Form Handling

- Forms are very common in any web application development.

- Unlike angular and angularJS, that gives form validation out of the box, we have to handle forms ourself in React.

- That includes complications like
  1. How to get form values.
  2. How to manage the form state.
  3. How to validate the form on the fly.
  4. How to show validation messages.

- In React, forms can be created in 2 ways.
  1. Controlled Component
  2. Uncontrolled Component

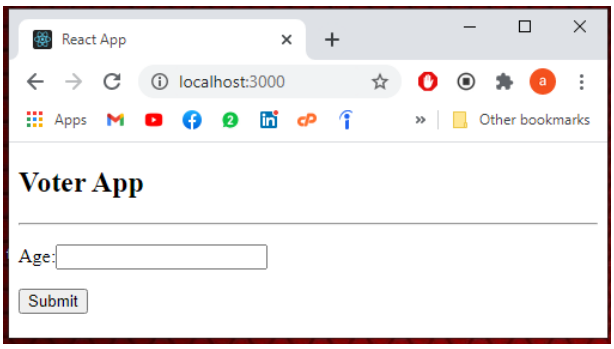
Working with Controlled Component Approach:

- In HTML, form elements such as **<input>, <textarea>, and <select>** typically maintain their own state and update it based on user input.

- In React, state of these input elements is typically kept in the **state** property of components and only updated with **setState().**

- An input form element whose value is controlled by React in this way is called a "controlled component".

USE CASE:



Example:

```
import React from "react";
import ReactDOM from "react-dom";

class VoterComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      age: "",
    };
    this.changeAgeHandler = this.changeAgeHandler.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  changeAgeHandler(event) {
    this.setState({ age: event.target.value });
  }

  handleSubmit(event) {
    event.preventDefault();
    if (this.state.age >= 18) {
      alert(`U are eligible to vote...`);
    } else {
      alert(`U are not eligible to vote... `);
    }
  }

  render() {
    return (
      <div>
        <form onSubmit={this.handleSubmit}>
          <h2>Voter App</h2>
          <hr />
          <p>
            Age:
            <input
              type="number"
              name="age"
              value={this.state.age}
              onChange={this.changeAgeHandler}
            />
          </p>
          <input type="submit" value="Submit" />
        </form>
      </div>
    );
  }
}
const element = <VoterComponent />;
ReactDOM.render(element, document.getElementById("root"));
```

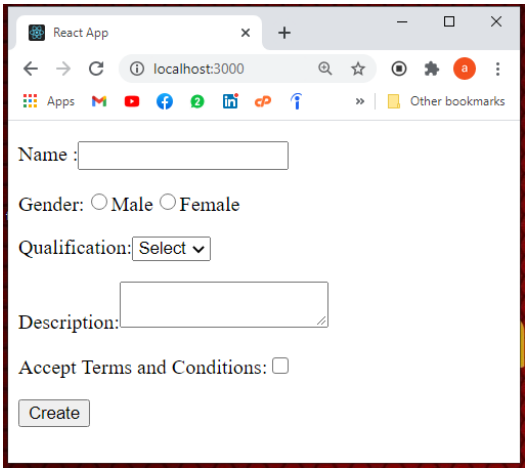
Working with Uncontrolled Component:

- In most cases, we recommend using controlled components to implement forms.
- In a controlled component, form data is handled by a React component.
- The alternative is uncontrolled components, where form data is handled by the DOM itself.
- To write an uncontrolled component, instead of writing an event handler for every state update, you can use a **ref** to get form values from the DOM.

