Implementing Form Validations in React.js

In this lab we will implement the methodology for validating the React.js form. We will implement the form validations for Email and Password.

Task 1: Create a component of name FormValidation, and add the following code in it.

Define the state object in the constructor

```
constructor (props) {
   super(props);
   this.state = {
    email: '',
    password: '',
    formErrors: {email: '', password: ''},
    emailValid: false,
    passwordValid: false,
    formValid: false
   }
}
```

The above code contains state properties like email, password, emailValid, passwordValid, formValid and the formError object.

Task 2: Add the following methods for defining validation rules

```
validateField(fieldName, value) {
   let fieldValidationErrors = this.state.formErrors;
   let emailValid = this.state.emailValid;
   let passwordValid = this.state.passwordValid;

   switch(fieldName) {
    case 'email':
     emailValid = value.match(/^([\w.%+-]+)@([\w-]+\.)+([\w]{2,})$/i);
     fieldValidationErrors.email = emailValid ? '' : ' is invalid';
     break;
    case 'password':
     passwordValid = value.length >= 6;
     fieldValidationErrors.password = passwordValid ? '': ' is too short';
     break;
    default:
     break;
   }
   this.setState({formErrors: fieldValidationErrors,
                  emailValid: emailValid,
                  passwordValid: passwordValid
                }, this.validateForm);
  }

  validateForm() {
   this.setState({formValid: this.state.emailValid && this.state.passwordValid});
  }
```

The **validateField** method defines validation rules for Email and Password. If the field is invalid, them **formValidationErrors** will store validation error messages. Finally the **formErrors** object will contain the state of the form. The **validateForm** method set the state for the form validation.

Task 3: Add the following method in the component class for handling input changes for all textboxes

```
handleUserInput(e) {
  const name = e.target.name;
  const value = e.target.value;
  this.setState({[name]: value},
            () => { this.validateField(name, value) });
}
```

Task 4: Add the following method to set the error class

```
errorClass(error) {
  return(error.length === 0 ? '' : 'has-error');
}
```

Task 5: Add the following markup for rendering the form

```
render () {
  return (
    <form className="demoForm">
      <h2>Sign up</h2>
      <div className="panel panel-default">
        <FormErrors formErrors={this.state.formErrors} />
      </div>
      <div className={`form-group ${this.errorClass(this.state.formErrors.email)}`}>
        <label htmlFor="email">Email address</label>
        <input type="email" required className="form-control" name="email"
          placeholder="Email"
          value={this.state.email}
          onChange={this.handleUserInput.bind(this)}  />
      </div>
      <div className={`form-group ${this.errorClass(this.state.formErrors.password)}`}>
        <label htmlFor="password">Password</label>
        <input type="password" className="form-control" name="password"
          placeholder="Password"
          value={this.state.password}
          onChange={this.handleUserInput.bind(this)}  />
      </div>
      <button type="submit" className="btn btn-primary" disabled={!this.state.formValid}>Sign
up</button>
    </form>
  )
}
```

The above markup will render the form with all validations. Note that, the markup uses the **FormErrors** custom component. This component will be used to show all error messages based on **formErrors** state passed to it.

Task 6: Add a new component for showing FormError as shown in the following code

```
import React from 'react';
// Validation Summary
export const FormErrors = ({formErrors}) =>
  <div className='formErrors'>
    {Object.keys(formErrors).map((fieldName, i) => {
      if(formErrors[fieldName].length > 0){
        return (
          <p key={i}>{fieldName} {formErrors[fieldName]}</p>
        )
      } else {
        return '';
      }
    })}
  </div>
```

The above code shows the validation summary based on error messages.

Run the application