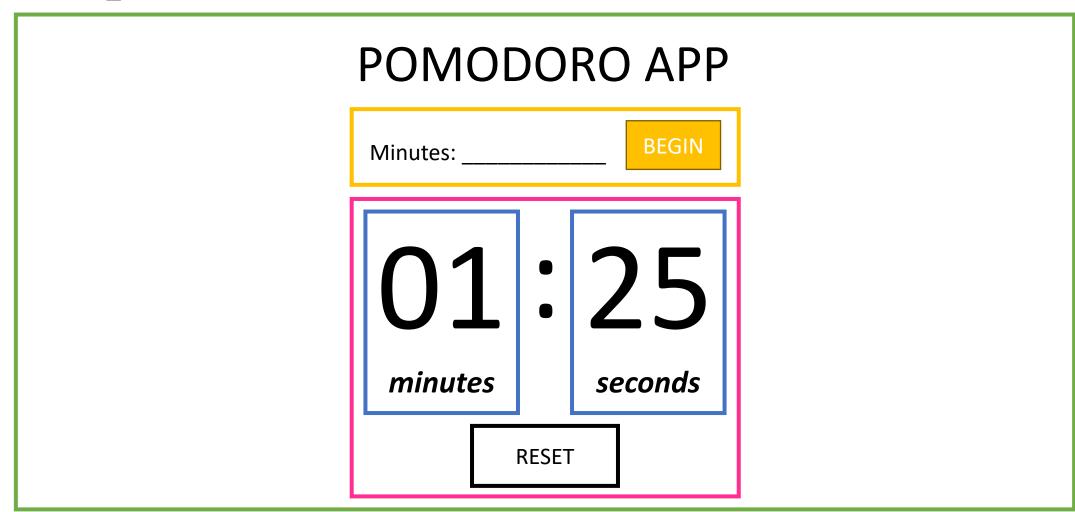
POMODORO APP

BY XTINA PARK

REACT JS, styled-components, react-hook-form, react-confetti, use-sound



Green = App; owns timeLeft and isPaused.

Gold = Form; modifies timeLeft and isPaused.

Pink = Timer; composed of Blocks; uses timeLeft, isPaused and modifies both.

Blue = Block; uses *minutes* or *seconds*, whichever is passed in as a prop from Timer.



timeLeft < 0

POMODORO APP Minutes: ______ BEGIN *CONFETTI ANIMATION USING 'react-confetti'

You have finished your Pomodoro! You can choose to keep working, or to take a short break!

RESET

Green = App; owns timeLeft.

Gold = Form; modifies *timeLeft*

Purple = Finished; displays when *timeLeft <0*, uses *setTimeLeft* to reset *timeLeft* to zero when the reset button is used.





useRef & useEffect

Friends forever, like Patrick and SpongeBob.



Example of useRef, useEffect, and setInterval

Below is an example of keeping a mutable variable in a ref. The component <Timer> initializes a setInterval on every re-render and needs to implement a callback to stop its interval imperatively:

```
import React, { useRef, useEffect } from "react";

const Timer = () => {
   const intervalRef = useRef();

   useEffect(() => {
      const id = setInterval(() => {
       console.log("A second has passed");
    }, 1000);
```

```
// We need the interval id to be accessible from the whole component.
11
12
         // If we stored the id in a state variable, the component would be re-rendered
         // after the state update so a new interval will be created (this effect is triggere
13
         // after every re-render) leading us to the infinite loop hell.
14
         intervalRef.current = id;
15
16
17
         return () => clearInterval(intervalRef.current);
18
      });
19
20
       const handleCancel = () => clearInterval(intervalRef.current);
21
22
       return (
                      When does useEffect run?
23
                      By default, it runs after every render but we can customize it with the
24
           //...
                      second param of the useEffect function. As a second argument, the
25
         </>
                      useEffect function accepts an array that allow us to tell React when we
26
                      want our effect to be called.
27
timer.js hosted with \vec{\psi} b
                      After a render and before calling an effect. React will compare the array of
                      values defined in the second r
                                                              he effect with the array defined
                      in the same effect from the previous render. React will only call the effect
                      when any value of the array has changed since the previous render.
```

Source: https://medium.com/trabe/react-useref-hook-b6c9d39e2022

Author: Ceci García García



UseRef notes cont'd

Summing up

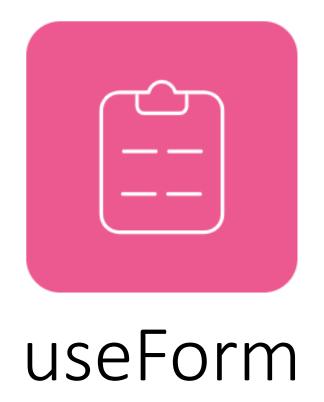
The useRef Hook lets us create mutable variables inside functional components. There are three main key points that you should keep in mind when using the useRef Hook:

- A ref created with useRef will be created only when the component has been mounted and preserved for the full lifecycle.
- Refs can be used for accessing DOM nodes or React elements, and for storing mutable variables (like with instance variables in class components).
- Updating a ref is a side effect so it should be done only inside a useEffect (or useLayoutEffect) or inside an event handler.

Source: https://medium.com/trabe/react-useref-hook-b6c9d39e2022

Author: Ceci García García





AKA React Hook Form, the easiest way to handle user input!



UseForm (1/2)

```
import { useForm } from "react-hook-form";
function Form({submitForm}) {
      const { register, handleSubmit, errors } = useForm();
      const onSubmit = data => {
            console.log(data);
            submitForm();
```



UseForm (2/2)

```
return (<div align='center'>
<form onSubmit={handleSubmit(onSubmit)} noValidate id="pomodoro-form"> ...
<input
          type="number" name="minutes" id="minutes-input"
          ref={register({
                  required: {value: true, message: "Please enter the number of minutes ..."},
                  min: {value: .01, message: "Please enter a number between .01 and 100."},
                  max: {value: 100, message: "Please enter number between .01 and 100."},
          })}
        />
<input type="submit"/>
        {errors.minutes && {errors.minutes.message}}
      </form>
    </div>)
```





Random Snippets I Know I Will Forget

"Pride is not the opposite of shame, but its source. True humility is the only antidote to shame." — Uncle Iroh



IMPORTS

```
import { useForm } from "react-hook-form";
import * as $ from 'jquery';
import Confetti from 'react-confetti';
import useSound from 'use-sound';
```



STYLING:

```
display:flex;
justify-content: center;
```

margin/padding: top-bottom left-right; ^Four inputs = clockwise from top

text-shadow: 2px 2px grey;



OTHER:

Math.floor()

```
$('#pomodoro-form').trigger("reset");

Remember to pass data to onSubmit:

console.log('minutes:' + data.minutes);
```

let width=window.innerWidth;
<Confetti width={width}/>
const [play] = useSound(ObjectName, { volume: .25 });

