Building Timers in React: Stopwatch and Countdown

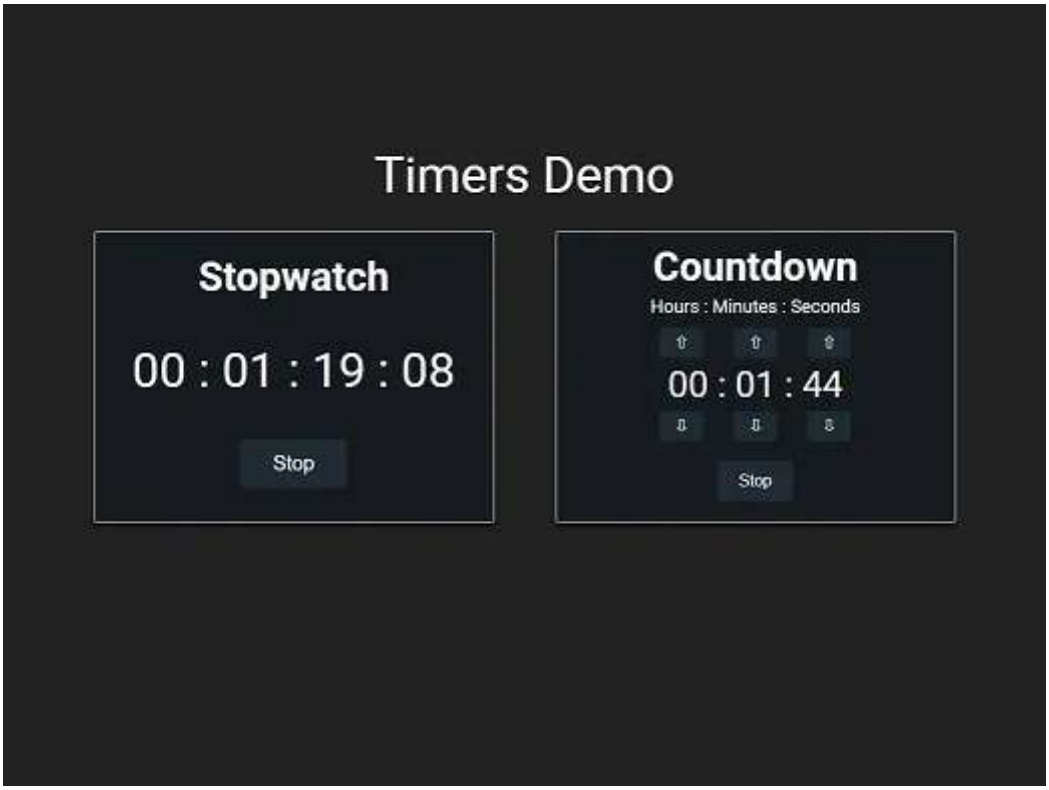
Peter Durham · Follow

10 min read · Feb 15, 2019

Stopwatch and Countdown Timers Tutorial

In this tutorial we will be using React to build a stopwatch timer and a countdown timer. Both timers will utilize intervals to keep track of time and can start, stop, resume, and reset.

The Github repo for this tutorial can be found here



Finished Stopwatch and Countdown Timers

Setup

To begin with, I will be using Create React App to build out the basic files we will need. Assuming you have Node installed (if not download it), enter the command:

```
npx create-react-app timers-demo
```

(or whatever name you choose for your application)

In order to simplify things, I will be deleting the following files in the project:

- `App.test.js`
- `index.css`
- `logo.svg`
- `serviceWorker.js`

If you deleted these files too, make sure remove unnecessary code `index.js`, which will now look like this:

`index.js`

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";

ReactDOM.render(<App />, document.getElementById("root"));
```

We will also clean up the imports from `App.js` as well as adding in our new components which we will soon build.

