React
Работа с формами

Контролируемые и неконтролируемые компоненты
Своя валидация
Yup
React-hook-form
Formik
Redux-Form

Форма во Front-end-е (<Form />)

Формы в front-end разработке играют важную роль, поскольку они служат основным способом взаимодействия пользователей с веб-приложениями. Вот ключевые значения форм:

- **Сбор данных:** Формы используются для ввода и отправки данных на сервер. Например, регистрация, авторизация, заполнение анкет, обратная связь.
- **Интерактивность:** Формы предоставляют пользователям возможность взаимодействовать с приложением. Это может быть отправка сообщений, оставление комментариев, загрузка файлов и другое.
- **Валидация данных:** С помощью форм можно проверять данные, введенные пользователями, как на стороне клиента, так и на стороне сервера. Это повышает качество вводимой информации и предотвращает ошибки.
- **Динамическое управление содержимым:** Современные формы могут динамически изменяться в зависимости от действий пользователя (например, скрывать или показывать поля).
- **Улучшение UX/UI:** Хорошо спроектированные формы с удобным пользовательским интерфейсом делают взаимодействие с приложением приятным и эффективным, улучшая общий пользовательский опыт.
- **Асинхронная обработка данных:** С использованием AJAX или fetch API можно отправлять данные формы без перезагрузки страницы, что ускоряет и упрощает работу с приложением.

Контролируемые и неконтролируемые компоненты

Контролируемые и неконтролируемые компоненты в React относятся к различным подходам управления состоянием элементов формы (например, <input>, <textarea>, <select>). Понимание их разницы поможет вам выбрать наиболее подходящий способ работы с формами в зависимости от конкретных требований приложения.

Разница между контролируемыми и неконтролируемыми компонентами:

Контроль состояния:

- В контролируемых компонентах состояние хранится в state React.
- В неконтролируемых компонентах состояние хранится в DOM.

Обработка данных:

- В контролируемых компонентах каждое изменение данных немедленно попадает в состояние.
- В неконтролируемых компонентах данные доступны только при обращении к элементу через ref.

Когда использовать:

- Контролируемые компоненты лучше подходят для сложных форм с валидацией, когда нужно постоянно контролировать и обрабатывать ввод данных.
- Неконтролируемые компоненты полезны для простых форм или если нужно минимизировать количество рендеров и не требуется полное управление значениями элементов.

Контролируемые компоненты обеспечивают полный контроль над данными формы и их валидностью, но требуют больше кода и могут увеличивать количество рендеров. Неконтролируемые компоненты проще в настройке и могут быть более эффективными в плане производительности, но сложнее в управлении и обработке данных. Выбор подхода зависит от сложности формы и требований к управлению состоянием.

Контролируемые компоненты

Контролируемые компоненты — это те, где React отслеживает состояние ввода с помощью state. Все изменения в форме управляются React, что позволяет больше контролировать логику формы.

Ключевые моменты:

- **value** задает текущее состояние поля ввода.
- **onChange** обновляет состояние при каждом изменении в поле.

```jsx
import { useState } from "react";

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

  const handleSubmit = (e) => {
    e.preventDefault();
    alert(`Submitted Name: ${name}`);
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input
          type="text"
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
      </label>
      <button type="submit">Submit</button>
    </form>
  );
}

export default ControlledForm;
```

Неконтролируемые компоненты

В неконтролируемых компонентах состояние формы не управляется React, и доступ к значению формы получают через рефы.

Ключевые моменты:

- В **ref** сохраняется ссылка на элемент DOM.
- **useRef** не вызывает повторный рендеринг компонента, когда его значение изменяется.

```jsx
import { useRef } from "react";

function UncontrolledForm() {
  const nameInput = useRef(null);

  const handleSubmit = (e) => {
    e.preventDefault();
    alert(`Submitted Name: ${nameInput.current.value}`);
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" ref={nameInput} />
      </label>
      <button type="submit">Submit</button>
    </form>
  );
}

export default UncontrolledForm;
```

Валидация

React поддерживает различные способы валидации данных в формах. Можно использовать валидацию на стороне клиента, простую логику проверки внутри компонента, или подключать библиотеки, такие как Formik или react-hook-form.

Контролируемые компоненты позволяют более точно управлять поведением формы.

Неконтролируемые компоненты используют рефы для работы с элементами напрямую.

