

React.js with PHP API REST

9. Formulaire contrôlé et validation en temps réel

Nous allons créer un nouveau composant ModalContact.jsx pour le formulaire de contact.

Nous reprenons le code du composant Contact.jsx tel qu'il était à la fin du TP7 (avant l'ajout de la validation onSubmit)

Vous pouvez récupérer le code sur la branche Step07 et le coller dans le nouveau fichier ModalContact.jsx

```
src > components > contact > ModalContact.jsx > ModalContact
1  function ModalContact() {
2
3      const handleFormSubmit = (event) => {
4          event.preventDefault();
5          const formData = new FormData(event.target);
6          const jsonData = Object.fromEntries(formData.entries());
7          console.log(jsonData);
8          const submitData = async () => {
9              const url = "http://api.php-blog-project.loc/contact";
10             const options = {
11                 method: "POST",
12                 body: JSON.stringify(jsonData),
13             }
14             const response = await fetch(url, options);
15             console.log(response);
16             if (!response.ok) {
17                 throw new Error('Erreur de réseau');
18             }
19             const json = await response.json();
20             console.log(json.result);
21         }
22         submitData();
23     };
24 }
```

Pour le formulaire dans le render :

```
<div className="modal-body">
  <form id="contactForm" onSubmit={handleFormSubmit}>
    <div className="mb-3">
      <label className="form-label" htmlFor="fullname">Name</label>
      <input
        className="form-control" name="fullname" type="text" placeholder="Name" />
    </div>
    <div className="mb-3">
      <label className="form-label" htmlFor="email">Email Address</label>
      <input
        className="form-control" name="email" type="email" placeholder="Email Address" />
    </div>
    <div className="mb-3">
      <label className="form-label" htmlFor="message">Message</label>
      <textarea
        className="form-control" name="message" type="text"
        placeholder="Message" style={{height: '10rem'}} />
    </div>
    <div className="mb-1 text-center">
      <button className="btn btn-success" name="send" type="submit">
        Envoyer
      </button>
    </div>
  </form>
</div>
```

Pensez à renommer le composant en ModalContact (suite au copié-collé 😊), puis remplacer le composant Contact par ModalContact dans le composant NavBar

```
src > components > navbar > ⚙️ NavBar.jsx > 📦 NavBar
1  import { useState } from "react";
2  import ModalContact from "../contact/ModalContact";
3
4  function NavBar() {
```

```
src > components > navbar > ⚙️ NavBar.jsx > 📦 NavBar
4  function NavBar() {
45
46    </div>
47    </nav>
48    <ModalContact />
49  }
50  export default NavBar;
```

Nous allons créer un state pour gérer les différentes valeurs du formulaire.

```
src > components > contact > ⚙️ ModalContact.jsx > 📦 ModalContact
1  import { useState } from "react";
2
3  function ModalContact() {
4
5    const [formState] = useState({
6      fullname: "",
7      email: "",
8      message: ""
9    })
10
```

Nous relierons chaque élément du formState aux inputs correspondants dans le formulaire

```
<div className="modal-body">
  <form id="contactForm" onSubmit={handleFormSubmit}>
    <div className="mb-3">
      <label className="form-label" htmlFor="fullname">Name</label>
      <input
        className="form-control" name="fullname" type="text"
        placeholder="Name" value={formState.fullname} />
    </div>
    <div className="mb-3">
      <label className="form-label" htmlFor="email">Email Address</label>
      <input
        className="form-control" name="email" type="email"
        placeholder="Email Address" value={formState.email} />
    </div>
    <div className="mb-3">
      <label className="form-label" htmlFor="message">Message</label>
      <textarea
        className="form-control" name="message" type="text"
        placeholder="Message" style={{height: '10rem'}} value={formState.message}>
      </textarea>
    </div>
    <div className="mb-1 text-center">
      <button className="btn btn-success" name="send" type="submit">
        Envoyer
      </button>
    </div>
  </form>
</div>
```

