Complete React Forms Guide

Table of Contents

---

1. Creating Forms in React

Forms in React are used to collect user input and handle form submissions. React provides two main approaches to handle forms: controlled and uncontrolled components.

Basic Form Structure

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

function BasicForm() {
  const handleSubmit = (event) => {
    event.preventDefault();
    // Handle form submission
    console.log('Form submitted');
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" placeholder="Enter text" />
      <button type="submit">Submit</button>
    </form>
  );
}
```

Key Concepts

- **Form elements**: input, textarea, select, button
- **Event handling**: onSubmit, onChange, onBlur, onFocus
- **Preventing default behavior**: `event.preventDefault()`

---

2. Controlled Form Components

Controlled components are form elements whose value is controlled by React state. The component's state becomes the "single source of truth" for the form data.

Characteristics

- Form data is handled by React component state
- Value is controlled by React state
- Changes are handled through event handlers
- Provides better control and validation

Example: Controlled Input

jsx

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

function ControlledForm() {
  const [formData, setFormData] = useState({
    username: '',
    email: '',
    message: ''
  });

  const handleInputChange = (event) => {
    const { name, value } = event.target;
    setFormData(prevState => ({
      ...prevState,
      [name]: value
    }));
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    console.log('Form Data:', formData);
  };

  return (
    <form onSubmit={handleSubmit}>
      <div>
        <label htmlFor="username">Username:</label>
        <input
          type="text"
          id="username"
          name="username"
          value={formData.username}
          onChange={handleInputChange}
          placeholder="Enter username"
        />
      </div>

      <div>
        <label htmlFor="email">Email:</label>
        <input
          type="email"
          id="email"
          name="email"
          value={formData.email}
          onChange={handleInputChange}
          placeholder="Enter email"
        />
      </div>

      <div>
        <label htmlFor="message">Message:</label>
        <textarea
          id="message"
          name="message"
          value={formData.message}
          onChange={handleInputChange}
          placeholder="Enter message"
          rows="4"
        />
      </div>

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

Benefits of Controlled Components

- Immediate validation
- Conditional rendering based on input
- Dynamic form behavior
- Easy to debug and test

---

3. Uncontrolled Form Components

Uncontrolled components let the DOM handle the form data. You use refs to access form values when needed.

Characteristics

- Form data is handled by DOM
- Use refs to access values
- Less code for simple forms
- Limited control over form behavior

Example: Uncontrolled Form

jsx

```
import React, { useRef } from 'react';

function UncontrolledForm() {
  const usernameRef = useRef();
  const emailRef = useRef();
  const messageRef = useRef();

  const handleSubmit = (event) => {
    event.preventDefault();

    const formData = {
      username: usernameRef.current.value,
      email: emailRef.current.value,
      message: messageRef.current.value
    };

    console.log('Form Data:', formData);
  };

  return (
    <form onSubmit={handleSubmit}>
      <div>
        <label htmlFor="username">Username:</label>
        <input
          type="text"
          id="username"
          ref={usernameRef}
          defaultValue=""
          placeholder="Enter username"
        />
      </div>
      <div>
        <label htmlFor="email">Email:</label>
        <input
          type="email"
          id="email"
          ref={emailRef}
          defaultValue=""
          placeholder="Enter email"
        />
      </div>

      <div>
        <label htmlFor="message">Message:</label>
        <textarea
          id="message"
          ref={messageRef}
          defaultValue=""
          placeholder="Enter message"
          rows="4"
        />
      </div>

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

When to Use Uncontrolled Components

- Simple forms with minimal validation
- Integrating with non-React code

- File uploads
- When you need to optimize performance

---

4. Form Elements

Textbox (Input)

```jsx
function TextboxExample() {
  const [text, setText] = useState('');

  return (
    <div>
      <label htmlFor="textInput">Text Input:</label>
      <input
        type="text"
        id="textInput"
        value={text}
        onChange={(e) => setText(e.target.value)}
        placeholder="Enter text here"
      />
      <p>You typed: {text}</p>
    </div>
  );
}
```

Dropdown (Select)

jsx

```jsx
function DropdownExample() {
  const [selectedOption, setSelectedOption] = useState('');

  const options = [
    { value: '', label: 'Select an option' },
    { value: 'option1', label: 'Option 1' },
    { value: 'option2', label: 'Option 2' },
    { value: 'option3', label: 'Option 3' }
  ];

  return (
    <div>
      <label htmlFor="dropdown">Choose an option:</label>
      <select
        id="dropdown"
        value={selectedOption}
        onChange={(e) => setSelectedOption(e.target.value)}
      >
        {options.map(option => (
          <option key={option.value} value={option.value}>
            {option.label}
          </option>
        ))}
      </select>
      <p>Selected: {selectedOption}</p>
    </div>
  );
}
```

