Hacé click acá para dejar tu feedback sobre esta clase.

Hacé click acá completar el quiz teórico de esta lecture.

React Forms

Los formularios son muy útiles en cualquier aplicacion WEB. En React tenemos que manejar estos formularios nosotros mismos. Por ejemplo: obtener los valores que se ingresan, cómo administramos el state del form, las validaciones de cada valor ingresado, mostrar los mensajes de validacion, etc. Existen diferentes métodos y librerias que nos ayudan con esto, pero como no queremos depender de codigo de otro, lo haremos nosotros.

Tipos de Componentes para un Formulario

En React tenemos dos tipos de componentes para crear nuestro Form: `Controlled Components` y `Uncontrolled Components`.

Componentes Controlados

Como sabemos, el estado en React es mutable, y lo mantenemos dentro del componente. En un componente controlado, que renderiza el Formulario, tambien controla lo que sucede con el. Es decir que a medida que cambien los valores del Form, el componente guarda esos valores en su state. Aqui vemos un pequeño ejemplo:

Con Hooks:

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

function Form() {
  const [name, setName] = useState("");

  function handleChange(e) {
    setName(e.target.value);
  }
```

```
  return (
    <form>
      <input type="text" name="name" value={name} onChange={handleChange} />
    </form>
  );
}
```

Con Class:

```
import React from "react";

class Form extends React.Component {
  constructor() {
    super();
    this.state = {
      name: "",
    };
    this.handleChange = this.handleChange.bind(this);
  }

  handleChange(e) {
    this.setState({
      name: e.target.value,
    });
  }

  render() {
    return (
      <form>
        <input
          type="text"
          name="name"
          value={this.state.name}
          onChange={this.handleChange}
        />
      </form>
    );
  }
}

export default Form;
```

Nuestro objetivo es que cada vez que cambien los valores de nuestro Form lo vayamos almacenando en nuestro state. Esto significa que nuestro Form puede responder a los cambios que se hagan en cada input. Por ejemplo agregar validaciones, deshabilitar un boton hasta que todos los campos esten llenos o validados, forzar un formato especifico en cada input.

Manejando multiples inputs:

En la mayoria de los casos tendremos mas de un solo input. Para manejarlos podemos agregar el atributo `name` a cada uno y dejar que nuestra funcion que maneje los cambios decida que hacer dependiendo de cada valor de `e.target.name`:

Con Hooks:

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

function Form() {
  const [input, setInput] = useState({
    name: "",
    lastname: "",
  });

  function handleChange(e) {
    const value = e.target.value;
    const name = e.target.name;

    setInput({
      [name]: value, // Sintaxis ES6 para actualizar la key correspondiente
    });
  }

  return (
    <form>
      <input name="name" type="text" value={name} onChange={handleChange} />
      <input
        name="lastname"
        type="text"
        value={lastname}
        onChange={handleChange}
      />
    </form>
  );
}

export default Form;
```

Con Class:

```
import React from "react";

class Form extends React.Component {
  constructor() {
    super();
    this.state = {
      name: "",
      lastname: "",
    };
```

```
    this.handleChange = this.handleChange.bind(this);
  }

  handleChange(e) {
    const value = e.target.value;
    const name = e.target.name;

    this.setState({
      [name]: value, // Sintaxis ES6 para actualizar la key correspondiente
    });
  }

  render() {
    return (
      <form>
        <input
          name="name"
          type="name"
          value={this.state.name}
          onChange={this.handleChange}
        />
        <input
          name="lastname"
          type="name"
          value={this.state.lastname}
          onChange={this.handleChange}
        />
      </form>
    );
  }
}

export default Form;
```

Validando nuestros Inputs

Otra de las cosas que queremos hacer en nuestro componente es validar los input dependiendo de que datos se tengan que ingresar. Por ejempo: en el caso de validar un email, en el momento de cambiar nuestro state queremos 'filtrar' el valor de ese input por una funcion que valide los valores ingresados. En el caso que no sea valido mostraremos un mensaje de error.

