## Step 1: Set Up the Project

1. **Create a new React project:** we will be setting up the project, creating reusable components, managing form state, handling validation, and form submission. Open your terminal and run the following command to create a new React application called `contact-form-app`:

   ```
   npx create-react-app contact-form-app

   cd contact-form-app
   ```

2. **Install necessary dependencies:** We will use Material-UI (MUI) for styling the form components. Run the following command to install MUI:

   ```
   npm install @mui/material @emotion/react @emotion/
   styled
   ```

## Step 2: Create Reusable Components

1. **Create a folder for components:** Inside the src directory, create a new folder called components to store our reusable components:

   ```
   mkdir src/components
   ```

**Create a TextInput component:** The TextInput component will be a reusable input field with built-in error handling.

```
// src/components/TextInput.js

import React from 'react';
import TextField from '@mui/material/TextField';

const TextInput = ({ label, name, value, onChange, onBlur,
error, helperText }) => {
  return (
    <TextField
      fullWidth
      variant="outlined"
```

```
      label={label}
      name={name}
      value={value}
      onChange={onChange}
      onBlur={onBlur}
      error={!!error}
      helperText={helperText}
    />
  );
};

export default TextInput;
```

- **Props Explanation:**
  - `label`: The label displayed above the input field.
  - `name`: The name attribute for the input field, used to identify the field.
  - `value`: The current value of the input field.
  - `onChange`: A function to handle changes to the input field.
  - `onBlur`: A function to handle the blur event (when the input field loses focus).
  - `error`: A boolean indicating if the input field has an error.
  - `helperText`: Text to display as a helper or error message.

**Create a SubmitButton component:** The SubmitButton component will be a reusable button for submitting the form.

```
// src/components/SubmitButton.js

import React from 'react';
import Button from '@mui/material/Button';

const SubmitButton = ({ children, ...props }) => {
  return (
    <Button variant="contained" color="primary"
type="submit" {...props}>
      {children}
    </Button>
  );
};

export default SubmitButton;
```

- **Props Explanation:**
  - `children`: The content to be displayed inside the button.
  - `...props`: Any additional props to be passed to the button component.

## Step 3: Create the Contact Form

**Create the ContactForm component:** This component will manage the form state, handle validation, and handle form submission.

```
// src/components/ContactForm.js

import React, { useState } from 'react';
import TextInput from './TextInput';
import SubmitButton from './SubmitButton';
import { Box } from '@mui/material';

const ContactForm = () => {
  // State to manage form values
  const [values, setValues] = useState({
    name: '',
    email: '',
    message: ''
  });

  // State to manage form errors
  const [errors, setErrors] = useState({});

  // Handle input value changes
  const handleChange = (e) => {
    const { name, value } = e.target;
    setValues({
      ...values,
      [name]: value
    });
  };

  // Validate form values
  const validate = () => {
    let tempErrors = {};
    if (!values.name) tempErrors.name = 'Name is required';
    if (!values.email) tempErrors.email = 'Email is
required';
    else if (!/\S+@\S+\.\S+/.test(values.email))
tempErrors.email = 'Email is not valid';
    if (!values.message) tempErrors.message = 'Message is
required';
    setErrors(tempErrors);
    return Object.keys(tempErrors).length === 0;
  };

  // Handle form submission
  const handleSubmit = (e) => {
    e.preventDefault();
    if (validate()) {
      // Simulate an API call
      setTimeout(() => {
        alert(JSON.stringify(values, null, 2));
        setValues({ name: '', email: '', message: '' });
      }, 500);
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <Box mb={2}>
        <TextInput
          name="name"
          label="Name"
          value={values.name}
          onChange={handleChange}
          error={!!errors.name}
          helperText={errors.name}
        />
      </Box>
      <Box mb={2}>
        <TextInput
          name="email"
          label="Email"
          value={values.email}
          onChange={handleChange}
          error={!!errors.email}
          helperText={errors.email}
        />
      </Box>
      <Box mb={2}>
        <TextInput
          name="message"
          label="Message"
          multiline
          rows={4}
          value={values.message}
          onChange={handleChange}
          error={!!errors.message}
          helperText={errors.message}
        />
      </Box>
      <SubmitButton>Submit</SubmitButton>
    </form>
  );
};

export default ContactForm;
```

- **State Management:**
  - `values`: An object to store the current values of the form fields.
  - `setValues`: A function to update the form values.
  - `errors`: An object to store error messages for the form fields.
  - `setErrors`: A function to update the error messages.
- **Event Handlers:**
  - handleChange: Updates the form values when an input field changes.
  - `validate`: Checks the form values for errors and updates the errors state.
  - `handleSubmit`: Handles the form submission, validates the form, and if valid, simulates an API call and resets the form.

## Step 4: Integrate the Form into Your App

**Update the App.js file to include the ContactForm component:**

```
// src/App.js

import React from 'react';
import Container from '@mui/material/Container';
import ContactForm from './components/ContactForm';

const App = () => {
  return (
    <Container maxWidth="sm">
      <h1>Contact Us</h1>
      <ContactForm />
    </Container>
  );
};

export default App;
```

- **Explanation:**
  - The App component uses the Container component from Material-UI to center the content.
  - The ContactForm component is included inside the Container.

## Step 5: Run the Application

1. **Start the development server:** Open your terminal and run the following command to start the development server:

   ```
   npm start
   ```

2. **Open your browser and navigate to http://localhost:3000:** You should see your contact form with the fields for name, email, and message. The form will handle input changes, validation, and submission.

## Detailed Explanation of the Contact Form

### TextInput Component

- **Purpose**: A reusable component for rendering text input fields with error handling.
- **Usage**: Used to render individual input fields in the form.

- **Props**:
    - `label`: The label for the input field.
    - `name`: The name of the input field.
    - `value`: The current value of the input field.
    - `onChange`: The function to call when the input value changes.
    - `onBlur`: The function to call when the input loses focus.
    - `error`: Boolean indicating if the input has an error.
    - `helperText`: Text to display as a helper or error message.

**SubmitButton Component**

- **Purpose**: A reusable component for rendering a submit button.
- **Usage**: Used to render the submit button for the form.
- **Props**:
    - `children`: The content to be displayed inside the button.
    - `...props`: Additional props to be passed to the button component.

**ContactForm Component**

- **Purpose**: Manages the state, validation, and submission of the contact form.
- **State**:
    - `values`: An object to store the current values of the form fields.
    - `errors`: An object to store error messages for the form fields.
- **Event Handlers**:
    - `handleChange`: Updates the form values when an input field changes.
    - `validate`: Checks the form values for errors and updates the errors state.
    - `handleSubmit`: Handles the form submission, validates the form, and if valid, simulates an API call and resets the form.