Validations in React Form

- Validation is the process of verifying user input.
- Validation is required to ensure that contractionary and unauthorized data is not get stored into database.
- Validation can be handled manually or by using pre-defined library functions and properties.
- Formik supports synchronous and asynchronous validations.
- Formik handles validation at 2 levels
    - Formstate validation / Form-level validation
    - Inputstate validation / Field-level validation
- "Yup" is used for Schema based validation.
- Formik uses various methods/events for validating the values
    - After Change events/methods
        - handleChange
        - setFieldValue
        - setValues
    - After Blur events/methods
        - handleBlur
        - setTouched
        - setFieldTouched
    - After Submit events/methods
        - handleSubmit
        - submitForm

FormikValidation.js

```
import React from 'react';

import { useFormik } from 'formik';



const ValidateProduct = productData => {

   const errors = {};

    if(!productData.Name) {

       errors.Name = 'Product Name Required';

    } else if(productData.Name.length<4) {

        errors.Name = 'Name too short.. Min 4 Chars Required';

    }


    if(!productData.Price) {

        errors.Price = 'Product Price Required';

    } else if(isNaN(productData.Price)) {

        errors.Price = 'Price must be a Number';

    }


    if(!productData.Code) {

        errors.Code = 'Product Code Required';
    } else if (!/[A-Z]{3}[0-9]{2}/.test(productData.Code)) {
       errors.Code = 'Invalid Code';
    }

   return errors;
}



export default function FormikValidation(){

   const formik = useFormik({
      initialValues: {
         Name: '',
         Price: 0,
         Code: ''
      },
      validate: ValidateProduct,
      onSubmit : values => {
         alert(JSON.stringify(values));
      }
   })
```

```
    return(
        <div className="container-fluid">
          <form onSubmit={formik.handleSubmit}>
            <h3>Register Product</h3>
                <dl>
                    <dt>Name</dt>
                    <dd><input type="text"
onBlur={formik.handleBlur}
onChange={formik.handleChange}
value={formik.values.Name} name="Name" /></dd>
                    <dd className="text-danger">
                        {(formik.touched.Name &&
(formik.errors.Name)?formik.errors.Name:null)}
                    </dd>
                    <dt>Price</dt>
                    <dd><input type="number"
onBlur={formik.handleBlur}
onChange={formik.handleChange}
value={formik.values.Price} name="Price" /></dd>
                    <dd className="text-danger">
                        {(formik.touched.Price &&
(formik.errors.Price)?formik.errors.Price:null)}
                    </dd>
                    <dt>Product Code</dt>
                <dd><input type="text"
onBlur={formik.handleBlur}
onChange={formik.handleChange}
value={formik.values.Code} name="Code" /></dd>
                <dd className="text-danger">
                  {(formik.touched.Code &&
(formik.errors.Code)?formik.errors.Code:null)}
                </dd>
              </dl>
              <button>Register</button>
         </form>
        </div>
  )
}
```

Yup Library

- It provides object schema validation.
- It provides a validationSchema object.
- Schema object comprises of key and value reference.
- It uses an error object that can bind with any HTML element and configure the validations for element.

- It comprises of Key and Value reference, where key refers to validation error and value refers to validation message.

Syntax:

```
 validationSchema: yup.object({
          Name:
yup.DataType().required().max().email()
 })
```

- Yup library uses “formik.getFieldProps()” that can give access to values of elements.

Syntax:

```
 <input type=”text” name=”Name”
{…formik.getFieldProps(“Name”)} />
```

```
> npm install yup --save
```