```
import React, { Component } from "react";

import Stopwatch from "./Stopwatch";
import Countdown from "./Countdown";

class App extends Component {
  render() {
    return (
```

```
        <div className="App">
          <div className="App-title">Timers Demo</div>
          <div className="Timers">
            <Stopwatch />
            <Countdown />
          </div>
        </div>
      );
    }
  }

  export default App;
```

Next, add two files in the src folder called `Stopwatch.js` and `Countdown.js`. These will be class based components so we can save our timer data. I will also import our `App.css` file into each for styling later.

Your `Stopwatch.js` file should look like this:

```
  import React, { Component } from "react";
  import "./App.css";

  class Stopwatch extends Component {
    render() {
      return (
        <div className="Stopwatch">
          <div className="Stopwatch-header">Stopwatch</div>
        </div>
      );
    }
  }

  export default Stopwatch;
```

and `Countdown.js` like this:

```
  import React, { Component } from "react";
  import "./App.css";
```

```
class Countdown extends Component {
  render() {
    return (
      <div className="Countdown">
        <div className="Countdown-header">Countdown</div>
      </div>
    );
  }
}

export default Countdown;
```

Now that the project is configured for a basic setup, test it out using the following command

```
npm start
```

Stopwatch

The stopwatch timer we are building will fulfill the following requirements:

- the timer will start at 0
- the timer will be able to stop and reset
- the `Stopwatch` component will display the time and control buttons

Component state

Since we need to store the timer data, we will use React's state for this purpose. We will be keeping track of:

`timerOn` : boolean value for if the timer is on
`timerStart` : the Unix Epoch (ms after 1970) time when the timer was started

(or the past projected start time if the timer is resumed)

`timerTime` : total time (ms) that the timer has been running since start/reset

To do this, add the following code to the `Stopwatch` component in the `Stopwatch.js` file just above the `render(){` line

```
state = {
  timerOn: false,
  timerStart: 0,
  timerTime: 0
};
```

Starting the Stopwatch timer

Javascript allows us to set intervals as often as we like which will continuously repeat a given function. If we return a `setState` call, adjusting the time every 10ms, we can keep a very accurate stopwatch. Add the following method to your Stopwatch class component below the state declaration:

```
startTimer = () => {
  this.setState({
    timerOn: true,
    timerTime: this.state.timerTime,
    timerStart: Date.now() - this.state.timerTime
  });
  this.timer = setInterval(() => {
    this.setState({
      timerTime: Date.now() - this.state.timerStart
    });
  }, 10);
};
```

This function, `startTimer` will be called when the timer is started or resumed. At first, it will use the `setState` method to turn the timer on, set the timer to represent the current time, and initialize the start time. Subtracting `this.state.timerTime` from `Date.now()` will set our start time either to when the timer was started, or what that time would have been if the timer is resumed.

Next in the `startTimer` function, we will initialize a timer interval with `this.timer` which sets the timer interval to the `Stopwatch` component. This interval needs to return a method to call every time it goes off, and an interval time. In our return we can call `this.setState` to adjust the current `timerTime` to the number of miliseconds since `timerStart`.

Stop and Reset

Now that the stopwatch start logic is set up, we can add in functions for `stop` and `reset` below `startTimer`

```
stopTimer = () => {
  this.setState({ timerOn: false });
  clearInterval(this.timer);
};

resetTimer = () => {
  this.setState({
    timerStart: 0,
    timerTime: 0
  });
};
```

In the `stopTimer` method, we are setting `timerOn` to false and clearing the interval on `this.timer`.

The `resetTimer` method returns the `timerStart` and `timerTime` back to 0.

Formatting and Display

With all the functionality we need for a timer in order, we need a way to display the current time in `hours`, `minutes`, `seconds`, and `centiseconds`. Add the following code to `Stopwatch.js` inside the `render()` method, above the `return`