Con Hooks:

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

function Form() {
  const [input, setInput] = useState({
    name: "",
    lastname: "",
    user: "",
  });
```

```
  const [error, setError] = useState("");

  function validateEmail(value) {
    var emailPattern = /\S+@\S+\.\S+/; // Expresion Regular para validar Emails.

    if (!emailPattern.test(value)) {
      console.log("entro al if");
      setError("El usuario debe ser un email");
    } else {
      setError("");
    }
  }

  function handleChange(e) {
    const { value, name } = e.target;

    if (name === "user") {
      validateEmail(input.user);
    }

    setInput({
      ...input,
      [name]: value, // Sintaxis ES6 para actualizar la key correspondiente
    });
  }
  return (
    <form>
      <input
        name="name"
        type="text"
        value={input.name}
        onChange={handleChange}
        placeholder="Nombre"
      />
      <input
        name="lastname"
        type="text"
        value={input.lastname}
        onChange={handleChange}
        placeholder="Apellido"
      />
      <input
        name="user"
        type="text"
        value={input.user}
        onChange={handleChange}
        placeholder="Usuario"
      />
      {!error ? null : <div>{error}</div>}
      <input type="submit" value="Submit" />
    </form>
  );
}
```

```
export default Form;
```

Con Class:

```
import React from "react";

class Form extends React.Component {
  constructor() {
    super();
    this.state = {
      name: "",
      lastname: "",
      user: "",
      error: "",
    };

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

  validateEmail(value) {
    var emailPattern = /\S+@\S+\.\S+/; // Expresion Regular para validar Emails.

    if (!emailPattern.test(value)) {
      this.setState({
        error: "El usuario debe ser un email",
      });
    } else {
      this.setState({
        error: "",
      });
    }
  }

  handleChange(e) {
    const { value, name } = e.target;

    if (name === "user") {
      this.validateEmail(this.state.user);
    }

    this.setState({
      [name]: value, // Sintaxis ES6 para actualizar la key correspondiente
    });
  }

  render() {
    return (
      <form>
        <input
          name="name"
          type="text"
```

```
            value={this.state.name}
            onChange={this.handleChange}
            placeholder="Nombre"
          />
          <input
            name="lastname"
            type="text"
            value={this.state.lastname}
            onChange={this.handleChange}
            placeholder="Apellido"
          />
          <input
            name="user"
            type="text"
            value={this.state.user}
            onChange={this.handleChange}
            placeholder="Usuario"
          />
          {!this.state.error ? null : <div>{this.state.error}</div>}
          <input type="submit" value="Submit" />
        </form>
      );
    }
  }

  export default Form;
```

Para tener un Formulario completo. Tenemos que hacer algun tipo de validacion en cada input dependiendo de que es lo que se quiera ingresar. Y para finalizar, podemos agregar una ultima validacion en nuestro boton de submit para saber si pasamos o no las validaciones anteriores. Si tenemos algun error, deshabilitamos el input de submit. Ahora nuestro state `error`, pasara a ser un objecto con cada tipo de error segun nuestro input, y agregaremos `disabled` para saber si nuestro form esta habilidado o no.

```
this.state = {
  name: "",
  lastname: "",
  user: "",
  errors: {
    name: "",
    lastname: "",
    user: "",
  },
  disabled: true,
};
```

Dentro de `handleChange` agregamos un switch statement para validar cada input:

```
  handleChange(e) {
    const { name, value } = e.target;
```

```
    let errors = this.state.errors;

    switch (name) {
      case 'name':
        errors.name = value.length < 5 ? 'Nombre debe tener almenos 5 caracteres'
: '';
        break;
      case 'lastname':
        errors.lastname = value.length < 5 ? 'Apellido debe tener almenos 5
caracteres' : '';
        break;
      case 'user':
        var emailPattern = /\S+@\S+\.\S+/;
        errors.user = emailPattern.test(value) ? '' : 'El usuario debe ser un
email';
        break;
      default:
        break;
    }
    this.setState({
      [name]: value,
      errors
    });
  }
```

Crearemos una funcion que valide que nuestro Formulario no tenga ningun error para habilitar el boton submit:

```
  validarForm(errors) {
    let valid = true;
    Object.values(errors).forEach( (val) => val.length > 0 && (valid = false));
    if(valid) {
      this.setState({
        disabled: false
      })
    } else {
      this.setState({
        disabled: true
      })
    }
  }
```

Y asi tendriamos un Formulario Controlado, en donde estamos haciendo validaciones por cada input, y deshabilitamos el input de submit hasta pasar todas las validaciones.