Puis nous ajoutons la méthode `handleInputChange()` pour mettre à jours l'élément du `formState` correspondant à l'input dont la valeur change (événement `onChange`)

```
src > components > contact > ModalContact.jsx > ModalContact
3  function ModalContact() {
4
5      const [formState, setFormState] = useState({
6          fullname: "",
7          email: "",
8          message: ""
9      })
10
11      const handleInputChange = (event) => {
12          const { name, value } = event.target;
13          setFormState((oldState) => {
14              console.log(name, value)
15              return {...oldState, [name]: value}
16          });
17      };
18  }
```

Nous récupérons les attributs `name` et `value` de l'élément ayant déclenché l'événement `onChange` (ligne 12)

Puis nous mettons à jour l'élément du `formState` correspondant en récupérant l'ancien state et en modifiant uniquement l'élément qui vient de changer (ligne 13 à 16)

Pensez à ajouter l'événement pour chaque input du formulaire

```
<input
  className="form-control" name="fullname" type="text"
  placeholder="Name" value={formState.fullname} onChange={handleInputChange}/>
</input>

<input
  className="form-control" name="email" type="email"
  placeholder="Email Address" value={formState.email} onChange={handleInputChange}/>
</input>

<textarea
  className="form-control" name="message" type="text" onChange={handleInputChange}
  placeholder="Message" style={{height: '10rem'}} value={formState.message}>
</textarea>
```

Regardez ce qui se passe en console à chaque modification d'un champ du formulaire.

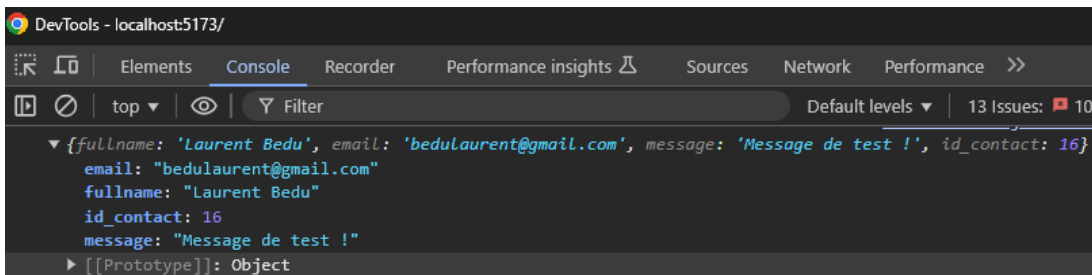
-> voir page suivante

Il nous reste à adapter la fonction `handleFormSubmit()`

```
ModalContactModel.jsx > ModalContact
3  function ModalContact() {
18
19      const handleFormSubmit = (event) => {
20          event.preventDefault();
21          console.log(formState);
22          const submitData = async () => {
23              const url = "http://api.php-blog-project.loc/contact";
24              const options = {
25                  method: "POST",
26                  body: JSON.stringify(formState),
27              }
28              const response = await fetch(url, options);
29              console.log(response);
30              if (!response.ok) {
31                  throw new Error('Erreur de réseau');
32              }
33              const json = await response.json();
34              console.log(json.result);
35          }
36          submitData();
37      };
}
```

Les valeurs contenues dans le formulaire (contrôlé) sont directement accessibles car stockées dans le `formState` (en temps réel), lors de l'envoi du formulaire nous passons cet objet (stringifié) dans le `body` de la requête.

Vous pouvez tester que l'envoi du formulaire fonctionne est qu'une ligne est bien créée en DB.



The screenshot shows the Chrome DevTools Console with the 'Console' tab selected. A log message is displayed, showing a JSON object with the following properties: `fullname: 'Laurent Bedu', email: 'bedulaurent@gmail.com', message: 'Message de test !', id_contact: 16`. The object is expanded, showing the values for each property. The console also shows the prototype chain as `[[Prototype]]: Object`.

Validation en temps réel du formulaire

Nous ajoutons 2 states, un premier pour gérer les messages d'erreur liées à la validation du formulaire. Un autre pour stocker la validité complète du formulaire (si tous les champs sont valides).