Radio Buttons

jsx

```
function RadioExample() {
  const [selectedRadio, setSelectedRadio] = useState('');

  const radioOptions = [
    { value: 'small', label: 'Small' },
    { value: 'medium', label: 'Medium' },
    { value: 'large', label: 'Large' }
  ];

  return (
    <div>
      <fieldset>
        <legend>Choose size:</legend>
        {radioOptions.map(option => (
          <div key={option.value}>
            <input
              type="radio"
              id={option.value}
              name="size"
              value={option.value}
              checked={selectedRadio === option.value}
              onChange={(e) => setSelectedRadio(e.target.value)}
            />
            <label htmlFor={option.value}>{option.label}</label>
          </div>
        ))}
      </fieldset>
      <p>Selected size: {selectedRadio}</p>
    </div>
  );
}
```

Checkbox

jsx

```
function CheckboxExample() {
  const [checkedItems, setCheckedItems] = useState({
    option1: false,
    option2: false,
    option3: false
  });

  const handleCheckboxChange = (event) => {
    const { name, checked } = event.target;
    setCheckedItems(prevState => ({
      ...prevState,
      [name]: checked
    }));
  };

  return (
    <div>
      <fieldset>
        <legend>Select options:</legend>
        <div>
          <input
            type="checkbox"
            id="option1"
            name="option1"
            checked={checkedItems.option1}
            onChange={handleCheckboxChange}
          />
          <label htmlFor="option1">Option 1</label>
        </div>

        <div>
          <input
```

```
            type="checkbox"
            id="option2"
            name="option2"
            checked={checkedItems.option2}
            onChange={handleCheckboxChange}
          />
          <label htmlFor="option2">Option 2</label>
        </div>

        <div>
          <input
            type="checkbox"
            id="option3"
            name="option3"
            checked={checkedItems.option3}
            onChange={handleCheckboxChange}
          />
          <label htmlFor="option3">Option 3</label>
        </div>
      </fieldset>

      <p>Selected: {Object.keys(checkedItems).filter(key => checkedItems[key]).join(', ')
    </div>
  );
}
```

5. Select Box with Default Selected Value

jsx

```jsx
function SelectWithDefault() {
  const [country, setCountry] = useState('usa'); // Default value

  const countries = [
    { value: 'usa', label: 'United States' },
    { value: 'canada', label: 'Canada' },
    { value: 'uk', label: 'United Kingdom' },
    { value: 'australia', label: 'Australia' },
    { value: 'india', label: 'India' }
  ];

  return (
    <div>
      <label htmlFor="country">Country:</label>
      <select
        id="country"
        value={country}
        onChange={(e) => setCountry(e.target.value)}
      >
        {countries.map(country => (
          <option key={country.value} value={country.value}>
            {country.label}
          </option>
        ))}
      </select>
      <p>Selected country: {country}</p>
    </div>
  );
}
```

Multiple Select

jsx

```
function MultipleSelect() {
  const [selectedLanguages, setSelectedLanguages] = useState(['javascript']);

  const languages = [
    { value: 'javascript', label: 'JavaScript' },
    { value: 'python', label: 'Python' },
    { value: 'java', label: 'Java' },
    { value: 'csharp', label: 'C#' },
    { value: 'php', label: 'PHP' }
  ];

  const handleMultipleSelect = (event) => {
    const values = Array.from(event.target.selectedOptions, option => option.value);
    setSelectedLanguages(values);
  };

  return (
    <div>
      <label htmlFor="languages">Programming Languages:</label>
      <select
        id="languages"
        multiple
        value={selectedLanguages}
        onChange={handleMultipleSelect}
      >
        {languages.map(lang => (
          <option key={lang.value} value={lang.value}>
            {lang.label}
          </option>
        ))}
      </select>
      <p>Selected: {selectedLanguages.join(', ')}</p>
    </div>
  );
}
```

