Notions

- Pour gérer un formulaire, il est possible de tout réaliser « manuellement » ou de passer par un module supplémentaire de node.js.
- Le module « **formik** », recommandé par REACT met à disposition des fonctionnalités permettant **d'identifier et de gérer des événements** sur le formulaire.
- Les « states » contenant les informations saisies ne sont plus nécessaires, car c'est « formik » qui va gérer les données.
- 3 parties principales sont à renseigner avec l'utilisation de « formik » :
  - **mapPropsToValues** : créé un « maping » entre les champs et les valeurs, basé sur l'attribut « name » des inputs
  - **validate** : gère les règles de validation du formulaire et des champs
  - **handleSubmit** : gère la soumission du formulaire

Exemples

```
npm install --save formik
```

```
import {withFormik} from "formik";
```

```
<div className="form-group">
    <label htmlFor="titre">Titre du livre </label>
    <input type="text" className="form-control"
    name= "titre"
    value={this.props.values.titre}
    onChange={this.props.handleChange}
    onBlur={this.props.handleBlur}
    />
    {
        (this.props.touched.titre && this.props.errors.titre) &&
        <span style={{color:"red"}}>{this.props.errors.titre}</span>
    }
</div>
```

```
<Bouton typeBtn={"warning"} clic={this.props.handleReset} >Reset</Bouton>
<Bouton typeBtn={"primary"} clic={this.props.handleSubmit} >Valider</Bouton>
```

```
export default withFormik({
    mapPropsToValues: () => ({
        titre:'',
        auteur:'',
        nbPages:''
    }),
    validate: values => {
        const errors = {};
        Object.keys(values).forEach(v => {
            if(!values[v]) {
                errors[v] = "Le champ est obligatoire !";
            }
        })
        if(values.titre.length > 15) {
            errors.titre = "Le titre doit avoir moins de 15 caractères !"
        }
        return errors;
    },
    handleSubmit: (values,{props}) => {
        props.validationAjout(values.titre,values.auteur,values.nbPages);
    }
})(FormulaireAjout);
```

Notions

- Pour faciliter la validation des données, il est possible d'utiliser un autre module, « Yup », proposant de nombres fonctions de vérification
- Pour utiliser « Yup » avec « Formik », il faudra remplacer la fonction « validate » par la fonction « validationSchema » et lui fournir un Schema.
- « Yup » permet de définir un schéma avec la fonction shape() sur les objets : Yup.object().shape({*leSchema*})
- Pour utiliser les différentes fonctions de vérification de « Yup », se reporter à la documentation : https://github.com/jquense/yup

Exemples

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

```
import * as Yup from "yup";
```

```
export default withFormik({
    mapPropsToValues: () => ({
        titre:'',
        auteur:'',
        nbPages:''
    }),
    validationSchema: Yup.object().shape({
        titre: Yup.string()
            .min(3,'Le titre doit avoir plus de 3 caractères')
            .max(15,'Le titre doit faire moins de 15 caractères')
            .required('Le titre est obligatoire !'),
        auteur: Yup.string()
            .min(3," L'auteur doit avoir plus de 3 caractères")
            .required('L\'auteur est obligatoire !'),
        nbPages: Yup.number()
            .lessThan(1000,'Nombre de page < 1000')
            .moreThan(50,'Nombre de page > 50')
    }),
    handleSubmit: (values,{props}) => {
        props.validationAjout(values.titre,values.auteur,values.nbPages);
    }
})(FormulaireAjout);
```

Transformation en Entier

```
<div className="form-group">
    <label htmlFor="titre">Nombre de pages </label>
    <input type="number" className="form-control"
    name="nbPages"
    value={this.props.values.nbPages}
    onChange={(event) => this.props.setFieldValue("nbPages", +event.target.value)}
    onBlur={this.props.handleBlur}
    />
    {
        (this.props.touched.nbPages && this.props.errors.nbPages) &&
        <span style={{color:"red"}}>{this.props.errors.nbPages}</span>
    }
</div>
```