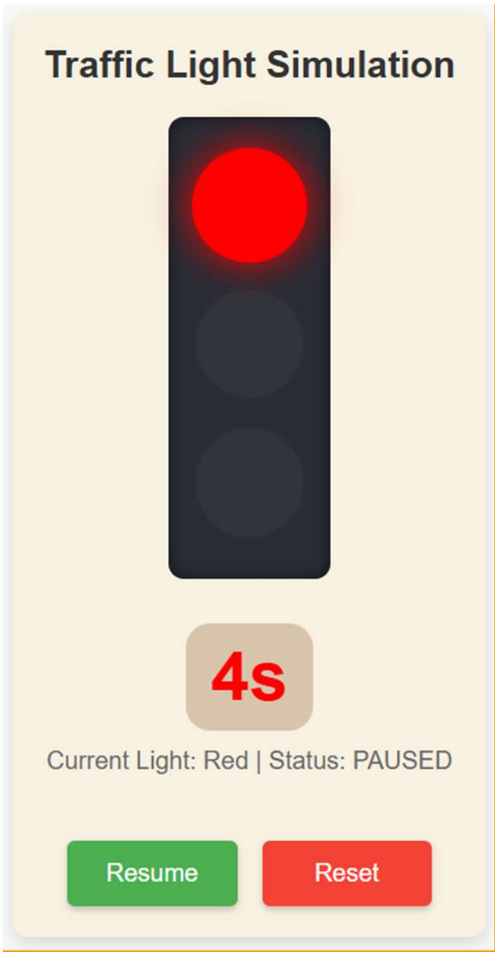
Pre-Assessment Test Pahamify Internship 2025

Documentation for Frontend Path

By: Irfan Sidiq Permana



Last Committed:
26 June 2025, 18:53:01

Problem Author: @arkanmis

Problem Link: https://github.com/arkanmis/traffic-light-simulator

Problem Description

In Pre-Assesment Frontend Path for Pahamify Internship 2025, we are tasked to build a **React-based traffic light simulation** that cycles through Red, Green, and Yellow lights in a fixed order with specified durations for each color:

- Red: 5 seconds
- Green: 4 seconds
- Yellow: 2 seconds

The system must support features such as:

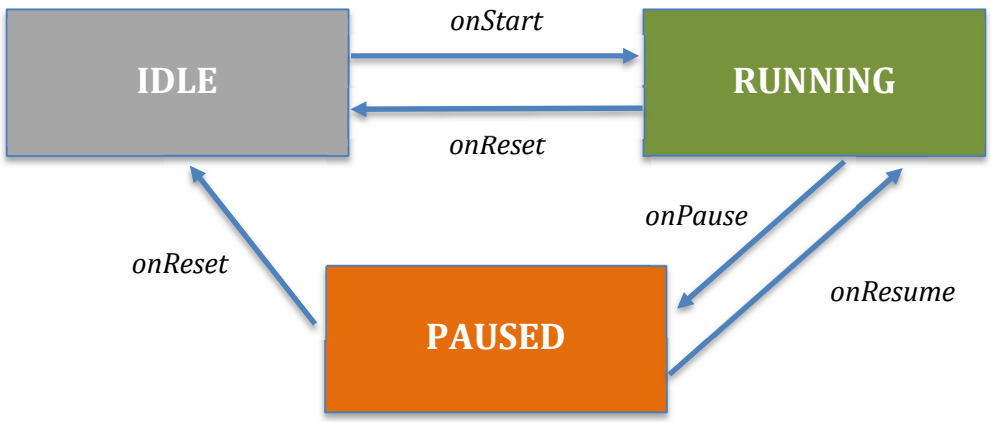
- **Pausing** (freezing the state and timer)
- **Resetting** (returning to Red and stopping)

The program should ideally use a **pure finite state machine (FSM)** model rather than simple timeout chaining. Additional features include handling edge cases like rapid pause/start and simulating lag, while ensuring clean separation between state logic and UI, proper timer management, and scalable, clear code design.

Solution Design

First, we define the possible FSM states for our traffic light and their implications:

- **Idle**: Program first loaded or user presses reset button → Color changes to Red, timer resets and stops running
- **Running**: User presses the start or resume button → Timer starts running, color changes over time
- **Paused**: User presses the pause button → Timer stops running, color stays the same



Next, we define what we'll store on our traffic light state, that is what information we'll be keeping track of. Our traffic light program will constantly keep track of its current state which contains:

- **currentLight**: The light that we're currently on (can be Red, Yellow, or Green)

- **timeLeft**: The amount of time we have left before changing to the next color (in seconds)
- **fsmStatus**: Current FSM status (can be Idle, Running, or Stopped)

Lastly, to set the timer for updating secondsLeft we use setInterval with the interval of 1 second:

```
On fsmStatus changed:
  if interval is running then
    Stop interval using clearInterval()
  if fsmStatus = 'running' then
    Set interval to execute handleTick() every 1 second
```

where handleTick() updates the seconds left:

```
function handleTick():
  if timeLeft > 1 then
    timeLeft ← timeLeft – 1
  else
    currentLight ← next color
    timeLeft ← duration of next color
```

Testing

Aside from testing by using the program normally, we can also test edge cases below:

a. Rapid Pause/Resume Toggling

We can test this using requestAnimationFrame to toggle pause/resume every frame.