---

6. Form Validation

Basic Validation

jsx

```
function ValidatedForm() {
  const [formData, setFormData] = useState({
    email: '',
    password: '',
    confirmPassword: ''
  });

  const [errors, setErrors] = useState({});

  const validateForm = () => {
    const newErrors = {};

    // Email validation
    if (!formData.email) {
      newErrors.email = 'Email is required';
    } else if (!/\S+@\S+\.\S+/.test(formData.email)) {
      newErrors.email = 'Email is invalid';
    }

    // Password validation
    if (!formData.password) {
      newErrors.password = 'Password is required';
    } else if (formData.password.length < 6) {
      newErrors.password = 'Password must be at least 6 characters';
    }

    // Confirm password validation
    if (formData.password !== formData.confirmPassword) {
      newErrors.confirmPassword = 'Passwords do not match';
    }

    setErrors(newErrors);
  return Object.keys(newErrors).length === 0;
};

const handleInputChange = (event) => {
  const { name, value } = event.target;
  setFormData(prevState => ({
    ...prevState,
    [name]: value
  }));

  // Clear error when user starts typing
  if (errors[name]) {
    setErrors(prevErrors => ({
      ...prevErrors,
      [name]: ''
    }));
  }
};

const handleSubmit = (event) => {
  event.preventDefault();
  if (validateForm()) {
    console.log('Form is valid:', formData);
  }
};

return (
  <form onSubmit={handleSubmit}>
    <div>
      <label htmlFor="email">Email:</label>
      <input
        type="email"
      id="email"
      name="email"
      value={formData.email}
      onChange={handleInputChange}
      className={errors.email ? 'error' : ''}
    />
    {errors.email && <span className="error-message">{errors.email}</span>}
  </div>

  <div>
    <label htmlFor="password">Password:</label>
    <input
      type="password"
      id="password"
      name="password"
      value={formData.password}
      onChange={handleInputChange}
      className={errors.password ? 'error' : ''}
    />
    {errors.password && <span className="error-message">{errors.password}</span>}
  </div>

  <div>
    <label htmlFor="confirmPassword">Confirm Password:</label>
    <input
      type="password"
      id="confirmPassword"
      name="confirmPassword"
      value={formData.confirmPassword}
      onChange={handleInputChange}
      className={errors.confirmPassword ? 'error' : ''}
    />
```

```
        {errors.confirmPassword && <span className="error-message">{errors.confirmPassword
      </div>

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

Real-time Validation

jsx

```
function RealTimeValidation() {
  const [username, setUsername] = useState('');
  const [isValid, setIsValid] = useState(null);

  const validateUsername = (value) => {
    if (value.length === 0) {
      setIsValid(null);
    } else if (value.length < 3) {
      setIsValid(false);
    } else if (!/^[a-zA-Z0-9_]+$/.test(value)) {
      setIsValid(false);
    } else {
      setIsValid(true);
    }
  };

  const handleUsernameChange = (event) => {
    const value = event.target.value;
    setUsername(value);
    validateUsername(value);
  };

  return (
    <div>
      <label htmlFor="username">Username:</label>
      <input
        type="text"
        id="username"
        value={username}
        onChange={handleUsernameChange}
        className={isValid === false ? 'invalid' : isValid === true ? 'valid' : ''}
      />
```

```
        {isValid === false && (
          <span className="error-message">
            Username must be at least 3 characters and contain only letters, numbers, and u
          </span>
        )}
        {isValid === true && (
          <span className="success-message">Username is valid!</span>
        )}
      </div>
    );
  }
```

---

7. Introduction to Refs

Refs provide a way to access DOM nodes or React elements directly. They're useful for focus management, triggering animations, or integrating with third-party libraries.

Creating Refs

jsx

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

function RefExample() {
  const inputRef = useRef(null);
  const divRef = useRef(null);

  useEffect(() => {
    // Focus input on component mount
    inputRef.current.focus();
  }, []);

  const handleFocusInput = () => {
    inputRef.current.focus();
  };

  const handleScrollToDiv = () => {
    divRef.current.scrollIntoView({ behavior: 'smooth' });
  };

  return (
    <div>
      <input ref={inputRef} type="text" placeholder="This will be focused on mount" />
      <button onClick={handleFocusInput}>Focus Input</button>

      <div style={{ height: '1000px' }}>Scroll down...</div>

      <div ref={divRef} style={{ backgroundColor: 'lightblue', padding: '20px' }}>
        Target div for scrolling
      </div>

      <button onClick={handleScrollToDiv}>Scroll to Div</button>
    </div>
    );
  }
```

Ref with Form Elements

jsx

```
function RefFormExample() {
  const fileInputRef = useRef(null);
  const formRef = useRef(null);

  const handleFileSelect = () => {
    fileInputRef.current.click();
  };

  const handleResetForm = () => {
    formRef.current.reset();
  };

  const handleFileChange = (event) => {
    const file = event.target.files[0];
    if (file) {
      console.log('Selected file:', file.name);
    }
  };

  return (
    <form ref={formRef}>
      <input
        type="file"
        ref={fileInputRef}
        onChange={handleFileChange}
        style={{ display: 'none' }}
      />

      <input type="text" placeholder="Enter text" />

      <button type="button" onClick={handleFileSelect}>
        Select File
        </button>

        <button type="button" onClick={handleResetForm}>
          Reset Form
        </button>
      </form>
    );
  }
```

