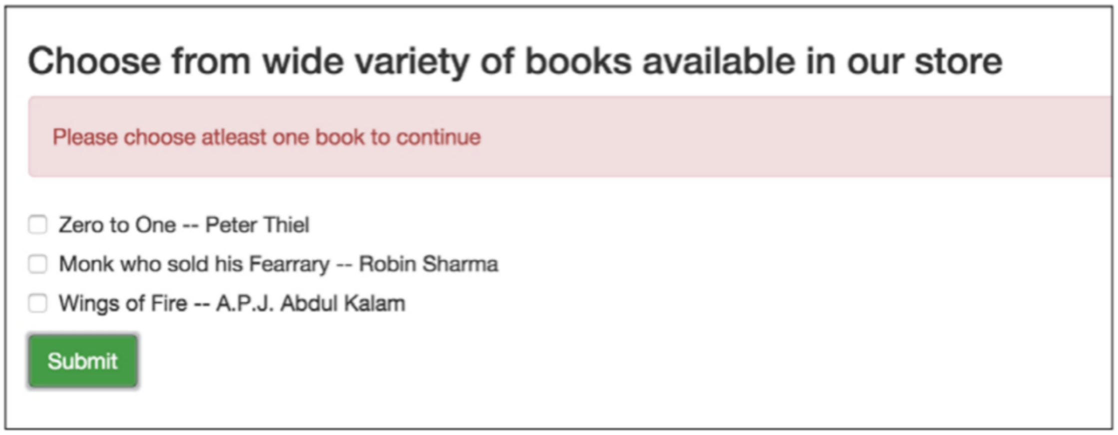
	Course Name: Advanced Web Technology	**AWT_Assignment-2**	
	Course Code: 23CP308T AY: 2023-24 **Faculty: Komal Singh**	**Branch: CSE**	**Semester: VI**
Submitted by: Patel Vraj ChetanKumar **Roll no: 21BCP362**			

Q1.

1. Implement basic validation within the BookList component to prevent progression to the next step unless a book has been selected by the user [**CO5**].

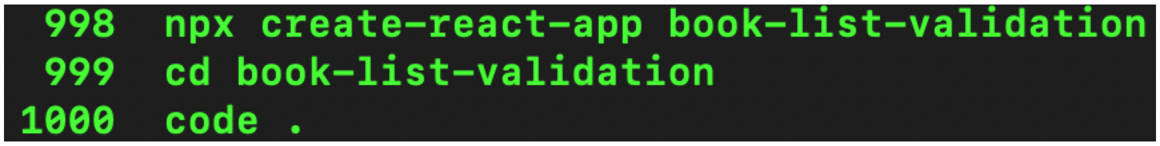


Choose from wide variety of books available in our store

Please choose atleast one book to continue

- [ ] Zero to One -- Peter Thiel
- [ ] Monk who sold his Fearrary -- Robin Sharma
- [ ] Wings of Fire -- A.P.J. Abdul Kalam

1. Create a react app and navigate to it



```
npx create-react-app book-list-validation
cd book-list-validation
code .
```

2. Create booklist.js file and its respective css file

```
import React, { useState } from 'react';
import './BookList.css'; // Importing the CSS for styling

const booksData = [
  { id: 'book1', title: 'Zero to One -- Peter Thiel' },
  { id: 'book2', title: 'Monk Who Sold His Ferrari -- Robin Sharma' },
  { id: 'book3', title: 'Wings of Fire -- A.P.J. Abdul Kalam' }
];

const BookList = () => {
  const [selectedBooks, setSelectedBooks] = useState([]);
  const [isError, setIsError] = useState(false);
  const [submitted, setSubmitted] = useState(false);

  const handleBookSelect = (bookId) => {
    const newSelectedBooks = selectedBooks.includes(bookId)
      ? selectedBooks.filter(id => id !== bookId)
      : [...selectedBooks, bookId];

    setSelectedBooks(newSelectedBooks);
    // Whenever the user selects a book, we assume they're still interacting with the form,
    // so we should hide the submission message if it's visible.
    if (submitted) {
      setSubmitted(false);
    }
    setIsError(false); // Reset error if the user has selected a book
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    if (selectedBooks.length === 0) {
      setIsError(true);
    } else {
      console.log('Selected Books:', selectedBooks);
      setSelectedBooks([]); // Reset the selected books
      setIsError(false); // Reset any error messages
      setSubmitted(true); // Show the thank you message
    }
  };

  return (
    <form onSubmit={handleSubmit} className="book-list-form">
      <h3>Choose from a wide variety of books available in our store</h3>

      {isError && <div className="error-message">Please choose at least one book to continue</div>}
      {submitted && <div className="success-message">Thanks for your submission!</div>}

      {booksData.map(book => (
        <label key={book.id} className="book-option">
          <input
```