Example:

```
import React from "react";
import ReactDOM from "react-dom";

class VoterComponent extends React.Component {
  constructor(props) {
    super(props);
    this.age = React.createRef();
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleSubmit(event) {
    event.preventDefault();
    if (this.age.current.value >= 18) {
      alert(`U are eligible to vote...`);
    } else {
      alert(`U are not eligible to vote... `);
    }
  }

  render() {
    return (
      <div>
        <form onSubmit={this.handleSubmit}>
          <h2>Voter App</h2>
          <hr />
          <p>
            Age:
            <input type="number" ref={this.age} />
          </p>
          <input type="submit" value="Submit" />
        </form>
      </div>
    );
  }
}
const element = <VoterComponent />;
ReactDOM.render(element, document.getElementById("root"));
```

Handling Form Components:

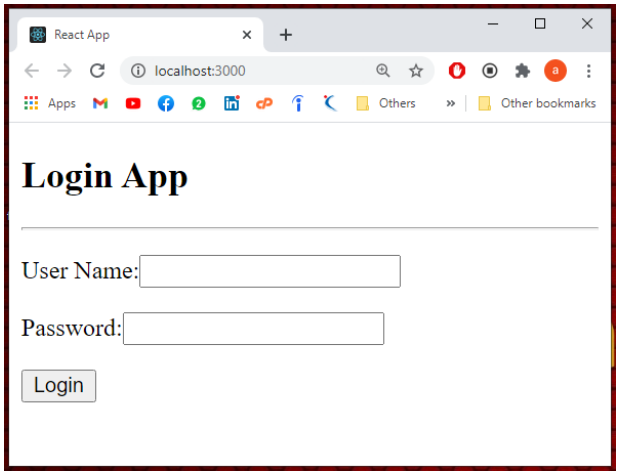



```
import React from "react";
import ReactDOM from "react-dom";

class MyFormComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: "",
      qual: "",
      gender: "",
      desc: "",
      tnc: false,
    };
    this.changeNameHandler = this.changeNameHandler.bind(this);
    this.changeGenderHandler = this.changeGenderHandler.bind(this);
    this.changeQualHandler = this.changeQualHandler.bind(this);
    this.changeDescHandler = this.changeDescHandler.bind(this);
    this.changeTncHandler = this.changeTncHandler.bind(this);
    this.onCreateEmployee = this.onCreateEmployee.bind(this);
  }

  changeNameHandler(event) {
    this.setState({ name: event.target.value });
  }

  changeQualHandler(event) {
    this.setState({ qual: event.target.value });
  }
  changeGenderHandler(event) {
    this.setState({ gender: event.target.value });
  }

  changeDescHandler(event) {
    this.setState({ desc: event.target.value });
  }
  changeTncHandler(event) {
    const value =
```

```
      event.target.type === "checkbox"
        ? event.target.checked
        : event.target.value;

    this.setState({ tnc: value });
  }

  onCreateEmployee(event) {
    event.preventDefault();
    let result = `${this.state.name} ${this.state.gender} ${this.state.qual} ${this.state.
desc} ${this.state.tnc}`;
    alert(result);
  }

  render() {
    return (
      <div>
        <form>
          <p>
            Name :
            <input
              type="text"
              name="name"
              value={this.state.name}
              onChange={this.changeNameHandler}
            />
          </p>
          <p>
            Gender:
            <input
              type="radio"
              name="gender"
              value="male"
              onChange={this.changeGenderHandler}
            />
            Male
            <input
              type="radio"
              name="gender"
              value="female"
              onChange={this.changeGenderHandler}
            />
            Female
          </p>
          <p>
            Qualification:
            <select value={this.state.qual} onChange={this.changeQualHandler}>
              <option value="">Select</option>
              <option value="ug">UG</option>
              <option value="pg">PG</option>
            </select>
          </p>
```

```
          <p>
            Description:
            <textarea
              value={this.state.desc}
              onChange={this.changeDescHandler}
            />
          </p>
          <p>
            Accept Terms and Conditions:
            <input
              type="checkbox"
              name="tnc"
              checked={this.state.tnc}
              onChange={this.changeTncHandler}
            />
          </p>
        </form>
        <button onClick={this.onCreateEmployee}>Create</button>
      </div>
    );
  }
}
const element = <MyFormComponent></MyFormComponent>;
ReactDOM.render(element, document.getElementById("root"));
```