Forward Refs

jsx

```jsx
import React, { forwardRef, useRef } from 'react';

const CustomInput = forwardRef((props, ref) => {
  return (
    <input
      ref={ref}
      {...props}
      style={{
        padding: '10px',
        border: '2px solid #ccc',
        borderRadius: '4px'
      }}
    />
  );
});

function ForwardRefExample() {
  const customInputRef = useRef(null);

  const handleFocus = () => {
    customInputRef.current.focus();
  };

  return (
    <div>
      <CustomInput ref={customInputRef} placeholder="Custom input component" />
      <button onClick={handleFocus}>Focus Custom Input</button>
    </div>
  );
}
```

8. Styles

Inline Styles

jsx

```
function InlineStylesForm() {
  const [formData, setFormData] = useState({ name: '', email: '' });

  const formStyle = {
    maxWidth: '400px',
    margin: '0 auto',
    padding: '20px',
    backgroundColor: '#f9f9f9',
    borderRadius: '8px',
    boxShadow: '0 2px 10px rgba(0,0,0,0.1)'
  };

  const inputStyle = {
    width: '100%',
    padding: '12px',
    marginBottom: '15px',
    border: '1px solid #ddd',
    borderRadius: '4px',
    fontSize: '16px'
  };

  const buttonStyle = {
    width: '100%',
    padding: '12px',
    backgroundColor: '#007bff',
    color: 'white',
    border: 'none',
    borderRadius: '4px',
    fontSize: '16px',
    cursor: 'pointer'
  };
  const handleInputChange = (event) => {
    const { name, value } = event.target;
    setFormData(prev => ({ ...prev, [name]: value }));
  };

  return (
    <form style={formStyle}>
      <input
        type="text"
        name="name"
        placeholder="Full Name"
        value={formData.name}
        onChange={handleInputChange}
        style={inputStyle}
      />

      <input
        type="email"
        name="email"
        placeholder="Email Address"
        value={formData.email}
        onChange={handleInputChange}
        style={inputStyle}
      />

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

CSS Classes

jsx

```
// CSS file (styles.css)
/*
.form-container {
  max-width: 400px;
  margin: 0 auto;
  padding: 20px;
  background-color: #f9f9f9;
  border-radius: 8px;
  box-shadow: 0 2px 10px rgba(0,0,0,0.1);
}

.form-input {
  width: 100%;
  padding: 12px;
  margin-bottom: 15px;
  border: 1px solid #ddd;
  border-radius: 4px;
  font-size: 16px;
}

.form-input:focus {
  outline: none;
  border-color: #007bff;
  box-shadow: 0 0 0 2px rgba(0,123,255,0.25);
}

.form-input.error {
  border-color: #dc3545;
}

.submit-button {
  width: 100%;
  padding: 12px;
  background-color: #007bff;
  color: white;
  border: none;
  border-radius: 4px;
  font-size: 16px;
  cursor: pointer;
  transition: background-color 0.3s ease;
}

.submit-button:hover {
  background-color: #0056b3;
}

.error-message {
  color: #dc3545;
  font-size: 14px;
  margin-top: -10px;
  margin-bottom: 10px;
  display: block;
}
*/

function StyledForm() {
  const [formData, setFormData] = useState({ name: '', email: '' });
  const [errors, setErrors] = useState({});

  const handleInputChange = (event) => {
    const { name, value } = event.target;
    setFormData(prev => ({ ...prev, [name]: value }));
  };
  return (
    <form className="form-container">
      <input
        type="text"
        name="name"
        placeholder="Full Name"
        value={formData.name}
        onChange={handleInputChange}
        className={`form-input ${errors.name ? 'error' : ''}`}
      />
      {errors.name && <span className="error-message">{errors.name}</span>}

      <input
        type="email"
        name="email"
        placeholder="Email Address"
        value={formData.email}
        onChange={handleInputChange}
        className={`form-input ${errors.email ? 'error' : ''}`}
      />
      {errors.email && <span className="error-message">{errors.email}</span>}