Используйте валидацию данных для улучшения UX и предотвращения ошибок при отправке формы.

```jsx
function FormWithValidation() {
  const [name, setName] = useState("");
  const [error, setError] = useState("");

  const handleSubmit = (e) => {
    e.preventDefault();
    if (name.trim() === "") {
      setError("Name is required");
    } else {
      setError("");
      alert(`Submitted Name: ${name}`);
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input
          type="text"
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
      </label>
      {error && <p style={{ color: "red" }}>{error}</p>}
      <button type="submit">Submit</button>
    </form>
  );
}

export default FormWithValidation;
```

Своя валидация

Для React можно создать простую библиотеку для управления формами с валидацией, используя хуки.

Хук **useForm**:

- Управляет состоянием значений полей и ошибок.
- Обрабатывает изменение полей формы через handleChange.
- Валидирует каждое поле с помощью функции валидатора.
- Валидация формы вызывается через validateForm, чтобы проверить все поля.

```
import { useState } from 'react';
export function useForm(initialValues, validators = {}) {
  const [values, setValues] = useState(initialValues);
  const [errors, setErrors] = useState({});
  const handleChange = (e) => {
    const { name, value } = e.target;
    setValues({
      ...values,
      [name]: value,
    });
    if (validators[name]) {
      const error = validateField(name, value);
      setErrors((prevErrors) => ({
        ...prevErrors,
        [name]: error,
      }));
    }
  };
  const validateField = (name, value) => {
    const validator = validators[name];
    if (validator) {
      return validator(value);
    }
    return '';
  };
  const validateForm = () => {
    const newErrors = {};
    Object.keys(values).forEach((key) => {
      newErrors[key] = validateField(key, values[key]);
    });
    setErrors(newErrors);
    return !Object.values(newErrors).some((error) => error);
  };
  return {
    values,
    errors,
    handleChange,
    validateForm,
  };
}
```

Валидационные функции (required, minLength, email):

Простые функции для проверки полей формы на наличие значений, минимальную длину и формат email.

```
// validationRules.js
export const required = (value) => (value ? '' : 'This field is required');

export const minLength = (length) => (value) =>
  value.length >= length ? '' : `Minimum length is ${length} characters`;

export const email = (value) =>
  /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(value) ? '' : 'Invalid email format';
```

Компонент

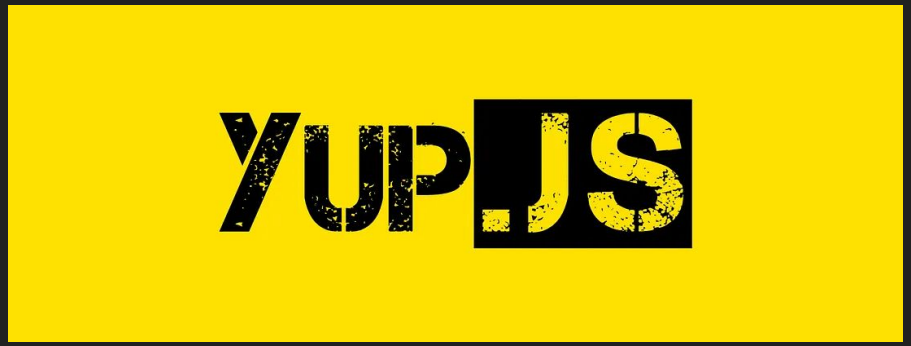
Использует хук useForm для управления формой.

При отправке формы вызывается валидация, и если форма валидна, она отправляется, иначе показываются ошибки.

```jsx
// MyForm.js
import React from 'react';
import { useForm } from './useForm';
import { required, minLength, email } from './validationRules';
const MyForm = () => {
  const { values, errors, handleChange, validateForm } = useForm(
    {
      name: '',
      email: '',
      password: '',
    },
    {
      name: required,
      email: (value) => required(value) || email(value),
      password: (value) => required(value) || minLength(6)(value),
    }
  );
  const handleSubmit = (e) => {
    e.preventDefault();
    if (validateForm()) {
      console.log('Form submitted successfully:', values);
    } else {
      console.log('Form has errors:', errors);
    }
  };
  return (
    <form onSubmit={handleSubmit}>
      <div>
        <label>Name:</label>
        <input
          type="text"
          name="name"
          value={values.name}
          onChange={handleChange}
        />
        {errors.name && <p>{errors.name}</p>}
      </div>
      <div>
        <label>Email:</label>
        <input
          type="email"
          name="email"
          value={values.email}
          onChange={handleChange}
        />
        {errors.email && <p>{errors.email}</p>}
      </div>
      <div>
        <label>Password:</label>
        <input
          type="password"
          name="password"
          value={values.password}
          onChange={handleChange}
        />
        {errors.password && <p>{errors.password}</p>}
      </div>
      <button type="submit">Submit</button>
    </form>
  );
```