YupValidation.js

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

```
export default function YupValidation(){
   const formik = useFormik({
      initialValues: {
         Name: '',
         Salary:0,
         Email: ''
      },
      validationSchema: yup.object({
         Name: yup.string()
                .min(4, "Name too short min 4 chars")
                .max(10, "Name too long max 10 chars only")
                .required("User Name Required"),
         Salary: yup.number()
                .required("Salary Required"),
         Email: yup.string()
                 .required("Email Required")
                 .email("Invalid Email")
      }),
      onSubmit: values => {
         alert(JSON.stringify(values));
      }
   })
```

```
    return(
        <div className="container-fluid">
            <h2>Register User</h2>
            <form  onSubmit={formik.handleSubmit}>
                <dl>
                    <dt>Name</dt>
                    <dd><input type="text" name="Name"
{...formik.getFieldProps("Name")} /></dd>
                    <dd className="text-danger">
                        {(formik.touched.Name &&
formik.errors.Name?formik.errors.Name:null)}
                    </dd>
                    <dt>Salary</dt>
                    <dd><input type="text" name="Salary"
{...formik.getFieldProps("Salary")} /></dd>
                    <dd className="text-danger">
                        {(formik.touched.Salary &&
formik.errors.Salary?formik.errors.Salary:null)}
                    </dd>
                    <dt>Email</dt>
                    <dd><input type="text" name="Email"
{...formik.getFieldProps("Email")} /></dd>
                    <dd className="text-danger">
                    {(formik.touched.Email &&
formik.errors.Email?formik.errors.Email:null)}
                </dd>
            </dl>
            <button>Register</button>
        </form>
    </div>
  )
}
```

Formik Components for Validation

- Formik provides pre-defined library of components.
- Component have pre-defined behaviour and functionality, which you can inject and use in your application.
- Formik components can bind with data fields, identify the errors and report errors.
- It keeps your UI cleaner.
- It reduces the logic and binding you have to configure manually.
- The components provided by Formik
    - <Formik>
    - <Form>
    - <Field>
    - <ErrorMessage>

Syntax:

```
<formik  initialValues = {
    { Name: ‘ ‘, Salary: ‘ ‘, Email: ‘ ‘ }
 },
 validationSchema = {
 },
onsubmit = {
 }
 { props => (
      <Form>
            <Field name=”Name” type=”text”>
</Field>
            <ErrorMessage name=”Name”>
</ErrorMessage>
      </Form>
  )
 }
```

Syntax:

```
<Formik initialValues={} validationSchema={} onSubmit={}>
   {
      props=> ()
    }
 </Formik>
```

Props function in Formik is responsible for identifying the state of every field.

dirty	: It returns true when any field in form is modified.
touched	**:** It returns true when any field is touched but value not modified.
isValid	**:** It returns true when all form fields are valid.

Ex: **ValidationComponent.js**

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

export default function ValidationComponent(){
   return(
      <Formik  initialValues={
         {
            Name: '',
            Salary: '',
            Email: '',
            City: ''
         }
      }
      validationSchema={
         yup.object({
            Name: yup.string().required("Name
Required").min(4,"Name too Short").max(10,"Name too
Long"),
            Salary: yup.number().required("Salary Required"),
            Email: yup.string().required("Email
Required").email("Invalid Email"),
         })
      }
      onSubmit= {
         values => {
            alert(JSON.stringify(values));
         }
      }
      >
       {
         props => (
                <div className="container-fluid">
                    <h3>Register User</h3>
                    <Form>
                        <dl>
                            <dt>Name</dt>
                            <dd> <Field type="text"
name="Name"></Field> </dd>
                            <dd className="text-danger">
<ErrorMessage name="Name"></ErrorMessage> </dd>
                            <dt>Salary</dt>
                            <dd> <Field type="text"
name="Salary"></Field> </dd>
                            <dd className="text-danger">
<ErrorMessage name="Salary"></ErrorMessage> </dd>
                            <dt>Email</dt>
                            <dd> <Field type="text"
name="Email"></Field> </dd>
                            <dd className="text-danger">
<ErrorMessage name="Email"></ErrorMessage> </dd>
                            <dt>City</dt>
                            <dd>
                                <Field as="select" name="City">
                                    <option>Delhi</option>
```

```
                              <option>Hyd</option>
                          </Field>
                      </dd>
                  </dl>
                  <button className="btn btn-primary"
disabled={props.isValid==false}>Register</button>
                  <button className="btn btn-success"
>Save</button>
              </Form>
          </div>
      )
    }
    </Formik>
  )
}
```