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

Styled Components (if using styled-components library)

jsx

```
import styled from 'styled-components';

const FormContainer = styled.form`
  max-width: 400px;
  margin: 0 auto;
  padding: 20px;
  background-color: #f9f9f9;
  border-radius: 8px;
  box-shadow: 0 2px 10px rgba(0,0,0,0.1);
`;

const FormInput = styled.input`
  width: 100%;
  padding: 12px;
  margin-bottom: 15px;
  border: 1px solid ${props => props.error ? '#dc3545' : '#ddd'};
  border-radius: 4px;
  font-size: 16px;

  &:focus {
    outline: none;
    border-color: #007bff;
    box-shadow: 0 0 0 2px rgba(0,123,255,0.25);
  }
`;

const SubmitButton = styled.button`
  width: 100%;
  padding: 12px;
  background-color: #007bff;
  color: white;
  border: none;
  border-radius: 4px;
  font-size: 16px;
  cursor: pointer;
  transition: background-color 0.3s ease;

  &:hover {
    background-color: #0056b3;
  }
`;

function StyledComponentsForm() {
  const [formData, setFormData] = useState({ name: '', email: '' });

  return (
    <FormContainer>
      <FormInput
        type="text"
        placeholder="Full Name"
        value={formData.name}
        onChange={(e) => setFormData(prev => ({ ...prev, name: e.target.value }))}
      />

      <FormInput
        type="email"
        placeholder="Email Address"
        value={formData.email}
        onChange={(e) => setFormData(prev => ({ ...prev, email: e.target.value }))}
      />

      <SubmitButton type="submit">
        Submit
      </SubmitButton>
    </FormContainer>
  );
}
```

---

9. Animation Introduction

CSS Transitions

jsx

```
function AnimatedForm() {
  const [isVisible, setIsVisible] = useState(false);
  const [isSubmitting, setIsSubmitting] = useState(false);

  const formStyle = {
    maxWidth: '400px',
    margin: '20px auto',
    padding: '20px',
    backgroundColor: '#fff',
    borderRadius: '8px',
    boxShadow: '0 2px 20px rgba(0,0,0,0.1)',
    transform: isVisible ? 'translateY(0)' : 'translateY(-20px)',
    opacity: isVisible ? 1 : 0,
    transition: 'all 0.3s ease-in-out'
  };

  const buttonStyle = {
    width: '100%',
    padding: '12px',
    backgroundColor: isSubmitting ? '#28a745' : '#007bff',
    color: 'white',
    border: 'none',
    borderRadius: '4px',
    cursor: 'pointer',
    transform: isSubmitting ? 'scale(0.98)' : 'scale(1)',
    transition: 'all 0.2s ease'
  };

  useEffect(() => {
    setIsVisible(true);
  }, []);
  const handleSubmit = async (event) => {
    event.preventDefault();
    setIsSubmitting(true);

    // Simulate API call
    setTimeout(() => {
      setIsSubmitting(false);
    }, 2000);
  };

  return (
    <form style={formStyle} onSubmit={handleSubmit}>
      <input
        type="text"
        placeholder="Name"
        style={{
          width: '100%',
          padding: '12px',
          marginBottom: '15px',
          border: '1px solid #ddd',
          borderRadius: '4px',
          transition: 'border-color 0.3s ease'
        }}
      />

      <button type="submit" style={buttonStyle} disabled={isSubmitting}>
        {isSubmitting ? 'Submitting...' : 'Submit'}
      </button>
    </form>
  );
}
```

Keyframe Animations

jsx

```
function KeyframeAnimatedForm() {
  const [showSuccess, setShowSuccess] = useState(false);

  const successStyle = {
    padding: '10px',
    backgroundColor: '#d4edda',
    color: '#155724',
    border: '1px solid #c3e6cb',
    borderRadius: '4px',
    marginTop: '10px',
    animation: showSuccess ? 'slideIn 0.5s ease-out' : 'slideOut 0.3s ease-in',
    opacity: showSuccess ? 1 : 0
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    setShowSuccess(true);

    setTimeout(() => {
      setShowSuccess(false);
    }, 3000);
  };

  return (
    <div>
      <style>
        {`
          @keyframes slideIn {
            from {
              transform: translateX(-100%);
              opacity: 0;
            }
      to {
        transform: translateX(0);
        opacity: 1;
      }
    }

    @keyframes slideOut {
      from {
        transform: translateX(0);
        opacity: 1;
      }
      to {
        transform: translateX(-100%);
        opacity: 0;
      }
    }

    .pulse {
      animation: pulse 2s infinite;
    }

    @keyframes pulse {
      0% { transform: scale(1); }
      50% { transform: scale(1.05); }
      100% { transform: scale(1); }
    }
  `}
</style>

<form onSubmit={handleSubmit}>
  <input
    type="text"
```

```
          placeholder="Enter your name"
          className="pulse"
          style={{
            width: '100%',
            padding: '12px',
            marginBottom: '15px',
            border: '2px solid #007bff',
            borderRadius: '25px'
          }}
        />

        <button type="submit">Submit</button>

        {showSuccess && (
          <div style={successStyle}>
            Form submitted successfully!
          </div>
        )}
      </form>
    </div>
  );
}
```