Form Validations




```
import React from "react";
import ReactDOM from "react-dom";

class LoginComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      uname: "",
      password: "",
      formErrors: {},
    };
    this.changeUserNameHandler = this.changeUserNameHandler.bind(this);
    this.changePasswordHandler = this.changePasswordHandler.bind(this);

    this.handleSubmit = this.handleSubmit.bind(this);
  }

  changeUserNameHandler(event) {
    this.setState({ uname: event.target.value });
  }

  changePasswordHandler(event) {
    this.setState({ password: event.target.value });
  }

  handleFormValidation() {
    const { uname, password } = this.state;

    let formIsValid = true;

    if (!uname) {
      formIsValid = false;
      alert("User Name is required.");
    }

    if (!password) {
      formIsValid = false;
```

```
      alert("Password is required.");
    }
    return formIsValid;
  }

  handleSubmit(event) {
    const { uname, password } = this.state;

    event.preventDefault();
    if (this.handleFormValidation()) {
      if (uname === "adi" && password === "adi123") {
        alert("Welcome..." + uname);
      } else {
        alert("Login Denied");
      }
    }
  }

  render() {
    return (
      <div>
        <form onSubmit={this.handleSubmit}>
          <h2>Login App</h2>
          <hr />
          <p>
            User Name:
            <input
              type="text"
              name="uname"
              value={this.state.uname}
              onChange={this.changeUserNameHandler}
            />
          </p>
          <p>
            Password:
            <input
              type="password"
              name="upwd"
              value={this.state.password}
              onChange={this.changePasswordHandler}
            />
          </p>
          <input type="submit" value="Login" />
        </form>
      </div>
    );
  }
}
const element = <LoginComponent />;
ReactDOM.render(element, document.getElementById("root"));
```

Working with Formik

- Formik is one of the popular library available and let's use this library for building powerful forms in our react application.

- Formik is a small group of React components and hooks for building forms in React and React Native.

- It helps with the three most parts:
  1. Getting values in and out of form state
  2. Validation and error messages
  3. Handling form submission

Installation of Formik:

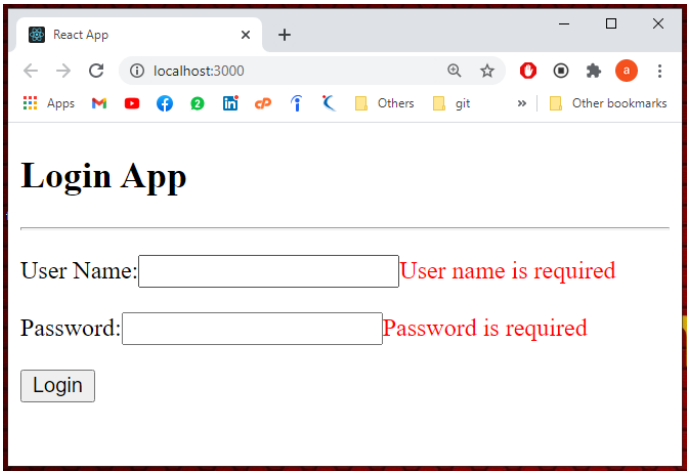
npm install formik - -save

Example:

```
import React from "react";
import ReactDOM from "react-dom";
import { useFormik } from "formik";

const LoginComponent = () => {
  const formik = useFormik({
    initialValues: {
      uname: "",
      password: "",
    },
    onSubmit: (values) => {
      if (values.uname === "adi" && values.password === "adi123") {
        alert("Welcome..." + values.uname);
      } else {
        alert("Login Denied");
      }
    },
  });

  return (
    <div>
      <form onSubmit={formik.handleSubmit}>
        <h2>Login App</h2>
        <hr />
        <p>
          User Name:
          <input
            type="text"
            name="uname"
            value={formik.values.uname}
            onChange={formik.handleChange}
```

```
          />
        </p>
        <p>
          Password:
          <input
            type="password"
            name="password"
            value={formik.values.password}
            onChange={formik.handleChange}
          />
        </p>

        <input type="submit" value="Login" />
      </form>
    </div>
  );
};

const element = <LoginComponent />;
ReactDOM.render(element, document.getElementById("root"));
```