```
import React from "react";

class Form extends React.Component {
  constructor() {
```

```
    super();
    this.state = {
      name: "",
      lastname: "",
      user: "",
      errors: {
        name: "",
        lastname: "",
        user: "",
      },
      disabled: true,
    };

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

  validarForm(errors) {
    let valid = true;
    Object.values(errors).forEach((val) => val.length > 0 && (valid = false));
    if (valid) {
      this.setState({
        disabled: false,
      });
    } else {
      this.setState({
        disabled: true,
      });
    }
  }

  handleChange(e) {
    const { name, value } = e.target;
    let errors = this.state.errors;

    switch (name) {
      case "name":
        errors.name = value.length < 5 ? "Nombre debe tener 5 caracteres" : "";
        break;
      case "lastname":
        errors.lastname =
          value.length < 5 ? "Apellido debe tener 5 caracteres" : "";
        break;
      case "user":
        var emailPattern = /\S+@\S+\.\S+/; // Expresion Regular para validar
Emails.
        errors.user = emailPattern.test(value)
          ? ""
          : "El usuario debe ser un email";
        break;
      default:
        break;
    }
    this.setState({
      [name]: value, // Sintaxis ES6 para actualizar la key correspondiente
```

```
        errors,
      });

      this.validarForm(this.state.errors);
    }

    render() {
      return (
        <form
          style={{ display: "flex", flexDirection: "column", width: "150px" }}
        >
          <input
            name="name"
            type="name"
            value={this.state.name}
            onChange={this.handleChange}
            placeholder="Nombre"
          />
          {!this.state.errors.name ? null : <div>{this.state.errors.name}</div>}
          <input
            name="lastname"
            type="name"
            value={this.state.lastname}
            onChange={this.handleChange}
            placeholder="Apellido"
          />
          {!this.state.errors.lastname ? null : (
            <div>{this.state.errors.lastname}</div>
          )}
          <input
            name="user"
            type="name"
            value={this.state.user}
            onChange={this.handleChange}
            placeholder="Usuario"
          />
          {!this.state.errors.user ? null : <div>{this.state.errors.user}</div>}
          <input disabled={this.state.disabled} type="submit" value="Submit" />
        </form>
      );
    }
  }

  export default Form;
```

Inputs Dinamicos en Forms Controlados

Otra de las cosas que podemos hacer en un Form controlado es tener Inputs Dinamicos, es decir, dependiendo de el usuario que utilice nuestre formulario, dinamicamente podemos crear inputs que se adapten a cada usuario. Por ejemplo, cuando en un Formulario agregamos miembros en nuestra familia, dinamicamente los inputs se van agregando.

Vamos paso a paso, utilizando Hooks. La idea de este ejemplo que haremos sera crear un simple input para que registre el nombre de una persona, y nos permita agregar familiares. Lo primero que haremos es crear el Form con los datos estaticos, en este caso el nombre de una persona.

```
import React from "react";

function Form() {
  return (
    <form>
      <label htmlFor="nombre">Nombre:</label>
      <input type="text" name="nombre" />
      <input type="submit" value="Submit" />
    </form>
  );
}
export default Form;
```

Para crear los inputs dinamicos utilizaremos un arreglo de objetos. Cada objeto tendra los datos de cada familiar. En este casi tendremos solo la key `nombre`. Agregaremos un boton que nos permite agregar un nuevo familiar, es decir, un nuevo objeto al arreglo. Lo que generara un cambio de state y que el componente se actualice con los cambios. Tendremos que iterar sobre ese arreglo y por cada elemento renderizar un nuevo input, dependiendo la cantidad de keys que tenga en el objeto.

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

function Form() {
  const [familiar, setFamiliar] = useState([{ nombre: "" }]);

  return (
    <form>
      <label htmlFor="nombre">Nombre:</label>
      <input type="text" name="nombre" />
      <input type="button" value="Agrega un Familiar" />
      {familiar.map((el, i) => (
        <div key={`persona-${i}`}>
          <label htmlFor={`nombre-${i}`}>{`Familiar #${i + 1}`}</label>
          <input
            type="text"
            name={`nombre-${i}`}
            id={i}
            data-name="nombre"
            value={e.nombre}
          />
        </div>
      ))}
      <input type="submit" value="Submit" />
    </form>
  );
```

```
}
export default Form;
```

Agregando inputs

Por el momento estamos iterando sobre el state `familiar` para mostrar un input. Todavia no es dinamico. Para eso tenemos que dejar que el usuario agregue inputs haciendo click en el boton que creamos. Vamos a crear un metodo que agregue un nuevo objeto a nuestro state.