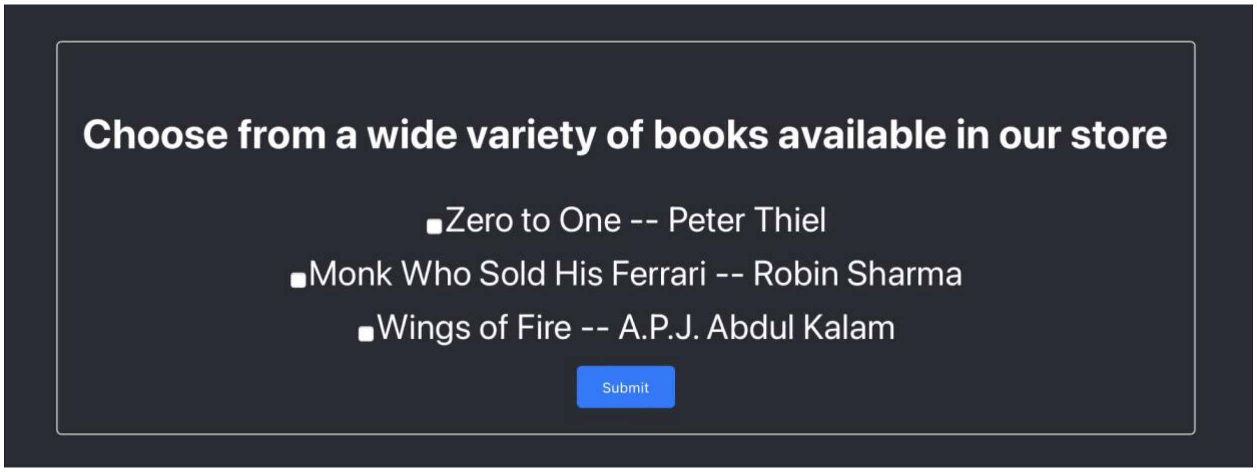
```
            type="checkbox"
            value={book.id}
            checked={selectedBooks.includes(book.id)}
            onChange={() => handleBookSelect(book.id)}
          />
          {book.title}
        </label>
      ))}

      <button type="submit" className="submit-btn">Submit</button>
    </form>
  );
};

export default BookList;
```

The **BookList** component, implemented in React, provides an interactive list of books from which users can make multiple selections. This component demonstrates a fundamental use case of React's stateful logic coupled with user interaction.
Upon importing necessary dependencies and styles, **booksData** is defined as an array of objects, where each object represents a book with a unique **id** and a **title**.
Within the **BookList** function component, three pieces of state are established using the **useState** hook:

- **selectedBooks** is an array that holds the IDs of the books selected by the user. Initialized as an empty array, it represents no initial selection.
- **isError** is a boolean flag that indicates whether an error message should be displayed, particularly when no book is selected upon form submission.
- **submitted** is another boolean flag that represents whether the form has been successfully submitted.

The **handleBookSelect** function is responsible for updating the **selectedBooks** state when a user interacts with the checkboxes corresponding to the book titles. If a book is already selected, it is removed from the array; otherwise, it is added. Additionally, it resets the **submitted** state to false if it was previously set to true, signifying that the user is still interacting with the form. It also resets the **isError** state to ensure that no error message is shown once a selection is made.
The **handleSubmit** function is called when the form is submitted. It prevents the default form submission action to control the process programmatically. If no books are

selected (**selectedBooks.length** is 0), **isError** is set to true, triggering the display of an error message. Otherwise, the current selection is logged to the console, **selectedBooks** is reset to an empty array, **isError** is reset to false, and **submitted** is set to true, leading to the display of a thank-you message.
The component's return statement renders a form containing a heading, dynamically generated checkboxes for each book, an error message if **isError** is true, and a success message if **submitted** is true. The checkboxes are controlled components, with their checked state determined by the presence of their corresponding book ID in the **selectedBooks** array. The **onChange** event for each checkbox is linked to the **handleBookSelect** function, which will update the component's state accordingly.
Finally, the **BookList** component is exported as a default export, making it available for use in other parts of the application.
This component illustrates core React concepts such as component structure, state management, conditional rendering, list rendering, event handling, and form submission.