```
const { timerTime } = this.state;
let centiseconds = ("0" + (Math.floor(timerTime / 10) %
100)).slice(-2);
let seconds = ("0" + (Math.floor(timerTime / 1000) % 60)).slice(-2);
let minutes = ("0" + (Math.floor(timerTime / 60000) % 60)).slice(-2);
let hours = ("0" + Math.floor(timerTime / 3600000)).slice(-2);
```

We have the value of the time we want to display stored in milliseconds in our state. First, we can destructure the `timerTime` to save so complexity. We are simply setting the variable `this.state.timerTime` to `timerTime`.

The modular arithmetic we are using here is finding the remainder of each unit of time we are using.

- `centiseconds` - 10 represents 1/100th of a second
- `seconds` - 1000 represents 1/60th of a minute
- `minutes` - 60000 represents 1/60th of an hour
- `hours` - 3600000 doesn't need a modulus if <100 hours

We are also formatting the times to display as 2 digits by concatenating a “0” on the front then slicing off the end if its more than 2 digits long.

Underneath our stopwatch header in the `Stopwatch.js` return statement, we can display our computed time variables by add the code:

```
<div className="Stopwatch-display">
  {hours} : {minutes} : {seconds} : {centiseconds}
</div>
```

Controls

Lastly for the Stopwatch, we will need buttons to `start`, `stop`, `resume`, and `reset`. We can conditionally render all 4 buttons depending on the status of the timer under our `stopwatch-display`.

```
{this.state.timerOn === false && this.state.timerTime === 0 && (
  <button onClick={this.startTimer}>Start</button>
)}
{this.state.timerOn === true && (
  <button onClick={this.stopTimer}>Stop</button>
)}
{this.state.timerOn === false && this.state.timerTime > 0 && (
  <button onClick={this.startTimer}>Resume</button>
)}
{this.state.timerOn === false && this.state.timerTime > 0 && (
  <button onClick={this.resetTimer}>Reset</button>
)}
```

Buttons:

`Start` - Show when the timer is off and the time is 0

`Stop` - Show when the timer is on

`Resume` - Show when the time is on, and the time is not 0

`Reset` - Show when the timer is off, and the time is not 0

Now that the Stopwatch is complete, we can start working on the Countdown Timer. The concepts used the Countdown timer will be very similar, try making this one yourself or follow along below.

Countdown

For the countdown timer we will be using the following strategy:

- the timer has buttons to adjust the start time
- the timer will be able to start, stop, and reset
- the timer will display an alert when it runs out
- the `Countdown` component will display the time and control buttons

We will be keeping track of the same state values for the Countdown, as this timer is using the same mechanics as the Stopwatch in reverse.

Add the same state code we used in the Stopwatch to the `Countdown` component in `Countdown.js`

```
state = {
  timerOn: false,
  timerStart: 0,
  timerTime: 0
};
```

Starting the Countdown timer

Our `startTimer` function for the Countdown timer will be very similar to the one we made for the Stopwatch:

```
startTimer = () => {
  this.setState({
    timerOn: true,
    timerTime: this.state.timerTime,
    timerStart: this.state.timerTime
  });
  this.timer = setInterval(() => {
    const newTime = this.state.timerTime - 10;
    if (newTime >= 0) {
      this.setState({
        timerTime: newTime
      });
    } else {
      clearInterval(this.timer);
      this.setState({ timerOn: false });
      alert("Countdown ended");
    }
  }, 10);
};
```

Here we are again initializing the timer to turn on, set the current time, and set the start time to the current time.

Our timer interval for the `Countdown` will first check that the next time will be more than zero. If this is the case, we will return the updated timer as expected. If the new time is less than 0, we need to stop the timer by using the `clearInterval` method, setting the `timerOn` value to false, informing the user with an `alert` message.

Stop and Reset

Our methods to stop and reset the Countdown timer will also be similar, with slight differences:

```
stopTimer = () => {
  clearInterval(this.timer);
  this.setState({ timerOn: false });
};
resetTimer = () => {
  if (this.state.timerOn === false) {
    this.setState({
      timerTime: this.state.timerStart
    });
  }
};
```

Here, our `stopTimer` method will again clear the interval and turn the timer off in component state. Our `resetTimer` function will this time first check to make sure the timer is off, the reset the `timerTime` to our `timerStart` time.

Adjusting the timer

In our Countdown component, we also have to build out the buttons to adjust the `hours`, `minutes`, and `seconds`. I will be creating a single function to handle all 6 buttons and set the state accordingly.