Example: Validations using formik




```
import React from "react";
import ReactDOM from "react-dom";
import { useFormik } from "formik";

const validateLoginForm = (data) => {
  const errors = {};

  if (!data.uname) {
    errors.uname = "User name is required";
  }

  if (!data.password) {
    errors.password = "Password is required";
  }

  return errors;
};

const LoginComponent = () => {
  const formik = useFormik({
    initialValues: {
      uname: "",
      password: "",
    },
    validate: validateLoginForm,
    onSubmit: (values) => {
      if (values.uname === "adi" && values.password === "adi123") {
        alert("Welcome..." + values.uname);
      } else {
        alert("Login Denied");
      }
    },
  });

  return (
    <div>
      <form onSubmit={formik.handleSubmit}>
```

```
        <h2>Login App</h2>
        <hr />
        <p>
          User Name:
          <input
            type="text"
            name="uname"
            value={formik.values.uname}
            onChange={formik.handleChange}
            onBlur={formik.handleBlur}
          />
          {formik.touched.uname && formik.errors.uname ? (
            <span style={{ color: "red" }}>{formik.errors.uname}</span>
          ) : null}
        </p>
        <p>
          Password:
          <input
            type="password"
            name="password"
            value={formik.values.password}
            onChange={formik.handleChange}
            onBlur={formik.handleBlur}
          />
          {formik.touched.password && formik.errors.password ? (
            <span style={{ color: "red" }}>{formik.errors.password}</span>
          ) : null}
        </p>

        <input type="submit" value="Login" />
      </form>
    </div>
  );
};

const element = <LoginComponent />;
ReactDOM.render(element, document.getElementById("root"));
```

Note:
Limitation of the above program is to set values for different attributes like **value**, **onchange**, **onblur** for input elements, we have written so much of code.

```
User Name:
<input
  type="text"
  name="uname"
  value={formik.values.uname}
  onChange={formik.handleChange}
  onBlur={formik.handleBlur}
/>
```

However, to save time, **useFormik()** returns a helper method called **formik.getFieldProps("properyName).**

```
User Name:
<input
   type="text"
   name="uname"
   {...formik.getFieldProps("uname")}
/>
```

Example:

```
import React from "react";
import ReactDOM from "react-dom";
import { useFormik } from "formik";

const validateLoginForm = (data) => {
  const errors = {};

  if (!data.uname) {
    errors.uname = "User name is required";
  }

  if (!data.password) {
    errors.password = "Password is required";
  }

  return errors;
};

const LoginComponent = () => {
  const formik = useFormik({
    initialValues: {
      uname: "",
      password: "",
    },
    validate: validateLoginForm,
    onSubmit: (values) => {
      if (values.uname === "adi" && values.password === "adi123") {
        alert("Welcome..." + values.uname);
      } else {
        alert("Login Denied");
      }
    },
  });

  return (
    <div>
      <form onSubmit={formik.handleSubmit}>
        <h2>Login App</h2>
        <hr />
        <p>
```

```
          User Name:
          <input type="text" name="uname" {...formik.getFieldProps("uname")} />
          {formik.touched.uname && formik.errors.uname ? (
            <span style={{ color: "red" }}>{formik.errors.uname}</span>
          ) : null}
        </p>
        <p>
          Password:
          <input
            type="password"
            name="password"
            {...formik.getFieldProps("password")}
          />
          {formik.touched.password && formik.errors.password ? (
            <span style={{ color: "red" }}>{formik.errors.password}</span>
          ) : null}
        </p>

        <input type="submit" value="Login" />
      </form>
    </div>
  );
};

const element = <LoginComponent />;
ReactDOM.render(element, document.getElementById("root"));
```

Using Yup Library for validations

- But we are free to use any third party validation library available and do the form validation.
- Formik's authors/a large portion of its users use Yup library for object schema validation.
- Since Formik authors/users love Yup so much, Formik has a special configuration option for Yup called **validationSchema** which will automatically transform Yup's validation errors messages into a pretty object whose keys match our forms input values.