React Transition Group (if using react-transition-group)

jsx

```
import { CSSTransition, TransitionGroup } from 'react-transition-group';

function TransitionGroupForm() {
  const [fields, setFields] = useState([{ id: 1, value: '' }]);

  const addField = () => {
    const newField = { id: Date.now(), value: '' };
    setFields([...fields, newField]);
  };

  const removeField = (id) => {
    setFields(fields.filter(field => field.id !== id));
  };

  return (
    <div>
      <style>
        {`
          .field-enter {
            opacity: 0;
            transform: translateX(-100%);
          }

          .field-enter-active {
            opacity: 1;
            transform: translateX(0);
            transition: opacity 300ms, transform 300ms;
          }

          .field-exit {
            opacity: 1;
            transform: translateX(0);
      }

      .field-exit-active {
        opacity: 0;
        transform: translateX(100%);
        transition: opacity 300ms, transform 300ms;
      }
    `}
  </style>

  <form>
    <TransitionGroup>
      {fields.map(field => (
        <CSSTransition
          key={field.id}
          timeout={300}
          classNames="field"
        >
          <div style={{ marginBottom: '10px' }}>
            <input
              type="text"
              value={field.value}
              onChange={(e) => {
                const updatedFields = fields.map(f =>
                  f.id === field.id ? { ...f, value: e.target.value } : f
                );
                setFields(updatedFields);
              }}
              placeholder={`Field ${field.id}`}
              style={{ marginRight: '10px', padding: '8px' }}
            />
            <button
```

```
                  type="button"
                  onClick={() => removeField(field.id)}
                  disabled={fields.length === 1}
                >
                  Remove
                </button>
              </div>
            </CSSTransition>
          ))}
        </TransitionGroup>

        <button type="button" onClick={addField}>
          Add Field
        </button>
      </form>
    </div>
  );
}
```

---

10. Introduction to Formik Library

Formik is a popular library for building forms in React. It helps with form validation, error handling, and form submission while reducing boilerplate code.

Installation

```bash
npm install formik yup
# or
yarn add formik yup
```

Basic Formik Example

jsx

```
import { Formik, Form, Field, ErrorMessage } from 'formik';
import * as Yup from 'yup';

// Validation schema using Yup
const validationSchema = Yup.object({
  firstName: Yup.string()
    .max(15, 'Must be 15 characters or less')
    .required('Required'),
  lastName: Yup.string()
    .max(20, 'Must be 20 characters or less')
    .required('Required'),
  email: Yup.string()
    .email('Invalid email address')
    .required('Required'),
  age: Yup.number()
    .min(18, 'Must be at least 18 years old')
    .required('Required')
});

function BasicFormikForm() {
  const initialValues = {
    firstName: '',
    lastName: '',
    email: '',
    age: ''
  };

  const handleSubmit = (values, { setSubmitting, resetForm }) => {
    setTimeout(() => {
      console.log('Form submitted:', values);
      setSubmitting(false);
      resetForm();
  }, 400);
};

return (
  <Formik
    initialValues={initialValues}
    validationSchema={validationSchema}
    onSubmit={handleSubmit}
  >
    {({ isSubmitting }) => (
      <Form>
        <div>
          <label htmlFor="firstName">First Name:</label>
          <Field
            type="text"
            id="firstName"
            name="firstName"
            style={{ marginLeft: '10px', padding: '5px' }}
          />
          <ErrorMessage name="firstName" component="div" style={{ color: 'red' }} />
        </div>

        <div>
          <label htmlFor="lastName">Last Name:</label>
          <Field
            type="text"
            id="lastName"
            name="lastName"
            style={{ marginLeft: '10px', padding: '5px' }}
          />
          <ErrorMessage name="lastName" component="div" style={{ color: 'red' }} />
        </div>
```

```
          <div>
            <label htmlFor="email">Email:</label>
            <Field
              type="email"
              id="email"
              name="email"
              style={{ marginLeft: '10px', padding: '5px' }}
            />
            <ErrorMessage name="email" component="div" style={{ color: 'red' }} />
          </div>

          <div>
            <label htmlFor="age">Age:</label>
            <Field
              type="number"
              id="age"
              name="age"
              style={{ marginLeft: '10px', padding: '5px' }}
            />
            <ErrorMessage name="age" component="div" style={{ color: 'red' }} />
          </div>

          <button type="submit" disabled={isSubmitting}>
            {isSubmitting ? 'Submitting...' : 'Submit'}
          </button>
        </Form>
      )}
    </Formik>
  );
}
```