```
adjustTimer = input => {
  const { timerTime, timerOn } = this.state;
  const max = 216000000;
  if (!timerOn) {
    if (input === "incHours" && timerTime + 3600000 < max) {
      this.setState({ timerTime: timerTime + 3600000 });
    } else if (input === "decHours" && timerTime - 3600000 >= 0) {
      this.setState({ timerTime: timerTime - 3600000 });
    } else if (input === "incMinutes" && timerTime + 60000 < max) {
      this.setState({ timerTime: timerTime + 60000 });
    } else if (input === "decMinutes" && timerTime - 60000 >= 0) {
      this.setState({ timerTime: timerTime - 60000 });
    } else if (input === "incSeconds" && timerTime + 1000 < max) {
      this.setState({ timerTime: timerTime + 1000 });
    } else if (input === "decSeconds" && timerTime - 1000 >= 0) {
      this.setState({ timerTime: timerTime - 1000 });
    }
```

```
    }
  };
```

In this method we are using the `timerTime`, and `timerOn` from state enough times to benefit from destructuring these variables at top. We can also set a variable `max` to 216000000 (60 hours) to simplify the code.

Each button case will check if the name input as an argument is appropriate, and also that the `timerTime` will not increase or decrease outside of the timer boundary (0ms to 60 hours). If the conditions are met, we will call `setState` to adjust the `timerTime`

If we set up the buttons now, we should be able to view and adjust our timer in the `React Devtools` in our browser. Here I am adding a label for the Countdown timer, and the increase/decrease buttons for `hours`, `minutes`, and `seconds`:

```
<div className="Countdown-label">Hours : Minutes : Seconds</div>
<div className="Countdown-display">
  <button onClick={() => this.adjustTimer("incHours")}>&#8679;
</button>
  <button onClick={() => this.adjustTimer("incMinutes")}>&#8679;
</button>
  <button onClick={() => this.adjustTimer("incSeconds")}>&#8679;
</button>
  <button onClick={() => this.adjustTimer("decHours")}>&#8681;
</button>
  <button onClick={() => this.adjustTimer("decMinutes")}>&#8681;
</button>
  <button onClick={() => this.adjustTimer("decSeconds")}>&#8681;
</button>
</div>
```

Now, if you open the browser an navigate to the `React` devtools inside the `Countdown` component, you will see the `timerTime` adjusts appropriately so long as the new time is at least `00:00:00` and at most `59:59:59`

Formatting the Countdown

We have the correct time displaying in the components state, but now we need to display it in terms of `hours`, `minutes`, and `seconds` on the screen.

I will use the following code which is similar in concept to our formatting from the `Stopwatch`

```
const { timerTime, timerStart, timerOn } = this.state;
let seconds = ("0" + (Math.floor((timerTime / 1000) % 60) %
60)).slice(-2);
let minutes = ("0" + Math.floor((timerTime / 60000) % 60)).slice(-2);
let hours = ("0" + Math.floor((timerTime / 3600000) % 60)).slice(-2);
```

We will first destructure our state variables, as the will be often used inside the render method. We are also using modular arithmetic, string concatenation, and slice to represent the correct time.

With the time formatted, now we can display our new variables in our `Countdown` component right between the increase buttons and the decrease buttons:

```
<div className="Countdown-time">
  {hours} : {minutes} : {seconds}
</div>
```

The last thing we need to add for the `Countdown` to be functional in the browser is our `start`, `stop`, `resume`, and `reset`, buttons.

The logic for our buttons will be slightly more complicated to ensure we are displaying them correctly.

