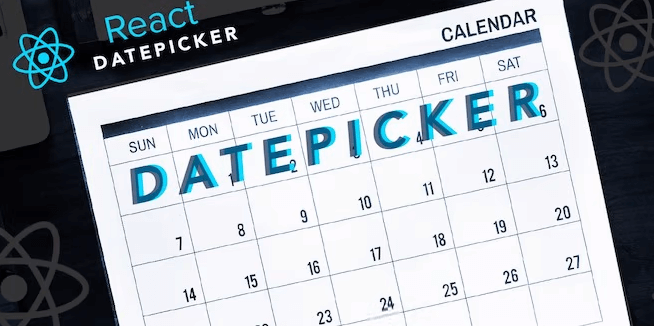
|  |  |
| --- | --- |
| **ch45.ReactJS Date Picker** | **Date: 22-02-2022** |

**Topics**

React Props Validation,

# React Props Validation

# React Date Picker



## **Introduction**

We often come across various forums and websites where we need to enter our date of birth or select a date for an appointment of any cause, and a calendar-like icon appears right in front of our eyes. How and where does that pop up from? It must be some specific and dynamically powered calendar that smartly recognizes and presents to us to select dates of our desire. This is where React Date Picker hops in. In this tutorial, we would be learning all such methods from scratch to advanced level on how it is implemented in React and how it can be customized. Let's proceed with the discussion.

The [React](https://www.javatpoint.com/reactjs-tutorial) Date Picker is a helpful and abundant Component used to display dates using the calendar dialog format. There are usually a lot a Date picker options available these days. All of them can have different technical aspects and applications. In this tutorial, we are specifically going to deal with React Date Picker. For that, we need to have a package for it that displays time and date. For better understanding, we will make an application that would help us understand React Date Picker better. But, before that, let's install it as shown in the below steps.

## **Installation**

To install Date Picker for our React application, we need to have [Node.js](https://www.javatpoint.com/nodejs-tutorial) pre-installed on our system. Although it is not important to always have node modules, it is highly recommended to download them so that all the dependencies are served efficiently. Therefore, the command to install React Date Picker is given below.

The package can be installed via **npm:**

**3:00:00/04:58**

28.2M

580

HTML Tutorial

1. npm install react-datepicker --save

Or via **Yarn:**

1. yarn add react-datepicker

There might arise a need to install React and its Proptypes individually because, without these dependencies, it is impossible to build a React Date Picker. Also, we may need to use [CSS](https://www.javatpoint.com/css-tutorial) from the external packages so that the Date Picker looks visually great. To get started with our application, we need to import the React Date picker in our main file, and we need to keep checking it through React view.

1. **import** React, { useState } from "react";
2. **import** DatePicker from "react-datepicker";
4. **import** "react-datepicker/dist/react-datepicker.css";
6. // CSS Modules, react-datepicker-cssmodules.css
7. // import 'react-datepicker/dist/react-datepicker-cssmodules.css';
9. **const** Example = () => {
10. **const** [startDate, setStartDate] = useState(**new** Date());
11. **return** (
12. <DatePicker selected={startDate} onChange={(date) =>
13. setStartDate(date)} />
14. );
15. };

## **React Datepicker example**

Assuming that our system is configured with all the installation requirements and dependencies, we will learn how to create a React application from scratch. This implementation is completely based on the application of React datepicker.

In the discussion above, we assume that we have installed React and PropTypes independently since these dependencies are not incorporated in the package itself. To proceed with the methods of building our application, we need to follow the below-given steps.

1. npx create-react-app react-datepicker-app

Move inside the project folder with the command.

1. cd react-datepicker-app

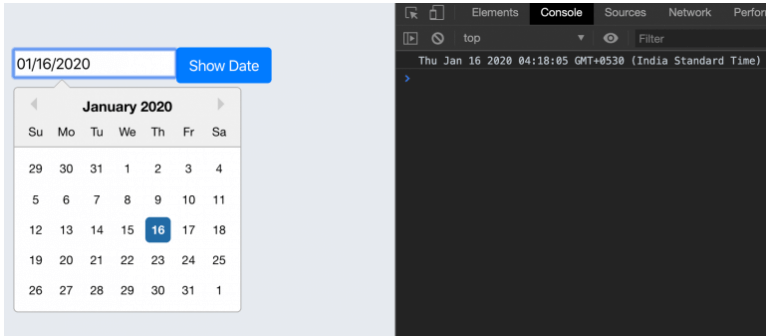
Start the React app.

