CheckOutForm.jsx

```jsx
import React, { useState } from 'react';
import { useForm } from 'react-hook-form';

const CheckoutForm = () => {
  const { register, handleSubmit, formState: { errors } } = useForm();
  const [billingSameAsShipping, setBillingSameAsShipping] = useState(false);
```

```jsx
  const onSubmit = (data) => {
    alert('Form submitted successfully!');
    console.log(data);
  };
```

```jsx
  const validateBillingAddress = (value) => {
    // Add custom validation logic for billing address
    const addressPattern = /^[a-zA-Z0-9\s,.'-]{3,}$/; // simple address pattern
    return addressPattern.test(value) || 'Invalid billing address format!';
  };
```

```jsx
  const validateCreditCard = (value) => {
    // Simple credit card validation (Luhn algorithm would be better in production)
    const cardPattern = /^[0-9]{16}$/;
    return cardPattern.test(value) || 'Invalid credit card number!';
  };
```

```jsx
  const validateCVV = (value) => {
    const cvvPattern = /^[0-9]{3,4}$/;
    return cvvPattern.test(value) || 'Invalid CVV!';
  };
```

```jsx
  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <h2>Billing Address</h2>
      <div>
        <label htmlFor="billingName">Name</label>
        <input
          id="billingName"
          {...register('billingName', { required: 'Billing name is required!' })}
        />
        {errors.billingName && <p>{errors.billingName.message}</p>}
      </div>
```

```jsx
      <div>
        <label htmlFor="billingAddress">Address</label>
        <input
          id="billingAddress"
          {...register('billingAddress', {
            required: 'Billing address is required!',
            validate: validateBillingAddress,
          })}
        />
        {errors.billingAddress && <p>{errors.billingAddress.message}</p>}
      </div>
```

```jsx
      <div>
        <input
          type="checkbox"
          id="sameAsShipping"
          onChange={(e) => setBillingSameAsShipping(e.target.checked)}
        />
        <label htmlFor="sameAsShipping">Billing same as shipping</label>
      </div>
```

```jsx
      {!billingSameAsShipping && (
        <>
          <h2>Shipping Address</h2>
          <div>
            <label htmlFor="shippingName">Name</label>
            <input
              id="shippingName"
              {...register('shippingName', { required: 'Shipping name is required!' })}
            />
            {errors.shippingName && <p>{errors.shippingName.message}</p>}
          </div>
```

```
          <div>
            <label htmlFor="shippingAddress">Address</label>
            <input
              id="shippingAddress"
              {...register('shippingAddress', { required: 'Shipping address is required!' })}
            />
            {errors.shippingAddress && <p>{errors.shippingAddress.message}</p>}
          </div>
        </>
      )}
```

```
      <h2>Payment Information</h2>
      <div>
        <label htmlFor="creditCardNumber">Credit Card Number</label>
        <input
          id="creditCardNumber"
          {...register('creditCardNumber', {
            required: 'Credit card number is required!',
            validate: validateCreditCard,
          })}
        />
        {errors.creditCardNumber && <p>{errors.creditCardNumber.message}</p>}
      </div>
```

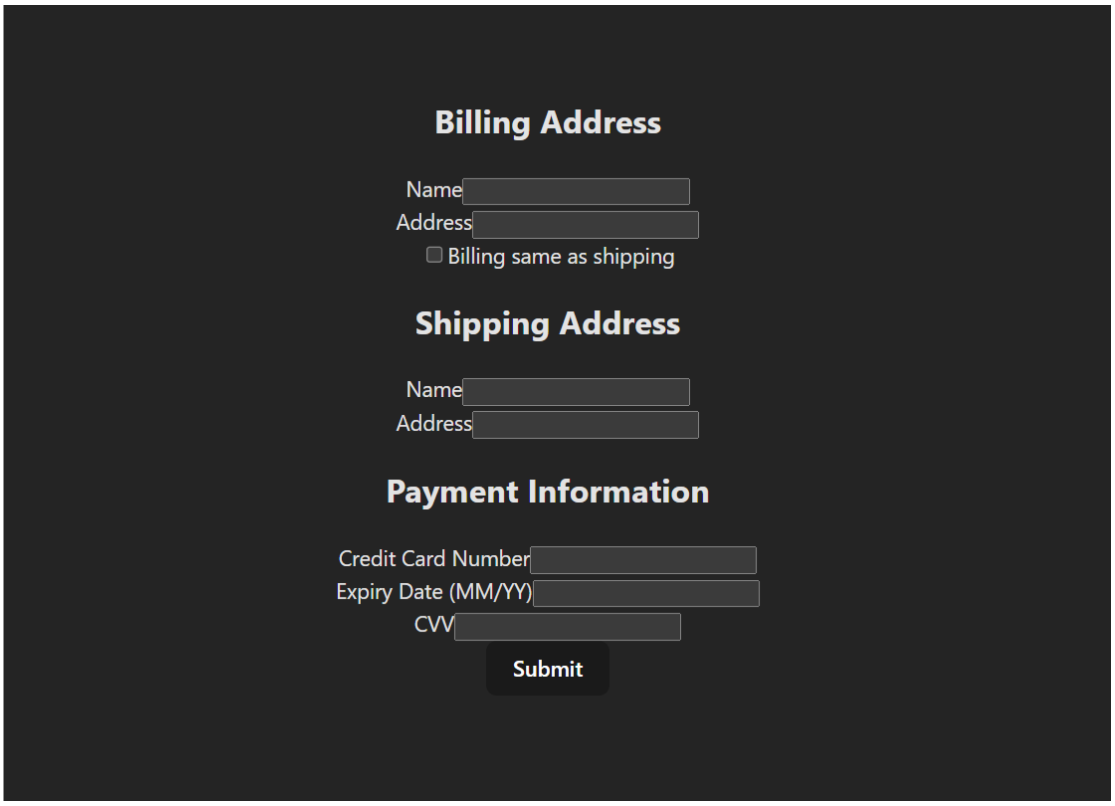
```
      <div>
        <label htmlFor="expiryDate">Expiry Date (MM/YY)</label>
        <input
          id="expiryDate"
          {...register('expiryDate', { required: 'Expiry date is required!' })}
        />
        {errors.expiryDate && <p>{errors.expiryDate.message}</p>}
      </div>
```

```
      <div>
        <label htmlFor="cvv">CVV</label>
        <input
          id="cvv"
          {...register('cvv', {
            required: 'CVV is required!',
            validate: validateCVV,
          })}
        />
        {errors.cvv && <p>{errors.cvv.message}</p>}
      </div>
```

```
      <button type="submit">Submit</button>
    </form>
  );
};
```

```
export default CheckoutForm;
```

