12. Formulari amb React hooks

Formulari amb React hooks

Els elements de formularis en HTML funcionen una mica diferent d'altres elements del DOM a React, ja que els elements de formularis conserven naturalment algun estat intern. Per exemple, aquest formulari només en HTML accepta un sol nom.

```
<form>
      <label>
      Nom:
      <input type=”text” name=”nom” />
      </label>
      <input type=”submit” value=”Enviar” \>
</form>
```

Formulari amb React hooks

Com veiem aquest formulari té el comportament predeterminat en HTML que consisteix a navegar a una pàgina nova quan l'usuari envia el formulari.

Si desitgem aquest comportament a React, simplement ja funciona així de base.

Però en la majoria de casos, és convenient tenir una funció en Javascript que s'encarregui de l'enviament del formulari, i que tingui accés a les dades que l'usuari va introduir al formulari.

En resum, la forma predeterminada per aconseguir això és una tècnica anomenada “components controlats”.

Formulari amb React hooks: Components controlats

A HTML, els elements de formularis com els <input>, <textarea> i el <select> normalment mantenen els seus propis estats i els actualitzem d'acord amb la interacció de l'usuari. A React, l'estat mutable es manté normalment en la propietat estat dels components, i només s'actualitza amb setState().

Podem combinar tots dos fent que l'estat de React sigui “l'única font de la veritat”. D'aquesta manera, els components React que renderitzin un formulari també controlen el que passa en aquest formulari amb les entrades subsegüents de l'usuari.

Un camp d'un formulari a on els valors del qual són controlats per React, és anomenat “component controlat”.

Formulari amb React hooks: Components controlats

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};

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

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

  handleSubmit(event) {
    alert('A name was submitted: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text" value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

Ja que l'atribut value és agregat al nostre element del formulari, el valor mostrat sempre serà el de this.state.value, fent que l'estat de React sigui la font de la veritat. Ja que handleChange corre cada cop que una tecla és oprimida per actualitzar l'estat de React, el valor mostrat serà actualitzat mentre que l'usuari escriu.

Amb un component controlat, el valor de l'input està dirigit sempre per l'estat de React. Si bé això vol dir que haureu d'escriure una mica més de codi, ara podreu passar també el valor a altres elements de la interfície d'usuari, o reiniciar-lo des d'altres manejadors d'esdeveniments.

Formulari amb React hooks: L'etiqueta textarea

En HTML, l'element <textarea> defineix el text pels seus fills:

```
<textarea>
  Hola, un texte de textarea.
</textarea>
```

Formulari amb React hooks: Components controlats

```
class EssayForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      value: 'Please write an essay about your favorite DOM element.'
    };

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

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

  handleSubmit(event) {
    alert('An essay was submitted: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Essay:
          <textarea value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

A React, un <textarea> utilitza un atribut value al seu lloc. D'aquesta manera, un formulari que fa ús d'un <textarea> pot ser escrit de manera semblant a un formulari que utilitza un camp en una sola línia:

Recorda que this.state.value és inicialitzat al constructor, de manera que l'àrea de text comenci amb una mica de text.

Formulari amb React hooks

Per poder pràcticar correctament amb els formularis, crearem una ruta nova anomenada contacta en index.js.

No ens hem d'oblidar també d'afegir-ho en el menú, per aixi facilitar la navegació.

Formulari amb React hooks

```
import React from 'react';

const MyForm = () => {
<div>
      <h1>Formulari simple</h1>
      <form>
        <div>
          <label>
            Nom:
            <input
              type="text"
              name="name“
            />
          </label>
        </div>
   <button type="submit">Enviar</button>
      </form>
</div>
}

export default MyForm;
```

Com hem comentat anteriorment, els hooks són funcions especials que ens permeten utilitzar característiques de React en components funcionals. Per exemple, podeu utilitzar variables d'estat i són boniques.

Ara veurem l'exemple equivalent d'un formulari usant hooks.

El hook que més farem servir és el de useState perquè ens permet definir variables d'estat dins d'un component funcional.

Iniciem realitzant el jsx d'un camp.

Formulari amb React hooks

```
import React from 'react';

const MyForm = () => {

<div>
      <h1>Formulari simple</h1>
      <form>
        <div>
          <label>
            Nom:
            <input
              type="text"
              name="name"
          value={name}
              onChange={handleChange} />
          </label>
        </div>
   <button type="submit">Enviar</button>
      </form>
</div>
}
export default MyForm;
```