1. npm start

After the Node engine is started, we can check our application through port number 3000 of the localhost. Also, we need to include the code snippet given below in our app.js file so that important components for React Date Picker are imported into our file.

1. // app.js
3. **import** React, { Component } from 'react';
4. **import** DatePicker from 'react-datepicker';
6. **import** "react-datepicker/dist/react-datepicker.css";
7. **import** 'bootstrap/dist/css/bootstrap.min.css';
9. **class** App **extends** Component {
11. constructor (props) {
12. **super**(props)
13. **this**.state = {
14. startDate: **new** Date()
15. };
16. **this**.handleChange = **this**.handleChange.bind(**this**);
17. **this**.onFormSubmit = **this**.onFormSubmit.bind(**this**);
18. }
20. handleChange(date) {
21. **this**.setState({
22. startDate: date
23. })
24. }
26. onFormSubmit(e) {
27. e.preventDefault();
28. console.log(**this**.state.startDate)
29. }
31. render() {
32. **return** (
33. <form onSubmit={ **this**.onFormSubmit }>
34. <div className="form-group">
35. <DatePicker
36. selected={ **this**.state.startDate }
37. onChange={ **this**.handleChange }
38. name="startDate"
39. dateFormat="MM/dd/yyyy"
40. />
41. <button className="btn btn-primary">Show Date</button>
42. </div>
43. </form>
44. );
45. }
47. }
49. export **default** App;

Our application would now look something like this.



In the above sample coded, we first imported the packages of **Datepicker** and [**Bootstrap**](https://www.javatpoint.com/bootstrap-tutorial) in the React template. Then, we defined it to bind the **handle change** and submit the Component in the event calendar. These events will automatically be triggered when a user submits or changes the Datepicker's input value values. Later, we initialized the Datepicker form state with a new **Date()** object inside the constructor. Lastly, we initiated the Datepicker with the below directive to add some properties to it.

1. **<DatePicker**
2. selected={ this.state.startDate }
3. onChange={ this.handleChange }
4. name="startDate"
5. dateFormat="MM/dd/yyyy"
6. **/>**

We can visualize the output in the form of a calendar-centric datepicker format. The above-given Datepicker adds customized properties to the React application components shown above. It allows us to pick the dates in the form of **months, days,** and **years**.

Furthermore, to customize the datepicker, we have various other methods, be it coloring the components or smartly applying the functions to pick up the dates. We can also customize them easily if we have HTML and CSS components related to the app.js file.

## **Localizing the Datepicker**

Another instance we are going to learn about is to localize the Datepicker. The Date Picker we are going to make highly depends on date-fns internationalization. This is because it is used to localize the elements that will be on display. If we need to use a locale except for the default en-US, we may need to import it to the project from the date-fns.

Moreover, English is the default locale which consists of 3 methods to set the locale.

**register local** (string, object): loads an imported locale object from date-fns.

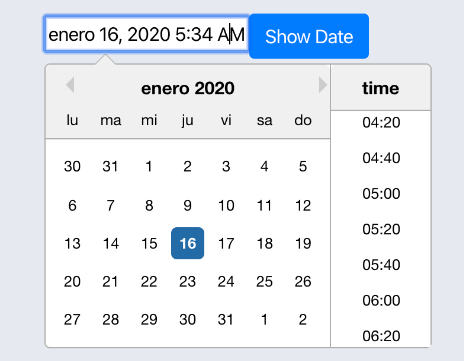
**Set default locale** (string): sets a registered locale as the default for all datepicker instances.

**getDefaultLocale:** returns a string showing the currently set default locale.

We can import these services to the locale of the calendar view using the below code snippet.

1. **import** { registerLocale, setDefaultLocale } from  "react-datepicker";
2. **import** es from 'date-fns/locale/es';
3. registerLocale('es', es)
4. <DatePicker
5. locale="es"
6. />

On importing and saving these locale services into our app.js file, our application will look like this.



In another instance, to change the locales, we need to primarily change the locale code or by using the internationalization of the date-fns so the code review can be supported.