-> voir page suivante

```
src > components > contact > ModalContact.jsx > ModalContact
3  function ModalContact() {
4
5      const [formState, setFormState] = useState({
6          fullname: "",
7          email: "",
8          message: ""
9      });
10     const [formErrors, setFormErrors] = useState({});
11     const [isFormValid, setIsFormValid] = useState(false);
12 }
```

Nous créons ensuite une fonction chargée de valider le formulaire

```
src > components > contact > ModalContact.jsx > ModalContact
3  function ModalContact() {
12
13     const validateForm = () => {
14         const errors = {};
15         if (formState.fullname.trim().length < 2) {
16             errors.fullname = "Le nom doit contenir au moins 2 caractères.";
17         }
18         const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
19         if (!emailRegex.test(formState.email)) {
20             errors.email = "L'email n'est pas valide.";
21         }
22         if (formState.message.trim().length < 3) {
23             errors.message = "Le message doit contenir au moins 3 caractères.";
24         }
25         setFormErrors(errors);
26         setIsFormValid(Object.keys(errors).length === 0);
27     };
}
```

Nous créons un objet pour stocker les erreurs éventuelles (ligne 14) puis nous vérifions successivement que les 3 valeurs saisies dans le formulaire sont correctes. Si ça n'est pas le cas, nous ajoutons un message dans l'objet errors. (ligne 15 à 24)

Enfin nous mettons à jour nos 2 states suite à ces vérifications (ligne 25 et 26) Le formulaire est entièrement valide si l'objet erreur ne contient aucun message.

Nous ajoutons un useEffect qui exécutera la validation du formulaire à chaque changement dans le formState, donc à chaque modification apportée dans un champ du formulaire.

```
src > components > contact > ModalContact.jsx > ModalContact
1  import { useState, useEffect } from "react";
2
3  function ModalContact() {
4
5      const [formState, setFormState] = useState({
6          fullname: "",
7          email: "",
8          message: ""
9      });
10     const [formErrors, setFormErrors] = useState({});
11     const [isFormValid, setIsFormValid] = useState(false);
12
13     useEffect(() => {
14         validateForm();
15     }, [formState]);
16
17     const validateForm = () => {
```

Nous modifions le render pour afficher les erreurs éventuelles sous les champs de saisie

```
<div className="mb-3">
  <label className="form-label" htmlFor="fullname">Name</label>
  <input
    className="form-control"
    name="fullname"
    type="text"
    placeholder="Name"
    value={formState.fullname}
    onChange={handleInputChange} />
  {formErrors.fullname && <div className="invalid-feedback">{formErrors.fullname}</div>}
</div>
```

```
<div className="mb-3">
  <label className="form-label" htmlFor="email">Email Address</label>
  <input
    className="form-control"
    name="email"
    type="email"
    placeholder="Email Address"
    value={formState.email}
    onChange={handleInputChange} />
  {formErrors.email && <div className="invalid-feedback">{formErrors.email}</div>}
</div>
```

```
<div className="mb-3">
  <label className="form-label" htmlFor="message">Message</label>
  <textarea
    className="form-control"
    name="message"
    placeholder="Message"
    style={{ height: '10rem' }}
    value={formState.message}
    onChange={handleInputChange}>
  </textarea>
  {formErrors.message && <div className="invalid-feedback">{formErrors.message}</div>}
</div>
```

Nous désactivons également le bouton submit si le formulaire n'est pas complètement valide

```
<div className="mb-1 text-center">
  <button className="btn btn-success" name="send" type="submit" disabled={!isFormValid}>
    Envoyer
  </button>
</div>
```

Il nous reste à créer une méthode pour utiliser la validation de formulaire de Bootstrap 5 en modifiant la className du champ suivant la validité de la saisie

```
src > components > contact > ModalContact.jsx > ModalContact
3   function ModalContact() {
61     const getInputClass = (field) => {
62       if (formState[field] === "") return "form-control";
63       return formErrors[field] ? "form-control is-invalid" : "form-control is-valid";
64     };
65   }
```

Si le state formErrors contient un message d'erreur pour le champs en question, celui ci n'est pas valide

Nous modifions nos champs pour utiliser cette méthode

```
<input
  className={getInputClass("fullname")}
  name="fullname"
  type="text"
  placeholder="Name"
  value={formState.fullname}
  onChange={handleInputChange} />
{formErrors.fullname && <div className="invalid-feedback">{formErrors.fullname}</div>}
```