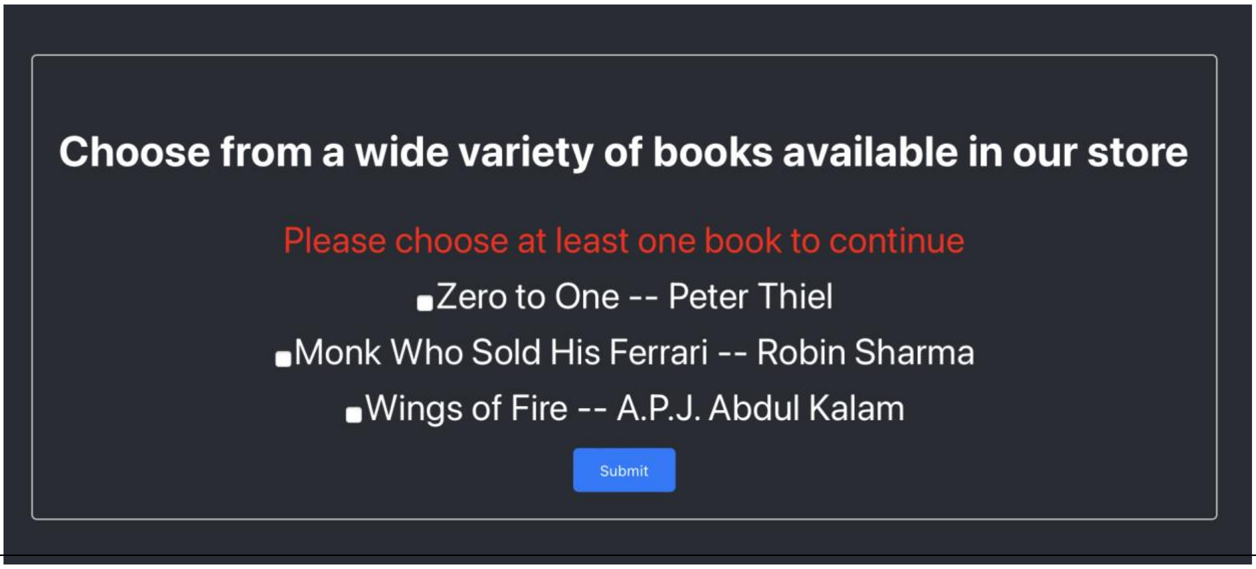
Choose from a wide variety of books available in our store

Zero to One -- Peter Thiel

Monk Who Sold His Ferrari -- Robin Sharma

Wings of Fire -- A.P.J. Abdul Kalam

Submitting without making a selection !



Choose from a wide variety of books available in our store

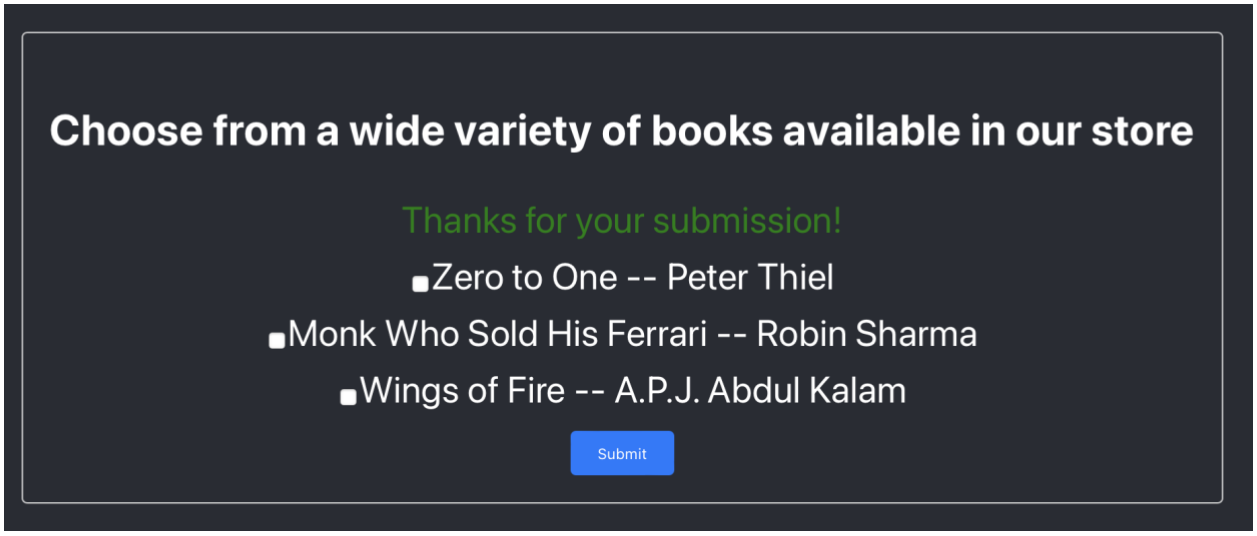
Please choose at least one book to continue