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

function Form() {
  const modeloFamiliar = { nombre: "" }; // Creamos un modelo de Familiar para
poder usar este objeto para agregar al state cada vez que agregamos un familiar
  const [familiar, setFamiliar] = useState([{ ...modeloFamiliar }]);

  const agregaFamiliar = () => {
    setFamiliar([...familiar, { ...modeloFamiliar }]); // Hacemos una copia del
state que teniamos y agregamos un objeto nuevo al state.
  };

  return (
    <form>
      <label htmlFor="nombre">Nombre:</label>
      <input type="text" name="nombre" />
      <input
        type="button"
        value="Agrega un Familiar"
        onClick={agregaFamiliar}
      />
      {familiar.map((el, i) => (
        <div key={`persona-${i}`}>
          <label htmlFor={`nombre-${i}`}>{`Familiar #${i + 1}`}</label>
          <input
            type="text"
            name={`nombre-${i}`}
            id={i}
            data-name="nombre"
            value={el.nombre}
          />
        </div>
      ))}
      <input type="submit" value="Submit" />
    </form>
  );
}
export default Form;
```

Controlando los inputs

Ya tenemos nuestros inputs, ahora tenemos que, utilizando lo antes visto. Capturar los valores ingresados utilizando un estado.

```
function Form() {
  const modeloFamiliar = { nombre: "" };
  const [familiar, setFamiliar] = useState([{ ...modeloFamiliar }]);

  const [persona, setPersona] = useState({
    nombre: "",
  });

  const agregaFamiliar = () => {
    setFamiliar([...familiar, { ...modeloFamiliar }]);
  };

  // Este metodo lo usamos para capturar el valor ingresado en nuestro input
estatico
  const handlePersonaChange = (e) =>
    setPersona({
      ...persona,
      [e.target.name]: e.target.value,
    });

  // Este metodo lo usamos para controlar los inputs que se van creando
dinamicamente
  // Primero hacemos una copia del estado `familiar`
  // Utilizamos el `id` y el atributo `data-name` que le asignamos a cada input
para poder reconocerlos entre si.
  // Y por ultimo modificamos el state para actualizar segun los cambios que se
realizaron en los inputs.
  const handleFamiliarChange = (e) => {
    const familiares = [...familiar];
    familiares[e.target.id][e.target.dataset.name] = e.target.value;
    setFamiliar(familiares);
  };

  return (
    <form>
      <label htmlFor="nombre">Nombre:</label>
      <input
        type="text"
        name="nombre"
        value={persona.nombre}
        onChange={handlePersonaChange}
      />
      <input
        type="button"
        value="Agrega un Familiar"
        onClick={agregaFamiliar}
      />
      {familiar.map((el, i) => (
        <div key={`persona-${i}`}>
```

```
            <label htmlFor={`nombre-${i}`}>{`Familiar #${i + 1}`}</label>
            <input
              type="text"
              name={`nombre-${i}`}
              id={i}
              data-name="nombre"
              value={el.nombre}
              onChange={handleFamiliarChange} // Agregamos el metodo a cada input
que generamos
            />
          </div>
        ))}
        <input type="submit" value="Submit" />
      </form>
    );
}

export default Form;
```

Y con ese ultimo paso creamos un simple Formulario con inputs dinamicos.

Uncontrolled Components

Siempre se recomienda trabajar con Componentes Controlados para la implementacion de Forms. La alternativa a esto son los Componentes no controlados, en donde los datos del Form son manejados por el propio DOM. En lugar de escribir eventos para las actualizaciones del state, podemos usar una referencia para obtener los valores del formulario desde el DOM, o utilizar los metodos de `document` como por ejemplo `document.querySelector()` para obtener los valores del Formulario. En este tipo de formularios solo tenemos la posibilidad de validar en el momento en que hacemos el submit del form, cuando obtenemos los datos ingresados en el form.

```
// Utilizando Referencias

class UncontrolledForm extends React.Component {
  constructor() {
    super();
    this.handleSubmit = this.handleSubmit.bind(this);
    this.input = React.createRef();
  }

  handleSubmit(e) {
    e.preventDefault();
    const name = this.input.current.value;
    console.log(name);
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
```

```
            Name:
            <input type="text" ref={this.input} />
          </label>
          <input type="submit" value="Submit" />
        </form>
      );
    }
  }

  // Utilizando Selectores

  function Uncontrolled() {

    function handleSubmit(e) {
      e.preventDefault();
      const username = document.querySelector('input[name=username]').value;
      const password = document.querySelector('input[name=password]').value;
      console.log(username, password);
    }
    render() {
      return (
        <form onSubmit={handleSubmit}>
          <input name="username" placeholder="username ej: toni@gmail.com" />
          <input name="password" type="password" placeholder="password" />
          <input type="submit" />
        </form>
      );
    }
  }
```

Homework

Completa la tarea descrita en el archivo README