```
const [values, setValues] =
React.useState({
    name: '',
})
const handleChange = (e) => {
  const { name, value } = e.target
  setValues({ ...values, [name]: value })
}

const { name } = values
```

No s'ha d'oblidar el definir una estructura useState per gestionar els diferents camps del formulari.

Ara anem a definir un atribut onChange perquè utilitzi el hook useState per enmagatzemar els valors aportats a través del formulari. Aquest alhora anomena la funció fletxa que enmagatzema les dades en el hook.

Formulari amb React hooks

```
import React from 'react';

const MyForm = () => {

<div>
      <h1>Formulari simple</h1>
      <form>
        <div>
          …
        </div>
   <button type="submit">Enviar</button>
      </form>
      <div>
        <h2>Valors del formulari</h2>
        <p>{JSON.stringify(values)}</p>
      </div>
</div>
}

export default MyForm;
```

```
const [values, setValues] =
React.useState({
    name: ‘’,
})
const handleChange = (e) => {
  const { name, value } = e.target
  setValues({ ...values, [name]: value })
}

const { name } = values
```

Ara afegirem un bloc delimitat amb les etiquetes div i que ens permetrà veure el valor dels hooks i alhora corraborar que s’han enmagatzemat correctamente els valors.

Formulari amb React hooks

```
const validateAll = () => {
        const { name } = values
        const validations = { name: '' }
        let isValid = true
        if (!isValid) {
            setValidations(validations)
        }
        return isValid
}
const validateOne = (e) => {
        const { name } = e.target
        const value = values[name]
        let message = ''
        if (!value) {
          message = `${name} és requerit`
        }
        if (value && name === 'name' && (value.length < 3 || value.length > 50)) {
          message = 'El nom té que contenir entre 3 i 50 caràcters'
        }
        setValidations({ ...validations, [name]: message })
}
```

Un pas més serà el validar la dada inclosa en el camp name i per això tenim que definir una altre estructura useState per enmagatzemar l'estat de la validació.

Adicionalment tenim que desenvolupar una funció per validar individualment el camp i el formulari sencer previ la seva execució.

Formulari amb React hooks

```
const handleSubmit = (e) => {
        e.preventDefault()

        const isValid = validateAll()

        if (!isValid) {
          return false
        }

        alert(JSON.stringify(values))
}

const { name} = values
const { name: nameVal} = validations

return (
    <div>
      <h1>Formulari simple</h1>
      <form onSubmit={handleSubmit}>
        <div>
```

També una funció per administrar l'acció del submit i que ens mostrarà adicionalment les dades en un alert().
Aquesta funció será invocada desde l'atribut onSubmit i será activat en el moment que l'usuari clicki el botò enviar.

Adicionalment en el camp input afegirem un últim atribut anomenat onBlur que validara la dada quan és retiri el focus del camp.

```
<label>
          Nom:
          <input
            type="text"
            name="name"
            value={name}
            onChange={handleChange}
            onBlur={validateOne} />
</label>
<div>{nameVal}</div>
```

Formulari amb React hooks

```
const [validations, setValidations] = React.useState({
        name: '',
        email: '',
})

const validateAll = () => {
 const { name, email } = values
 const validations = { name: '', email: '' }
 let isValid = true
 if (!isValid) {
  setValidations(validations)
 }
 …
 if (!email) {
  validations.email = 'Email es requerit'
  isValid = false
 }
 if (email && !/\S+@\S+\.\S+/.test(email)) {
  validations.email = 'El format del email ha de ser com example@mail.com'
  isValid = false
 }
        return isValid
```

Anem a adaptar el formulari per aceptar el camp email i corresponentment actualitzem totes les funcions de validació per així validar si acompleix el format.

```
<div>
        <label>
          Email: <input
            type="email"
            name="email"
            value={email}
            onChange={handleChange}
            onBlur={validateOne} />
        </label>
        <div>{emailVal}</div>
</div>
```

Formulari amb React hooks

```
const [validations, setValidations] = React.useState({
    name: '',
    email: '',
    gender: '',
  })

const validateAll = () => {
 const { name, email, gender } = values
 const validations = { name: '', email: '', gender: '' }
 …
if (!gender) {
      validations.gender = 'El genere és requerit'
      isValid = false
}
  return isValid
}
…
const { name, email, gender } = values
const { name: nameVal, email: emailVal,
gender: genderVal } = validations
```

Anem a adaptar el formulari per aceptar el camp genere i corresponentment actualitzem totes les funcions de validació per així validar si acompleix el format.

```
<label>
    Dona
    <input type="radio"
      name="gender" value="F"
      onChange={handleChange}
      onBlur={validateOne} />
  </label>
  <label>
    Home
    <input

type="radio“ name="gender“ value="M"
onChange={handleChange}
      onBlur={validateOne} />
  </label>
  <div>{genderVal}</div>
```