Zero to One -- Peter Thiel

Monk Who Sold His Ferrari -- Robin Sharma

Wings of Fire -- A.P.J. Abdul Kalam

Submitting the form after making the selection !

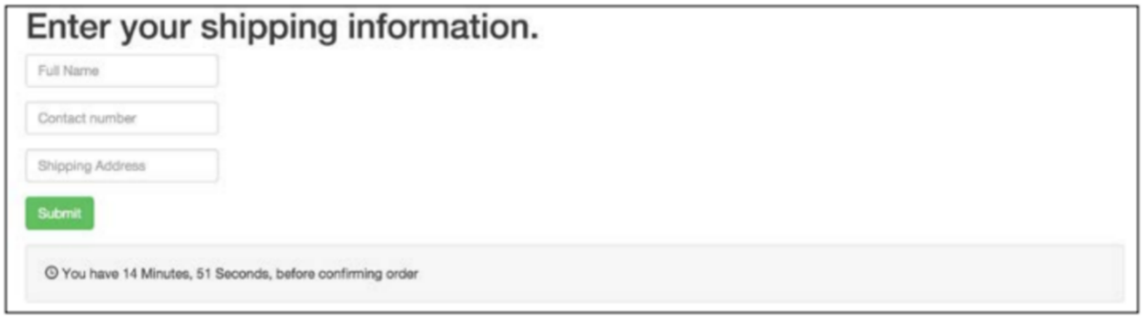

Choose from a wide variety of books available in our store

Thanks for your submission!

- Zero to One -- Peter Thiel
- Monk Who Sold His Ferrari -- Robin Sharma
- Wings of Fire -- A.P.J. Abdul Kalam

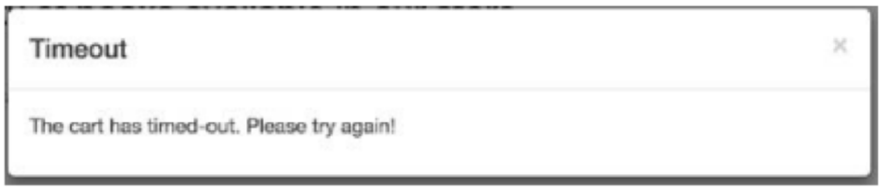
Q2

Make the form like below in React with decrement timer [**CO3,5**].


Enter your shipping information.

You have 14 Minutes, 51 Seconds, before confirming order

And print suitable message like below in same form if counter is timed-out. Print the data after clicking on submit button if data is passed in above form within timed-out period.


Timeout

The cart has timed-out. Please try again!