Installation of Yup:
D:\ReactJS\hello-app>npm install yup --save

Example:

```
import React from "react";
import ReactDOM from "react-dom";
import { useFormik } from "formik";
import * as yup from "yup";

const LoginComponent = () => {
  const formik = useFormik({
    initialValues: {
      uname: "",
      password: "",
    },
    validationSchema: yup.object({
      uname: yup
        .string()
        .matches("^[a-z]*$", "Only characters are allowed")
        .max(10, "User name should not exceed 10 Characters")
        .required("User Name is required"),

      password: yup
        .string()
        .matches("^[a-z0-9]*$", "Invalid Password")
        .max(10, "Password should not exceed 10 Characters")
        .required("Password is required"),
    }),
    onSubmit: (values) => {
      if (values.uname === "adi" && values.password === "adi123") {
        alert("Welcome..." + values.uname);
      } else {
        alert("Login Denied");
      }
    },
  });

  return (
    <div>
      <form onSubmit={formik.handleSubmit}>
```

```
        <h2>Login App</h2>
        <hr />
        <p>
          User Name:
          <input type="text" name="uname" {...formik.getFieldProps("uname")} />
          {formik.touched.uname && formik.errors.uname ? (
            <span style={{ color: "red" }}>{formik.errors.uname}</span>
          ) : null}
        </p>
        <p>
          Password:
          <input
            type="password"
            name="password"
            {...formik.getFieldProps("password")}
          />
          {formik.touched.password && formik.errors.password ? (
            <span style={{ color: "red" }}>{formik.errors.password}</span>
          ) : null}
        </p>

        <input type="submit" value="Login" />
      </form>
    </div>
  );
};

const element = <LoginComponent />;
ReactDOM.render(element, document.getElementById("root"));
```

Note:
Still in the above program, the error messages code is repeated for each and every property.
We can over this , limitation by using fallowing built-in elements of formik.

<Formik />, <Form />, <Field />, and <ErrorMessage />. <Formik/>

```
import React from "react";
import ReactDOM from "react-dom";
import "./index.css";
import { Formik, Field, Form, ErrorMessage } from "formik";
import * as yup from "yup";

const LoginComponent = () => {
  return (
    <Formik
      initialValues={{
        uname: "",
        password: "",
      }}
      validationSchema={yup.object({
        uname: yup
          .string()
          .matches("^[a-z]*$", "Only characters are allowed")
          .max(15, "User name should not be greater than 15 character")
          .required("User name is required!"),
        password: yup
          .string()
          .matches("^[a-z0-9]*$", "Invalid Password")
          .required("Password is required!"),
      })}
      onSubmit={(values) => {
        if (values.uname === "adi" && values.password === "adi123") {
          alert("Welcome..." + values.uname);
        } else {
          alert("Login Denied");
        }
      }}
    >
      <div>
        <h2>Login App</h2>
        <hr />
        <Form>
          <p>
            <label>User Name</label>
            <Field name="uname" type="text"></Field>
            <span style={{ color: "red" }}>
              <ErrorMessage name="uname"></ErrorMessage>
            </span>
          </p>
          <p>
            <label>Password </label>
            <Field name="password" type="password"></Field>
            <span style={{ color: "red" }}>
              <ErrorMessage name="password"></ErrorMessage>
            </span>
          </p>

          <button type="submit">Login</button>
```

```
        </Form>
      </div>
    </Formik>
  );
};

const element = <LoginComponent></LoginComponent>;
ReactDOM.render(element, document.getElementById("root"));
```