```
{timerOn === false &&
  (timerStart === 0 || timerTime === timerStart) && (
    <button onClick={this.startTimer}>Start</button>
  )}
{timerOn === true && timerTime >= 1000 && (
  <button onClick={this.stopTimer}>Stop</button>
  )}
{timerOn === false &&
  (timerStart !== 0 && timerStart !== timerTime && timerTime !== 0)
&& (
     <button onClick={this.startTimer}>Resume</button>
  )}
{(timerOn === false || timerTime < 1000) &&
  (timerStart !== timerTime && timerStart > 0) && (
    <button onClick={this.resetTimer}>Reset</button>
  )}
```

With our functionality out of the way, we can now set some styling for our components. I have added the following css styles to `App.css`. Feel free to copy these styles, or make your own.

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
.App {
  text-align: center;
  font-family: Helvetica, sans-serif;
  background-color: rgb(34, 34, 34);
  height: 100vh;
  color: rgb(250, 250, 250);
}
```

```
.App-title {
  font-size: 50px;
  padding: 25px 0;
}

.Timers {
  display: flex;
  justify-content: center;
}
@media (max-width: 900px) {
  .Timers {
    flex-direction: column;
    align-items: center;
  }
}
@media (max-width: 900px) {
  .Stopwatch {
    margin-bottom: 40px;
  }
}
.Countdown,
.Stopwatch {
  margin-left: 30px;
  margin-right: 30px;
  border: 2px solid grey;
  border-radius: 4px;
  padding: 20px;
  width: 400px;
  background-color: rgb(22, 27, 31);
  box-shadow: 0 3px 6px rgb(12, 12, 12);
}
.Countdown {
  padding-top: 10px;
}

.Countdown-header,
.Stopwatch-header {
  font-size: 40px;
  font-weight: bold;
}

button {
  background-color: #202b33;
  border: solid 1px transparent;
  border-radius: 4px;
  padding: 10px 20px;
  color: #ffffff;
  font-size: 16px;
  margin: 0 5px;
  cursor: pointer;
}
```

```
button:hover {
  background-color: #106ba3;
}
.Stopwatch button {
  padding: 12px 32px;
  font-size: 20px;
}
.Stopwatch-display {
  padding: 40px 0;
  font-size: 48px;
}
.Stopwatch-text {
}
.Countdown-display {
  margin-top: 5px;
  margin-bottom: 20px;
}
.Countdown-display button {
  margin: 0 15px;
  border: solid 1px transparent;
  border-radius: 2px;
  padding: 4px 16px;
  color: #ffffff;

  font-size: 16px;
}
.Countdown-display button:hover {
  background-color: #106ba3;
}
.Countdown-label {
  font-size: 18px;
  margin-top: 5px;
  margin-bottom: 10px;
}
.Countdown-time {
  font-size: 36px;
  margin: 5px 0;
}
```

We've successfully built two functional timers with React. I hope this tutorial was helpful in getting started using timers!

React JavaScript Web Development Tutorial Timer

Written by Peter Durham

153 Followers · 18 Following

Follow

Responses (4)

Write a response

What are your thoughts?

Haikal Azmi
Jan 2, 2020

Hi, I have some questions for you
Do you have any contact or social media so that I can talk to you or approach you much easier?

Abdulwahab Alhaji
Aug 27, 2019

Thank you so much for this very helpful and thorough explanation! I customized the approach to count down what's left till a process is done.

Reply

Meera Menon she/her
Jul 5, 2019

It is so well explained that I can understand even without having come up to the level of React...Thanks so much

Reply

See all responses

More from Peter Durham

Peter Durham

Basic commands to interact with the Bitcoin Core RPC console

In this tutorial we will explore using a fully synced Bitcoin Core node to run commands ...

Jun 26, 2019 377 2

Peter Durham

Build Your Own Bitcoin API using Node.js and Bitcoin Core

In this tutorial we will build an API using Node.js and Express that retrieves data from...