FormWithTimer.js

```
import React, { useState, useEffect } from 'react';

import './FormWithTimer.css';


const FormWithTimer = () => {

  // State for storing form input values

  const [formData, setFormData] = useState({

    fullName: '',

    contactNumber: '',

    shippingAddress: '',

  });


  // State for countdown timer starting at 15 minutes

  const [timeLeft, setTimeLeft] = useState(15 * 60);


  // State to track whether the form has been submitted

  const [submitted, setSubmitted] = useState(false);


  // State to track whether the form submission has timed out

  const [timeout, setTimeout] = useState(false);


  // Effect hook to handle the countdown timer

  useEffect(() => {

    // Set up an interval that updates the 'timeLeft' state every second

    const timer = timeLeft > 0 && setInterval(() => setTimeLeft(timeLeft - 1), 1000);


    // Cleanup function to clear the interval when the component unmounts or 'timeLeft' updates

    return () => clearInterval(timer);
  }, [timeLeft]);


  // Function to handle form input changes
  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData({ ...formData, [name]: value });
  };


  // Function to handle form submission
  const handleSubmit = (e) => {
    e.preventDefault();
    // Check if the timer has not run out
    if (timeLeft > 0) {
      setSubmitted(true);   // Indicate that the form has been submitted
      setTimeout(false);    // Reset the timeout state
      console.log('Form Data:', formData);  // Log form data to the console
      // Show the form data to the user (for demonstration purposes)
      alert(`Form Data: Full Name - ${formData.fullName}, Contact Number -
${formData.contactNumber}, Shipping Address - ${formData.shippingAddress}`);
      // Additional form submission handling would go here
    } else {
      setTimeout(true);     // Indicate that the submission has timed out
      setSubmitted(false);  // Reset the submitted state
    }
  };


  // Render a thank you message if the form has been submitted
  if (submitted) {
    return <div>Thank you for your submission!</div>;
```

```
}


// Render a timeout message if the submission has timed out
if (timeout) {
  return <div>Timeout. The cart has timed-out. Please try again!</div>;
}


// Render the form with inputs and the submit button
return (
  <div>
    <form onSubmit={handleSubmit}>
      <label>
        Full Name
        <input
          type="text"
          name="fullName"
          value={formData.fullName}
          onChange={handleChange}
        />
      </label>
      <label>
        Contact Number
        <input
          type="text"
          name="contactNumber"
          value={formData.contactNumber}
          onChange={handleChange}
        />
        </label>
        <label>
          Shipping Address
          <input
            type="text"
            name="shippingAddress"
            value={formData.shippingAddress}
            onChange={handleChange}
          />
        </label>
        <button type="submit">Submit</button>
      </form>
      <p>You have {Math.floor(timeLeft / 60)} Minutes, {timeLeft % 60} Seconds, before confirming order</p>
    </div>
  );
};

export default FormWithTimer;
```

The **FormWithTimer** component is designed to collect shipping information within a specified time limit. This time-sensitive form adds a layer of urgency for the user, prompting them to submit their details within 15 minutes.

Upon initialization, the component sets up its state using the **useState** hook for several aspects:

- **formData** holds the values entered into the form fields: full name, contact number, and shipping address. It is initialized with empty strings.
- **timeLeft** starts with 900 seconds, equivalent to 15 minutes, and serves as the countdown timer for the form submission window.
- **submitted** is a boolean flag indicating whether the form has been successfully submitted.
- **timeout** indicates if the time limit has been reached without submission.

The **useEffect** hook manages the countdown timer. It creates an interval that decrements the **timeLeft** state by one second every 1000 milliseconds. If the component unmounts or the **timeLeft** changes, the interval is cleared to prevent memory leaks and unintended behavior.

The **handleChange** function updates the **formData** state when the user types into the form inputs. It captures the input's name and value, ensuring the correct field is updated within the state.