Advanced Formik with Custom Components

jsx

```
import { Formik, Form, Field, FieldArray, ErrorMessage } from 'formik';
import * as Yup from 'yup';

const validationSchema = Yup.object({
  user: Yup.object({
    name: Yup.string().required('Name is required'),
    email: Yup.string().email('Invalid email').required('Email is required')
  }),
  hobbies: Yup.array().of(
    Yup.string().required('Hobby is required')
  ).min(1, 'At least one hobby is required'),
  preferences: Yup.object({
    newsletter: Yup.boolean(),
    notifications: Yup.boolean()
  })
});

function AdvancedFormikForm() {
  const initialValues = {
    user: {
      name: '',
      email: ''
    },
    hobbies: [''],
    preferences: {
      newsletter: false,
      notifications: true
    }
  };

  const handleSubmit = (values, actions) => {
    console.log('Advanced form submitted:', values);
  actions.setSubmitting(false);
};

return (
  <Formik
    initialValues={initialValues}
    validationSchema={validationSchema}
    onSubmit={handleSubmit}
  >
    {({ values, isSubmitting }) => (
      <Form>
        <h3>User Information</h3>
        <div>
          <label htmlFor="user.name">Name:</label>
          <Field name="user.name" type="text" />
          <ErrorMessage name="user.name" component="div" style={{ color: 'red' }} />
        </div>

        <div>
          <label htmlFor="user.email">Email:</label>
          <Field name="user.email" type="email" />
          <ErrorMessage name="user.email" component="div" style={{ color: 'red' }} />
        </div>

        <h3>Hobbies</h3>
        <FieldArray name="hobbies">
          {({ push, remove }) => (
            <div>
              {values.hobbies.map((hobby, index) => (
                <div key={index} style={{ marginBottom: '10px' }}>
                  <Field
                    name={`hobbies.${index}`}
            type="text"
            placeholder="Enter hobby"
          />
          <button
            type="button"
            onClick={() => remove(index)}
            disabled={values.hobbies.length === 1}
            style={{ marginLeft: '10px' }}
          >
            Remove
          </button>
          <ErrorMessage
            name={`hobbies.${index}`}
            component="div"
            style={{ color: 'red' }}
          />
        </div>
      ))}
      <button type="button" onClick={() => push('')}>
        Add Hobby
      </button>
    </div>
  )}
</FieldArray>

<h3>Preferences</h3>
<div>
  <label>
    <Field name="preferences.newsletter" type="checkbox" />
    Subscribe to newsletter
  </label>
</div>
```

```
          <div>
            <label>
              <Field name="preferences.notifications" type="checkbox" />
              Enable notifications
            </label>
          </div>

          <button type="submit" disabled={isSubmitting}>
            Submit
          </button>
        </Form>
      )}
    </Formik>
  );
}
```