Aug 5, 2019 556 7

Peter Durham

Using Local Storage in React to imitate database functionality

React is an excellent modern web framework for displaying data on the front end. This...

Feb 22, 2019 320 1

In The Startup by Peter Durham

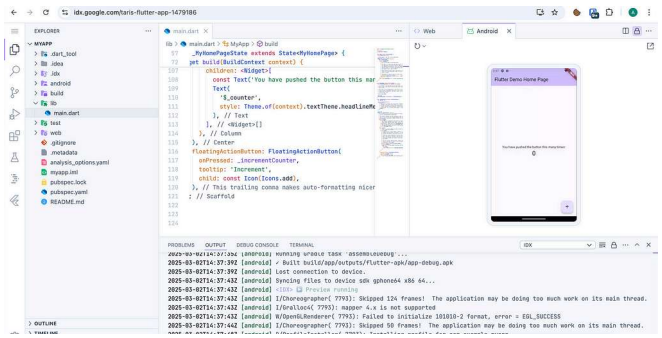
Parcel Basic Setup for Vanilla JS and React

Originally published at https://www.code-boost.com.

May 8, 2020 51

See all from Peter Durham

Recommended from Medium



In Dev Simplified by Neha Gupta

Why Companies Are Saying GoodBye to Next.js?

Are you using Next.js or planning to for your next project? Then you need to know this...

Apr 2 1.4K 79

In Coding Beauty by Tari Ibaba

This new IDE from Google is an absolute game changer

This new IDE from Google is seriously revolutionary.

Mar 12 4.3K 253

Sebastian Carlos

Fired From Meta After 1 Week: Here's All The Dirt I Got

This is not just another story of a disgruntled ex-employee. I'm not shying away from the...

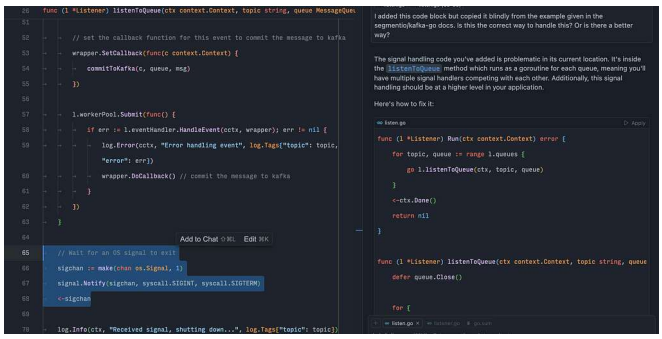
Jan 8 21K 465

Ravi Patel

A Beginner's Guide to the Node.js

Node.js is a powerful platform for building server-side applications, and its built-in fs...

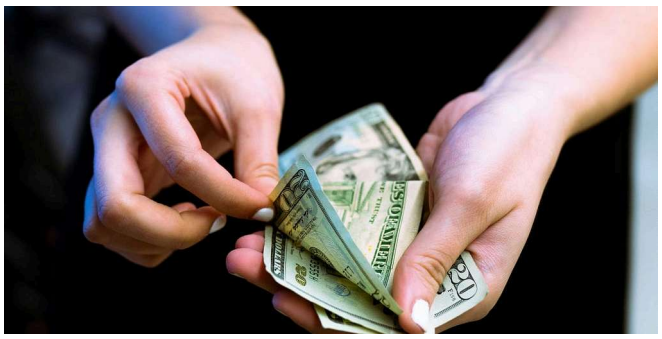
Dec 11, 2024



In Level Up Coding by Jacob Bennett

The 5 paid subscriptions I actually use in 2025 as a Staff Software...

Tools I use that are cheaper than Netflix



In Learn AI for Profit by Nipuna Maduranga

You Can Make Money With AI Without Quitting Your Job

I'm doing it, 2 hours a day

Jan 7 12.4K 317 Mar 25 5.2K 242

See more recommendations