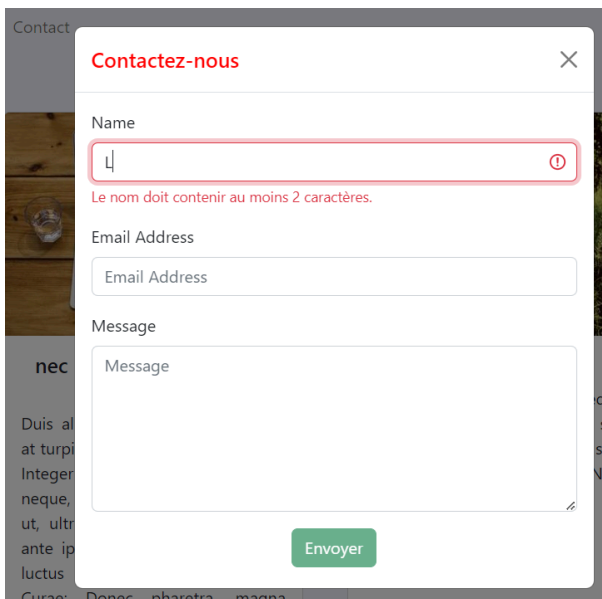
```
<input
  className={getInputClass("email")}
  name="email"
  type="email"
  placeholder="Email Address"
  value={formState.email}
  onChange={handleInputChange} />
{formErrors.email && <div className="invalid-feedback">{formErrors.email}</div>}
```

```
<textarea
  className={getInputClass("message")}
  name="message"
  placeholder="Message"
  style={{ height: '10rem' }}
  value={formState.message}
  onChange={handleInputChange}>
</textarea>
{formErrors.message && <div className="invalid-feedback">{formErrors.message}</div>}
```

Ressources :

<https://getbootstrap.com/docs/5.3/forms/validation/>

Testons notre formulaire contrôlé avec validation en temps réel



Contactez-nous

Name

l

Le nom doit contenir au moins 2 caractères.

Email Address

Email Address

Message

Message

Envoyer

Contactez-nous

Name

La

✓

Email Address

Email Address

Message

Message

Envoyer

Contactez-nous

Name

Laurent

✓

Email Address

b

ⓘ

L'email n'est pas valide.

Message

Message

Envoyer

Contactez-nous

Name

Laurent

✓

Email Address

bedulaurent@gmail.

ⓘ

L'email n'est pas valide.

Message

Message

Envoyer

Contactez-nous

Name
Laurent ✓

Email Address
bedulaurent@gmail.com ✓

Message
B ①

Le message doit contenir au moins 3 caractères.

Envoyer

Contactez-nous

Name
Laurent ✓

Email Address
bedulaurent@gmail.com ✓

Message
Bonjour ! ✓

Envoyer

Nous allons maintenant ajouter le reset du formulaire si l'envoi s'est bien passé en même temps que la fermeture du modal à l'aide d'un useRef

```
src > components > contact > ModalContact.jsx > ModalContact
1  import { useState, useEffect, useRef } from "react";
2
3  function ModalContact() {
4
5      const [formState, setFormState] = useState({
6          fullname: "",
7          email: "",
8          message: ""
9      });
10     const [formErrors, setFormErrors] = useState({});
11     const [isFormValid, setIsFormValid] = useState(false);
12
13     const closeBtnRef = useRef();
14
```

```

<div className="modal-header">
  <h1 className="modal-title fs-5" id="exampleModalLabel">Contactez-nous</h1>
  <button type="button" className="btn-close"
    data-bs-dismiss="modal" aria-label="Close"
    ref={closeBtnRef}
  ></button>
</div>

```

```

src > components > contact > ModalContact.jsx > ModalContact
3  function ModalContact() {
42
43    const handleFormSubmit = (event) => {
44      event.preventDefault();
45      console.log(formState);
46      const submitData = async () => {
47        const url = "http://api.php-blog-project.loc/contact";
48        const options = {
49          method: "POST",
50          body: JSON.stringify(formState),
51        }
52        const response = await fetch(url, options);
53        console.log(response);
54        if (!response.ok) {
55          throw new Error('Erreur de réseau');
56        }
57        const json = await response.json();
58        console.log(json.result);
59        closeBtnRef.current.click();
60        setFormState({
61          fullname: "",
62          email: "",
63          message: ""
64        })
65      }
66      submitData();
67    };