OUTPUT



Billing Address

Name
Address
☐ Billing same as shipping

Shipping Address

Name
Address

Payment Information

Credit Card Number
Expiry Date (MM/YY)
CVV
Submit

Q1. TimeManage.jsx

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

export default function TimeManage() {
  const [projectName, setProjectName] = useState("");
  const [hoursLogged, setHoursLogged] = useState("");
  const [logs, setLogs] = useState({});
```

```
  const handleLogHours = () => {
    if (!projectName || !hoursLogged || isNaN(hoursLogged)) {
      alert("Please enter a valid project name and hours!");
      return;
    }
```

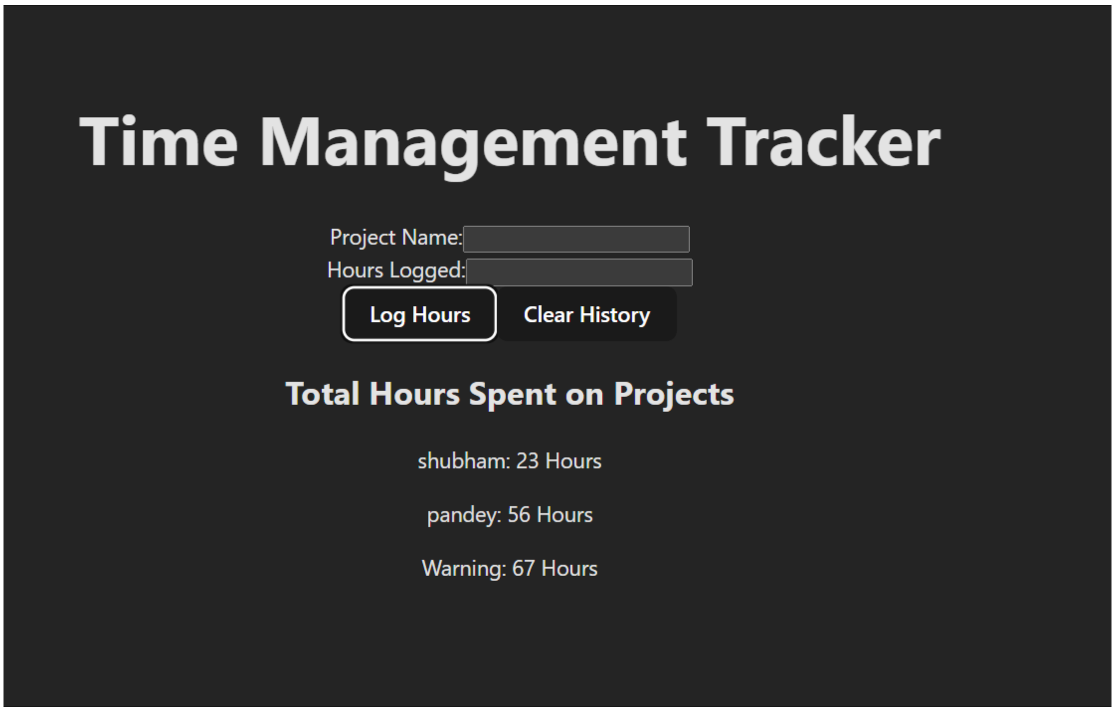
```
    // Update the logs for the specific project
    setLogs((prevLogs) => {
      const currentHours = prevLogs[projectName] || 0;
      return {
        ...prevLogs,
        [projectName]: currentHours + parseFloat(hoursLogged),
      };
    });
    // Clear input fields after logging
    setProjectName("");
    setHoursLogged("");
  };
```

```
  const handleClearHistory = () => {
    setLogs({});
  };
```

```
  return (
    <div>
      <h1>Time Management Tracker</h1>
      <label>
        Project Name:
        <input
          type="text"
          value={projectName}
          onChange={(e) => setProjectName(e.target.value)}
        />
      </label>
      <br />
      <label>
        Hours Logged:
        <input
          type="text"
          value={hoursLogged}
          onChange={(e) => setHoursLogged(e.target.value)}
        />
      </label>
      <br />
      <button onClick={handleLogHours}>Log Hours</button>
      <button onClick={handleClearHistory}>Clear History</button>
```

```
      <h2>Total Hours Spent on Projects</h2>
      {Object.keys(logs).length === 0 ? (
        <p>No logs yet.</p>
      ) : (
        Object.entries(logs).map(([project, hours]) => (
          <p key={project}>
            {project}: {hours} Hours
          </p>
        ))
      )}
    </div>
  );
}
```

Output



Time Management Tracker

Project Name:

Hours Logged:

Total Hours Spent on Projects

shubham: 23 Hours

pandey: 56 Hours

Warning: 67 Hours