1. setDefaultLocale('es')

## **Setting Calendar date range in Datepicker.**

We will learn how to implement the range functionality using **the minDate** and **maxDate property** in this step. To do so, we import the **addDays AP**I from the **date-fns** library to the top of our React component. It will add the specific number of days to the assigned date to set the range.

1. **import** addDays from 'date-fns/addDays'

The **addDays()** method usually takes two parameters:

**Date:** The date that needs to be updated.

**Amount:** The significant amount of days needed to be included.

We can easily set the date range from the current to the next seven days in the React calendar. This can be understood when we implement the **minDate** and **maxDate** methods in the example code shown below.

1. render() {
2. return (
3. **<form** onSubmit={ this.onFormSubmit }**>**
4. **<div** className="form-group"**>**
5. **<DatePicker**
6. selected={ this.state.startDate }
7. onChange={ this.handleChange }
8. showTimeSelect
9. timeFormat="HH:mm"
10. timeIntervals={20}
11. timeCaption="time"
12. dateFormat="MMMM d, yyyy h:mm aa"
13. minDate={new Date()}
14. maxDate={addDays(new Date(), 7)}
15. **/>**
16. **<button** className="btn btn-primary"**>**Show Date**</button>**
17. **</div>**
18. **</form>**
19. );
20. }

The full code snippet for our application after implementing all the steps given above is shown below.

1. **import** React, { Component } from 'react';
2. **import** DatePicker from 'react-datepicker';
3. **import** addDays from 'date-fns/addDays'
4. **import** "react-datepicker/dist/react-datepicker.css";
5. **import** 'bootstrap/dist/css/bootstrap.min.css';

8. **class** App **extends** Component {
10. constructor (props) {
11. **super**(props)
12. **this**.state = {
13. startDate: **new** Date()
14. };
15. **this**.handleChange = **this**.handleChange.bind(**this**);
16. **this**.onFormSubmit = **this**.onFormSubmit.bind(**this**);
17. }
19. handleChange(date) {
20. **this**.setState({
21. startDate: date
22. })
23. }
25. onFormSubmit(e) {
26. e.preventDefault();
27. console.log(**this**.state.startDate)
28. }
30. render() {
31. **return** (
32. <form onSubmit={ **this**.onFormSubmit }>
33. <div className="form-group">
34. <DatePicker
35. selected={ **this**.state.startDate }
36. onChange={ **this**.handleChange }
37. showTimeSelect
38. timeFormat="HH:mm"
39. timeIntervals={20}
40. timeCaption="time"
41. dateFormat="MMMM d, yyyy h:mm aa"
42. minDate={**new** Date()}
43. maxDate={addDays(**new** Date(), 7)}
44. />
45. <button className="btn btn-primary">Show Date</button>
46. </div>
47. </form>
48. );
49. }
51. }
52. export **default** App;

In the above example snippet, we have made **minDate** as a check-in component, and the user can only pick the date before today. We have used the date value for the check-out Component, which means that the user cannot pick a date before the check-in date.

In the next step, we will save the value of our state and define how each handle works. The main idea is to implement it by simply creating a state for each **check-in** and **check-out** date with the defined values and saving the data there. The only difference lies in the **minDate** method of the check-out component, as it is solely dependent on the check-in component. Thus, ensuring that everything is set in terms of values not before and not after, we can now easily select the dates and define the check-out.

## **Conclusion**

In this tutorial, we have been able to follow through a simple step-by-step guide on building a custom React Datepicker component that can be easily used as a replacement for the native [**HTML5**](https://www.javatpoint.com/html5-tutorial)**datepicker** input elements. We have learned how to set up the React application on priority because rendering the React components might seem complex for beginners, so setting up dependencies would always be preferred by a beginner. We also came across various examples to establish a crystal clear concept of using the components of Datepicker in our application. We also learned about the process of localizing the datepicker so that the date selection process doesn't cause a problem if they are selected for a specif duration on the calendar.

Although the custom datepicker created in this tutorial works as expected, it doesn't comply with all the advanced requirements for a datepicker element. Further improvements can be made, such as implementing the max and min via props, switching the input type from text to date, and establishing better accessibility improvement. We can also develop design methods for the datepicker that we implemented in this tutorial by importing Bootstrap packages and adding custom styling and properties of hovering to make it look more good.

Next Topic[React Helmet](https://www.javatpoint.com/react-helmet)