Scaler Companion beta

Here's a comprehensive set of revision notes based on the content from the class on building a stopwatch application and creating custom hooks in React:

```
# React Class Revision Notes

## Introduction to React StopWatch Application

### Problem Statement
The goal is to create a stopwatch application using React wi
- **Start**: Begin the time tracking.
- **Stop**: Halt the time tracking.
- **Reset**: Reset the stopwatch back to 00:00:00.
- **Display**: Show the elapsed time in hours:minutes:second

### Key Features
- **State Management**: Using `useState` to handle time and
- **Timers**: Utilizing `setInterval` and `clearInterval` fo
- **Refs**: Employ `useRef` to persist timer ID without caus
- **Formatting**: A format function to convert raw time into

### Implementation Steps
1. **Set Up the React Project**: Initialize using Create Rea
2. **Build Stopwatch Component**: Incorporate `useState`, `u
3. **Refactoring and Optimization**:
   - Use `React.memo` to prevent unnecessary renders.
   - Use `useCallback` to memoize functions【8:13†typed.md】

### Example Code Structure
```jsx
import React, { useState, useRef, useCallback } from 'react'

const Stopwatch = React.memo(() => {
  const [time, setTime] = useState(0);
  const [isRunning, setIsRunning] = useState(false);
  const timerRef = useRef(null);
```
```

```
      setIsRunning(true);
      timerRef.current = setInterval(() => {
        setTime(prevTime => prevTime + 1);
      }, 1000);
    }
  }, [isRunning]);

  const stopTimer = useCallback(() => {
    if (isRunning) {
      clearInterval(timerRef.current);
      setIsRunning(false);
    }
  }, [isRunning]);

  const resetTimer = useCallback(() => {
    clearInterval(timerRef.current);
    setIsRunning(false);
    setTime(0);
  }, []);

  const formatTime = (time) => {
    const getSeconds = `0${time % 60}`.slice(-2);
    const minutes = Math.floor(time / 60);
    const getMinutes = `0${minutes % 60}`.slice(-2);
    const getHours = `0${Math.floor(time / 3600)}`.slice(-2)
    return `${getHours}:${getMinutes}:${getSeconds}`;
  };

  return (
    <div>
      <h1>{formatTime(time)}</h1>
      <button onClick={startTimer}>Start</button>
      <button onClick={stopTimer}>Stop</button>
      <button onClick={resetTimer}>Reset</button>
    </div>
  );
});

export default Stopwatch;
```

Understanding Custom Hooks: `useVisibility`

Problem Statement

Create a reusable custom hook to manage the visibility of UI elements like modals or dropdowns, encapsulating functionality to show, hide, and toggle visibility【8:15†typed.md】.

Features

- **Initial State**: Set whether the item is initially visible or hidden.
- **Visibility Control**: Provide methods to toggle visibility.
- **Reusable Across Components**: Implement the hook in multiple UI elements without duplicating logic【8:15†typed.md】.

Implementation Steps

1. **Setup Custom Hook**: Define the initial visibility state using `useState` and manage visibility with `useCallback`.
2. **Integrate into Components**: Use the custom hook within a React component to control UI element visibility【8:15†typed.md】.

Example Code Structure

```
import { useState, useCallback } from 'react';

function useVisibility(initialVisibility = false) {
  const [isVisible, setIsVisible] = useState(initialVisibili

  const show = useCallback(() => {
    setIsVisible(true);
  }, []);

  const hide = useCallback(() => {
```

```
  const toggle = useCallback(() => {
    setIsVisible(prev => !prev);
  }, []);

  return {
    isVisible,
    show,
    hide,
    toggle,
  };
}

export default useVisibility;
```

【8:18†transcript.txt】

Using the Custom Hook

- Create a modal component leveraging the `useVisibility` hook to manage UI state, ensuring clean and maintainable code 【8:15†typed.md】.

This class covered important practical implementations in React, focusing on building a dynamic stopwatch and developing custom hooks to enhance component reusability and optimize state handling.