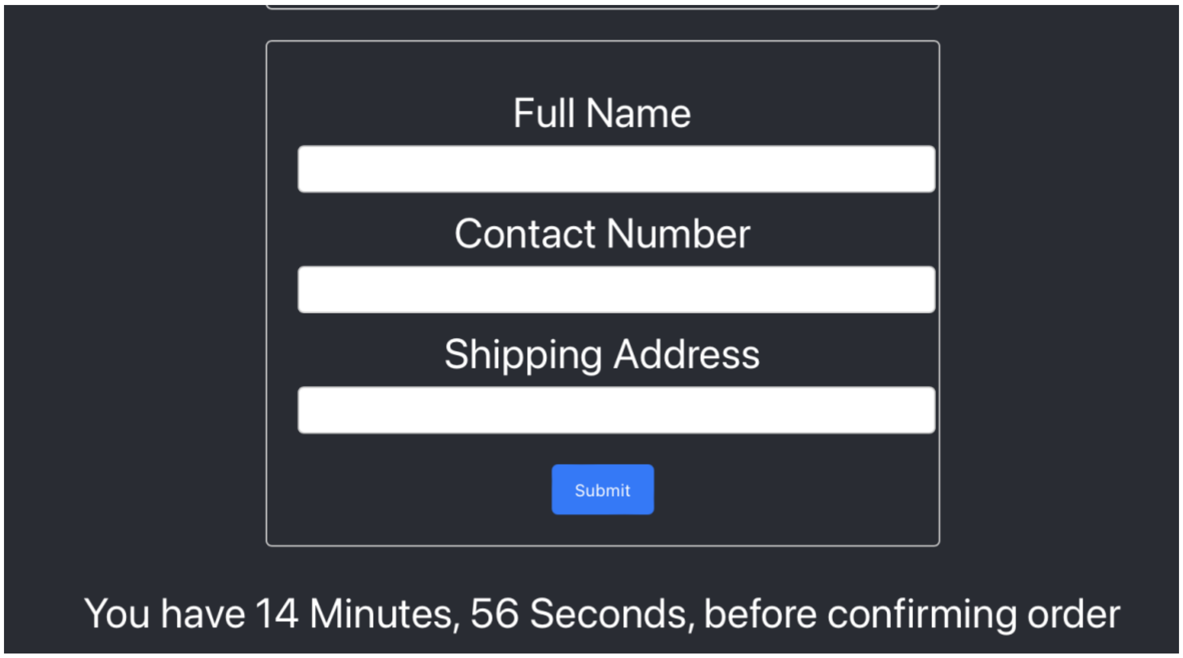
The **handleSubmit** function triggers when the form is submitted. It performs the following actions:

- Prevents the default form submission behavior, keeping the page from refreshing.
- Checks if there is time remaining (**timeLeft > 0**).
- If time is left, it sets **submitted** to true, logs the **formData** to the console, and displays the form data in an alert box to the user.
- If time has run out, it sets **timeout** to true and **submitted** to false, indicating the form was not submitted in time.

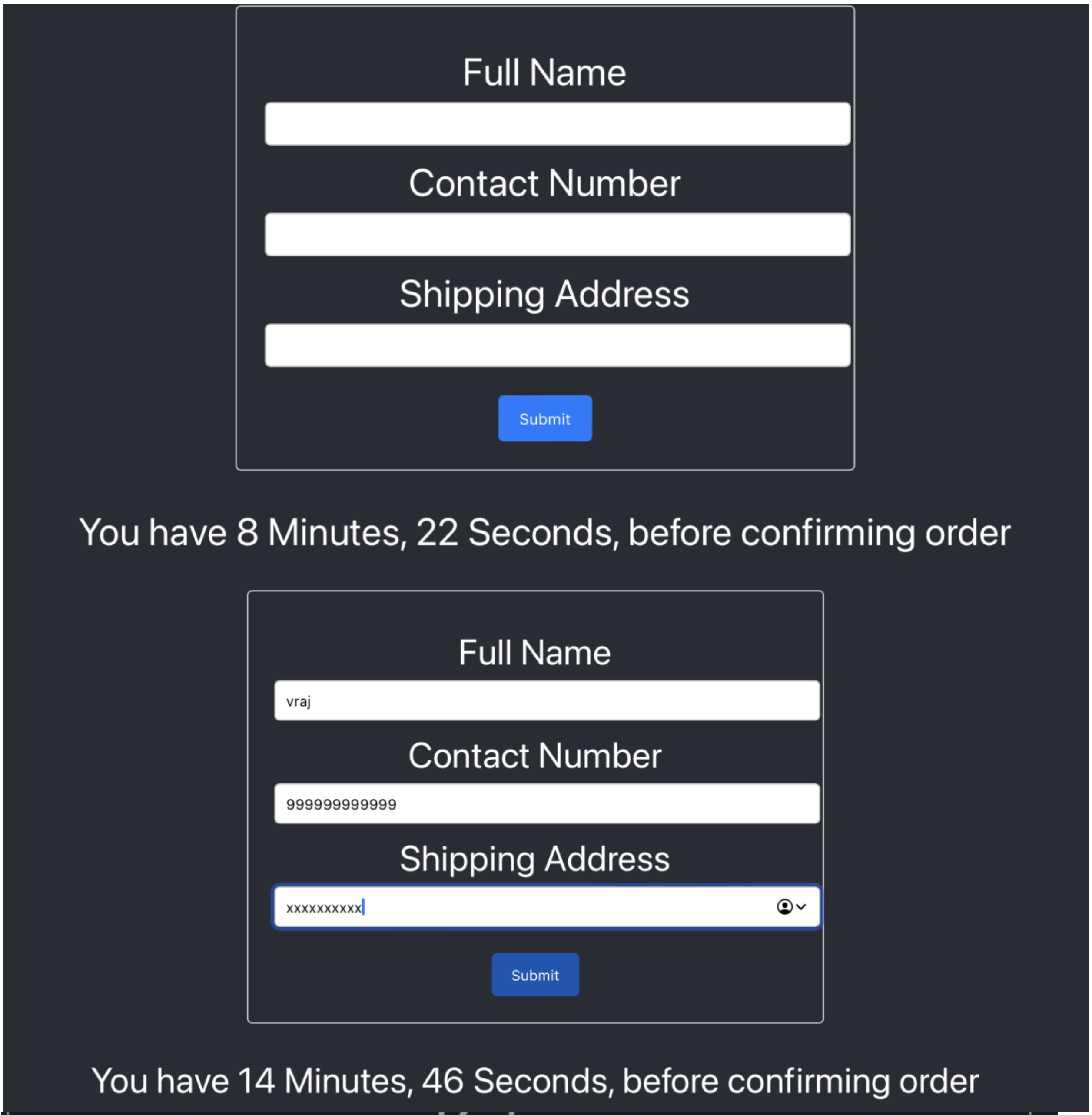
After submission, if **submitted** is true, a thank message is displayed, indicating successful submission. If **timeout** is true, an error message is shown, prompting the user to try again.