A dummy component has been prepared in `src/features/tests/TrafficLightTest.tsx` for this. To use it, just change App.tsx to use TrafficLightTest component instead of TrafficTest:

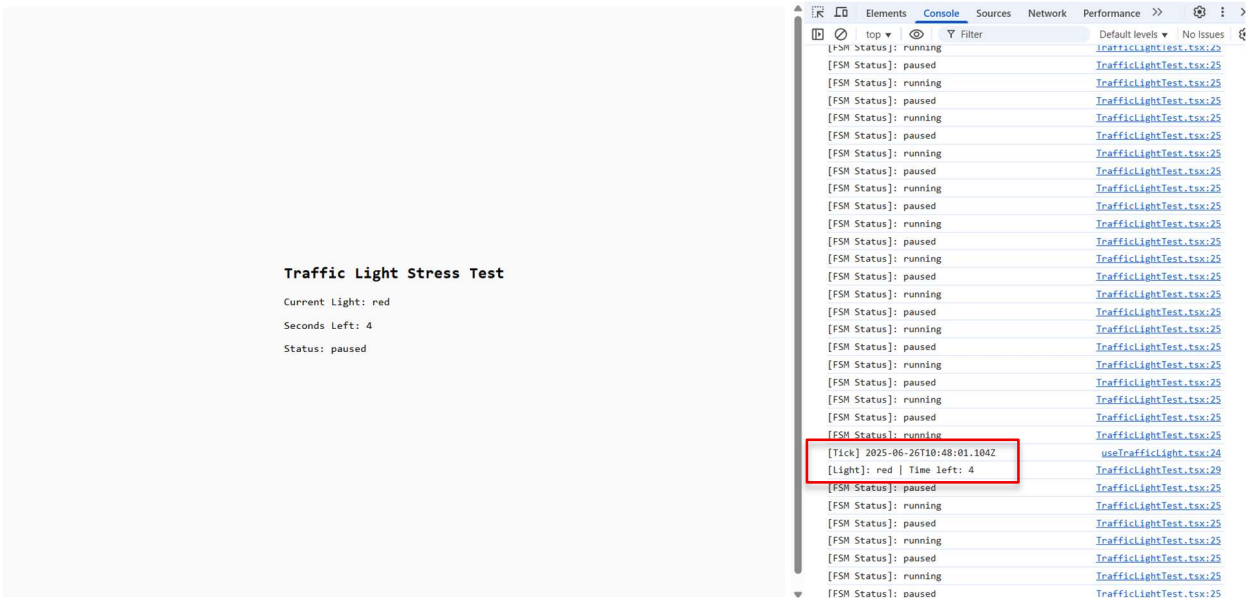
```tsx
import { TrafficLight } from "@/components/TrafficLight";
import { TrafficLightTest } from "./features/trafficLight";
import '@/App.css';


export default function App() {
  return (
    <div className="app-container">
      <TrafficLightTest />
    </div>
  );
}
```

Upon testing, we'll see that:

- Status is changing very rapidly from **Paused** to **Running**
- Seconds Left is decreasing very, **very** slowly (may appear not changing at all) due to the timer constantly being paused every frame

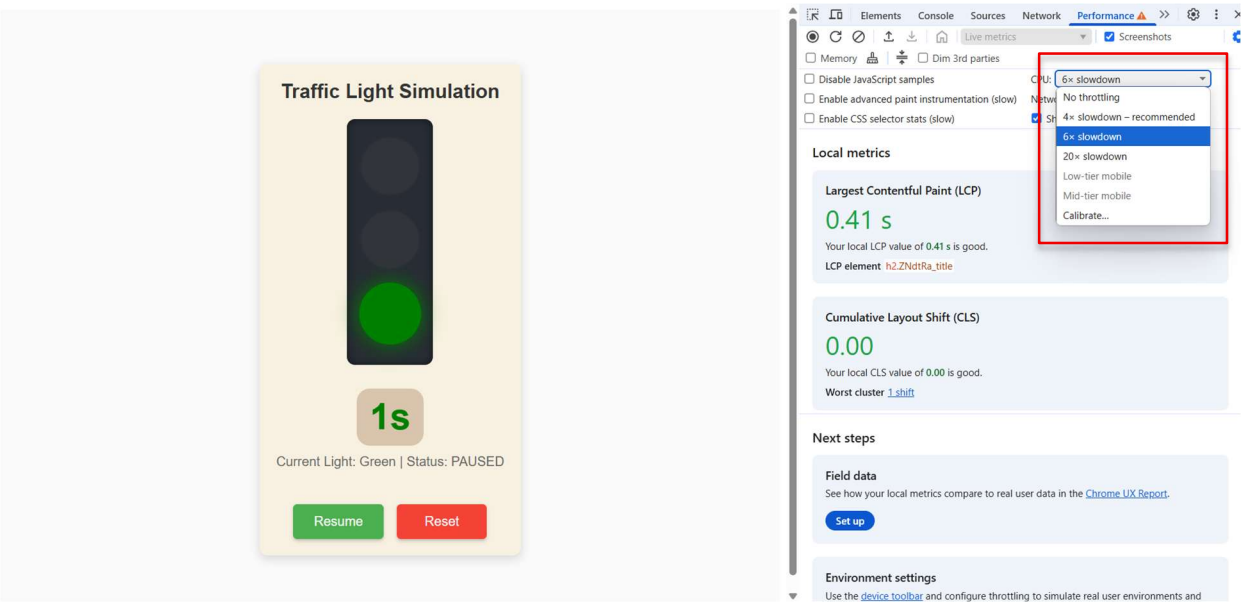
If you open the browser console, you'll see the logs that show the state is changing very quickly between paused and running. However, if you wait long enough **handleTick() will eventually be called**, decreasing the seconds left:



The color will also eventually change when seconds left reaches 0.

b. Skipping Frames

To skip a few frames, we can simulate lag using CPU throttling feature available on Chrome browser:



Upon applying **4x slowdown**, you should see that the timer still **decreases by one second each time** and **changing lights still happen normally**, albeit not very smoothly due to performance lag.