Formik with Custom Field Components

jsx

```
import { Formik, Form, Field, ErrorMessage } from 'formik';
import * as Yup from 'yup';

// Custom Input Component
const CustomInput = ({ field, form, ...props }) => {
  const hasError = form.touched[field.name] && form.errors[field.name];

  return (
    <input
      {...field}
      {...props}
      style={{
        padding: '10px',
        border: hasError ? '2px solid red' : '1px solid #ccc',
        borderRadius: '4px',
        width: '100%'
      }}
    />
  );
};

// Custom Select Component
const CustomSelect = ({ field, form, options, ...props }) => {
  return (
    <select
      {...field}
      {...props}
      style={{
        padding: '10px',
        border: '1px solid #ccc',
        borderRadius: '4px',
        width: '100%'
      }}
    >
      <option value="">Select an option</option>
      {options.map(option => (
        <option key={option.value} value={option.value}>
          {option.label}
        </option>
      ))}
    </select>
  );
};

const validationSchema = Yup.object({
  name: Yup.string().required('Name is required'),
  country: Yup.string().required('Country is required'),
  bio: Yup.string().max(200, 'Bio must be 200 characters or less')
});

function CustomComponentsForm() {
  const countries = [
    { value: 'us', label: 'United States' },
    { value: 'ca', label: 'Canada' },
    { value: 'uk', label: 'United Kingdom' },
    { value: 'au', label: 'Australia' }
  ];

  const initialValues = {
    name: '',
    country: '',
    bio: ''
  };
  return (
    <Formik
      initialValues={initialValues}
      validationSchema={validationSchema}
      onSubmit={(values) => console.log('Custom form submitted:', values)}
    >
      <Form>
        <div style={{ marginBottom: '15px' }}>
          <label htmlFor="name">Name:</label>
          <Field name="name" component={CustomInput} />
          <ErrorMessage name="name" component="div" style={{ color: 'red' }} />
        </div>

        <div style={{ marginBottom: '15px' }}>
          <label htmlFor="country">Country:</label>
          <Field name="country" component={CustomSelect} options={countries} />
          <ErrorMessage name="country" component="div" style={{ color: 'red' }} />
        </div>

        <div style={{ marginBottom: '15px' }}>
          <label htmlFor="bio">Bio:</label>
          <Field name="bio" as="textarea" rows="4" style={{ width: '100%', padding: '10px
          <ErrorMessage name="bio" component="div" style={{ color: 'red' }} />
        </div>

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

Formik Hooks (useFormik)

jsx

```
import { useFormik } from 'formik';
import * as Yup from 'yup';

function FormikHooksExample() {
  const formik = useFormik({
    initialValues: {
      email: '',
      password: ''
    },
    validationSchema: Yup.object({
      email: Yup.string()
        .email('Invalid email address')
        .required('Required'),
      password: Yup.string()
        .min(6, 'Password must be at least 6 characters')
        .required('Required')
    }),
    onSubmit: (values, { setSubmitting, resetForm }) => {
      setTimeout(() => {
        console.log('Login form submitted:', values);
        setSubmitting(false);
        resetForm();
      }, 400);
    }
  });

  return (
    <form onSubmit={formik.handleSubmit}>
      <div>
        <label htmlFor="email">Email:</label>
        <input
          id="email"
    name="email"
    type="email"
    onChange={formik.handleChange}
    onBlur={formik.handleBlur}
    value={formik.values.email}
    style={{
      padding: '10px',
      marginLeft: '10px',
      border: formik.touched.email && formik.errors.email ? '1px solid red' : '1px
    }}
  />
  {formik.touched.email && formik.errors.email && (
    <div style={{ color: 'red' }}>{formik.errors.email}</div>
  )}
</div>

<div>
  <label htmlFor="password">Password:</label>
  <input
    id="password"
    name="password"
    type="password"
    onChange={formik.handleChange}
    onBlur={formik.handleBlur}
    value={formik.values.password}
    style={{
      padding: '10px',
      marginLeft: '10px',
      border: formik.touched.password && formik.errors.password ? '1px solid red' :
    }}
  />
  {formik.touched.password && formik.errors.password && (
          <div style={{ color: 'red' }}>{formik.errors.password}</div>
        )}
      </div>

      <button type="submit" disabled={formik.isSubmitting}>
        {formik.isSubmitting ? 'Logging in...' : 'Login'}
      </button>
    </form>
  );
}
```

Key Takeaways

When to Use Each Approach

Controlled Components:

- When you need real-time validation
- When form data affects other parts of the UI
- When you need to transform user input
- For most React applications (recommended)

Uncontrolled Components:

- Simple forms with minimal validation
- When integrating with non-React code
- File uploads
- Performance optimization scenarios

Formik:

- Complex forms with extensive validation
- When you want to reduce boilerplate code
- Enterprise applications
- When you need advanced form features

Best Practices

1. **Always use labels** for accessibility
2. **Provide meaningful error messages**
3. **Use appropriate input types** (email, password, number, etc.)
4. **Implement proper validation** (client-side and server-side)
5. **Handle loading states** during form submission
6. **Use semantic HTML** elements
7. **Make forms responsive** for mobile devices
8. **Provide visual feedback** for user actions
9. **Test forms thoroughly** across different devices and browsers
10. **Consider user experience** in form design

Common Patterns

- **Multi-step forms**: Break long forms into multiple steps
- **Conditional fields**: Show/hide fields based on user input
- **Auto-save**: Save form data as user types
- **Confirmation dialogs**: Confirm before destructive actions

- **Progress indicators**: Show form completion progress

This comprehensive guide covers all the essential aspects of working with forms in React, from basic concepts to advanced techniques with libraries like Formik. Each section builds upon the previous one, providing you with a solid foundation for creating robust, user-friendly forms in your React applications.