The form's UI consists of labels and inputs for full name, contact number, and shipping address, with a submit button at the end. It also displays the remaining time for submission, updating every second.

Finally, the component is exported as **FormWithTimer**, making it available for import and use in other parts of the application.



You have 14 Minutes, 56 Seconds, before confirming order



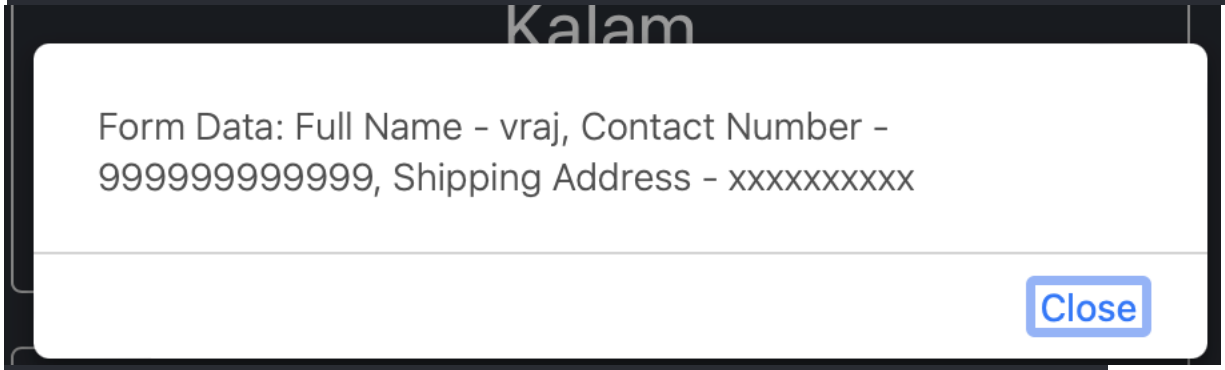
You have 8 Minutes, 22 Seconds, before confirming order

vraj

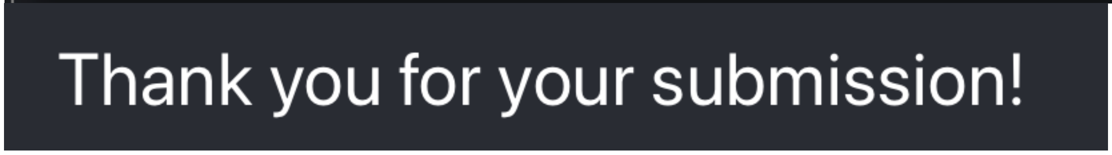
999999999999

xxxxxxxxxx

You have 14 Minutes, 46 Seconds, before confirming order



Form Data: Full Name - vraj, Contact Number - 999999999999, Shipping Address - xxxxxxxxxx


Thank you for your submission!