://fullcalendar.io/docs/event_data/Event_Object/

Using a JSON Feed

```
<full-calendar :event-sources="eventSources"></full-calendar>

...

<script>

...

  data() {
```

```
    return {
      eventSources: [
        {
          events(start, end, timezone, callback) {
            self.$http.get(`/myFeed`, {timezone: timezone}).th
en(response => {
              callback(response.data.data)
            })
          },
          color: 'yellow',
          textColor: 'black',
        },
        {
          events(start, end, timezone, callback) {
            self.$http.get(`/anotherFeed`, {timezone: self.tim
ezone}).then(response => {
              callback(response.data.data)
            })
          },
          color: 'red',
        },
      ]
    }
```

```
  }

...

</script>
```

Custom Config

You can pass any custom options through to fullcalendar by using the `config` prop, this includes extra event handlers.

```
<full-calendar :events="events" :config="config" />

...

<script>

...

  data() {

    return {

      events: [],

      config: {

        weekends: false,

        drop(...args) {

          //handle drop logic in parent

        },

      },

    }

  },

...

</script>
```

Locale

You can set the language of your calendar by importing the corresponding locale file and setting the `locale` key in the config prop. For example, to set up the Calendar in French...

```
<full-calendar :events="events" :config="config" />

...

<script>

import 'fullcalendar/dist/locale/fr'

...

  data() {

    return {

      events: [],

      config: {

        locale: 'fr',

      },

    }

  },

...

</script>
```

Code Sandbox

Note: You won't need to set the locale config key if your app only imports a single locale file

Further Props

You can edit the look and feel of fullcalendar by passing through extra props. These all have sensible defaults

- **header** - [obj] - docs
- **defaultView** - ['agendaWeek'] - docs
- **editable** - [true] - docs
- **selectable** - [true] - docs
- **selectHelper** - [true] - docs
- **config** - [true] - Pass your own custom config straight through to fullcalendar

Methods

Sometimes you may need to manipulate the Calendar from your parent component, you can use `fireMethod` for this. This works with anything in the Fullcalendar docs suffixed with `(method)` and it will dynamically handle as many arguments as needed.

```
<full-calendar :events="events" ref="calendar" />

...

<script>

...

  data() {

    return {

      events: [],

    }

  },



  methods: {

    next() {

      this.$refs.calendar.fireMethod('next')
```

```
    },

    changeView(view) {

      this.$refs.calendar.fireMethod('changeView', view)

    },

  },

...

</script>
```

Events and Hooks

Emitted

- **event-selected(event, jsEvent, view)** - Triggered on eventClick()
- **event-drop(event)** - Triggered on eventDrop()
- **event-resize(event)** - Triggered on eventResize()
- **event-created(event)** - Triggered on select()
- **event-receive(event)** - Triggered on eventReceive()
- **event-render(event)** - Triggered on eventRender()
- **view-render(view, element)** - Triggered on viewRender()
- **day-click(date, jsEvent, view)** - Triggered on dayClick()

You can listen for these events using the following markup

```
<full-calendar :event-sources="eventSources" @event-
selected="eventSelected"></full-calendar>
```

Listens on

- **render-event(event)** - Adds a new event to calendar
- **remove-event(event)** - Removes event from calendar
- **rerender-events()** - Rerenders events to reflect local changes
- **refetch-events()** - Makes another JSON call to event sources
- **reload-events()** - Removes all events and adds all events in this.events

You can trigger these events in the parent component like so...

```
<full-calendar ref="calendar" :event-
sources="eventSources"></full-calendar>
```

```
...
<script>
...
  methods: {
    refreshEvents() {
      this.$refs.calendar.$emit('refetch-events')
    },
  }
...
</script>
```