```

Composant AlertMessage

Nous allons coder un composant AlertMessage (ce qu'il était demandé de faire à la fin du TP 8). Nous allons nous servir du composant Alert de Bootstrap 5.3

Nous créons un objet permettant de fixer le type de l'alerte (success, danger, etc ...) un peu comme nous l'avons fait avec une enum en PHP.

```

src > components > messages > AlertMessage.jsx > ...
1
2  const AlertType = Object.freeze({
3    PRIMARY: "primary",
4    SECONDARY: "secondary",
5    SUCCESS: "success",
6    DANGER: "danger",
7    WARNING: "warning",
8    INFO: "info",
9    LIGHT: "light",
10   DARK: "dark",
11  });

```

Puis nous fixons les différentes props dont nous allons avoir besoin en fonction de ce que nous souhaitons contrôler sur l'alerte (son type, si elle est affichée ou masquée, etc ...)

```
src > components > messages > AlertMessage.jsx > AlertMessage
13
14 function AlertMessage(props) {
15   const {
16     type = AlertType.PRIMARY,
17     dismissible = false,
18     show = false,
19     children,
20     className,
21     closeFn,
22   } = props;
23
```

Le type correspondra à une valeur de l' "enum" AlertType.

dismissible pour indiquer si elle est détruite à la fermeture (plus utilisable ensuite car supprimé du DOM)

show indique si elle est visible ou masquée

children sert à ajouter le contenu de l'alerte (son message au format html)

className permet d'ajouter des classes pour personnaliser l'alert (comme text-center par exemple)

closeFn permet de passer une méthode pour fermer l'alert (sans la supprimer du DOM) et pouvoir ainsi la réutiliser

Nous ajoutons les proptypes

```
src > components > messages > AlertMessage.jsx > AlertMessage
1 import PropTypes from "prop-types";
2
```

```
src > components > messages > AlertMessage.jsx > AlertMessage
14 function AlertMessage(props) {
49
50 AlertMessage.propTypes = {
51   type: PropTypes.oneOf(Object.values(AlertType)),
52   dismissible: PropTypes.bool,
53   show: PropTypes.bool,
54   children: PropTypes.oneOfType([
55     PropTypes.arrayOf(PropTypes.element),
56     PropTypes.element,
57   ]),
58   className: PropTypes.string,
59   closeFn: PropTypes.func,
60 }
61
```

Enfin, nous exportons AlertType et AlerMessage

```
61
62 export { AlertMessage, AlertType };
63
```

Nous allons maintenant utiliser ce composant dans ModalContact pour afficher un message à l'utilisateur une fois l'envoi du formulaire effectué.

Nous commençons par importer AlertMessage et AlertType

```
src > components > contact > ModalContact.jsx > ModalContact
1  import { useState, useEffect, useRef } from "react";
2  import { AlertMessage, AlertType } from "../messages/AlertMessage";
3
4  function ModalContact() {
5
```

Nous créons un state pour pouvoir modifier les props passées au composant AlertMessage (pour pouvoir par exemple afficher l'alerte ou la masquer)

```
src > components > contact > ModalContact.jsx > ModalContact
4  function ModalContact() {
5
6      const [formState, setFormState] = useState({
7          fullname: "",
8          email: "",
9          message: ""
10     });
11     const [formErrors, setFormErrors] = useState({});
12     const [isFormValid, setIsFormValid] = useState(false);
13     const [alertProps, setAlertProps] = useState({});
14     const closeBtnRef = useRef();
15
16     useEffect(() => {
17         validateForm();
18     }, [formState]);
19
```

Puis nous ajoutons notre composant dans le render de ModalContact

```
src > components > contact > ModalContact.jsx > ModalContact
4  function ModalContact() {
95
96     return (
97         <AlertMessage closeFn={()=>{setAlertProps({show: false})}} {...alertProps} />
98         <div className="modal fade" id="myModal" tabIndex="-1" aria-labelledby="exampleModalLabel">
99             <div className="modal-dialog">
100                 <div className="modal-content">
101                     <div className="modal-header">
```

La props closeFn permettra de fermer (masquer l'alerte sans la supprimer du DOM) grâce à la modification du state alertProps. Les props contenues dans le state alertProps sont déstructurées pour être passées au composant

Avant de gérer l'affichage de l'alerte dans la méthode handleFormSubmit() nous créons une méthode resetForm() permettant de fermer la modale contenant le formulaire, de reset le formulaire et de masquer l'alerte (avec un délai de 5 sec. par exemple)

```
src > components > contact > ModalContact.jsx > ModalContact
4  function ModalContact() {
80
81     const resetForm = (reset = false) => {
82         closeBtnRef.current.click();
83         setTimeout(() => {
84             setAlertProps({show: false})
85             reset && setFormState({fullname: "", email: "", message: ""})
86         }, 5000);
87     }
88
```

Nous avons tout ce qu'il faut pour modifier la méthode `handleFormSubmit()`

```
src > components > contact > ModalContact.jsx > ModalContact
4  function ModalContact() {
43
44      const handleFormSubmit = (event) => {
45          event.preventDefault();
46          console.log(formState);
47          const submitData = async () => {
48              try {
49                  const url = "http://api.php-blog-project.loc/contact";
50                  const options = {
51                      method: "POST",
52                      body: JSON.stringify(formState),
53                  }
54                  const response = await fetch(url, options);
55                  console.log(response);
56                  if (!response.ok) {
57                      throw new Error('Erreur de réseau');
58                  }
59                  const json = await response.json();
60                  console.log(json.result);
61                  const alertMessage = (
62                      <span>Merci pour votre message {formState.fullname},<br/>
63                      vous recevrez très prochainement une réponse à l'adresse
64                      mail indiquées : {formState.email}</span>
65                  )
66                  setAlertProps({type:AlertType.SUCCESS, show: true, children : alertMessage});
67                  resetForm(true);
68              } catch(error){
69                  setAlertProps({
70                      show: true,
71                      children: "Une erreur est survenue lors de l'envoi du message.",
72                      type: AlertType.DANGER
73                  });
74                  resetForm();
75              }
76          }
77      }
78      submitData();
79  };

```

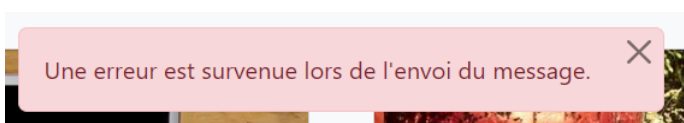
Nous ajoutons un try catch pour gérer l'erreur qui peut être déclenchée ligne 57 en cas de réponse incorrecte à la requête vers l'API Rest

En cas d'erreur (le message n'a pas pu être inséré en DB) nous affichons une alerte de type danger pour l'indiquer à l'utilisateur (lignes 70 à 74) puis avec la méthode `resetForm()`, nous masquons le formulaire de contact puis après un délai de 5 secondes, nous masquons l'alerte et faisons le reset du formulaire (ligne 75)

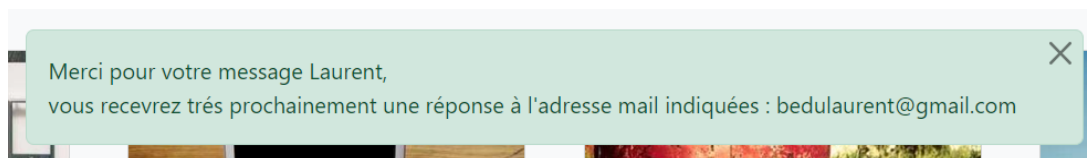
Si tout s'est bien passé, nous construisons le message à afficher dans l'alerte (ligne 61 à 65), nous affichons l'alerte de type success (ligne 66) et appelons `resetForm()` ensuite.

Nous passons aux tests ...

Tester l'envoi du formulaire de contact avec wamp fermé pour simuler un échec.



Puis ouvrir wamp pour envoyer à nouveau le formulaire



Pour finir notre application, il reste à développer les pages `series/articles/:id` et `techs/articles/:id` que nous avons fait en PHP MVC et à mettre en place un routeur.

1. Essayer de mettre en place un routeur en utilisant le package React Router v6.4
<https://reactrouter.com/en/main>
2. Développez les pages `SerieArticlesScreen` et `TechArticlesScreen` permettant d'afficher les articles pour une série ou une techs donnée en fonction de l'id.

git : https://github.com/DWWM-23526/REACT_BLOG_PROJECT/tree/Step09