YUP - валидация

Yup — это мощная и гибкая библиотека для валидации данных в JavaScript, которая особенно хорошо интегрируется с такими библиотеками, как Formik, react-hook-form, и многими другими. Основное назначение Yup — создание схем для валидации объектов, которые описывают правила проверки для каждого поля данных. Yup позволяет легко задавать сложные условия валидации, поддерживает кастомные сообщения об ошибках, а также предоставляет удобные методы для асинхронной валидации.



Основные особенности Yup:

- **Создание схем для валидации:** Используя цепочечные методы, вы можете описать структуру данных и правила валидации для каждого поля.
- **Поддержка вложенных объектов и массивов:** Позволяет валидировать сложные структуры данных, включая вложенные объекты и массивы.
- **Асинхронная валидация:** Поддерживает асинхронные проверки, что полезно для валидации, которая требует взаимодействия с сервером.
- **Кастомизация сообщений об ошибках:** Вы можете настраивать сообщения об ошибках для каждого поля или правила валидации.
- **Согласованность с библиотеками для работы с формами:** Yup часто используется вместе с Formik и react-hook-form для упрощения процесса валидации форм.

Как работает Yup:

Yup использует схемы для описания структуры данных, которую нужно валидировать. Эти схемы представляют собой набор правил, которые применяются к каждому полю объекта. Например, вы можете описать схему для объекта с полями name и age, где name — это строка, обязательная для заполнения, а age — число с минимальным значением.

В этом примере схема валидации содержит два поля:

- **name** — это строка, которая должна быть не короче 2 символов и обязательно для заполнения.
- **age** — это число, которое должно быть не меньше 18 лет и также обязательно для заполнения.

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

const validationSchema = Yup.object().shape({
  name: Yup.string()
    .required('Имя обязательно')
    .min(2, 'Имя должно быть не менее 2 символов'),
  age: Yup.number()
    .required('Возраст обязателен')
    .min(18, 'Возраст должен быть не менее 18 лет'),
});
```

Основные методы Yup:

required() — Проверяет, что поле не пустое.

min() — Определяет минимальное допустимое значение для чисел или длину для строк.

max() — Определяет максимальное допустимое значение для чисел или длину для строк.

email() — Проверяет, что значение является валидным email.

matches() — Проверяет, что строка соответствует регулярному выражению.

oneOf() — Проверяет, что значение находится в списке допустимых значений.

test() — Позволяет писать кастомные тесты для проверки значения.

Пример использования кастомного теста:

Здесь добавляется кастомный тест is-strong-password, который проверяет, что пароль содержит хотя бы одну цифру.

```
const validationSchema = Yup.object().shape({
  password: Yup.string()
    .required('Пароль обязателен')
    .min(8, 'Пароль должен быть не менее 8 символов')
    .test('is-strong-password', 'Пароль должен содержать хотя бы одну цифру', (value) =>
      /\d/.test(value)
    ),
});
```

Асинхронная валидация:

Yup поддерживает асинхронную валидацию, что позволяет, например, проверять уникальность email или имени пользователя на сервере.

```
const validationSchema = Yup.object().shape({
  username: Yup.string()
    .required('Имя пользователя обязательно')
    .test('check-unique-username', 'Имя пользователя уже занято', async (value) => {
      const isAvailable = await checkUsernameAvailability(value); // Функция запроса
      return isAvailable;
    }),
});
```

Поддержка вложенных объектов и массивов:

Yup также поддерживает валидацию сложных структур, таких как вложенные объекты и массивы.

```
const validationSchema = Yup.object().shape({
  friends: Yup.array().of(
    Yup.object().shape({
      name: Yup.string().required('Имя друга обязательно'),
    })
  ),
});
```

```
const validationSchema = Yup.object().shape({
  user: Yup.object().shape({
    firstName: Yup.string().required('Имя обязательно'),
    lastName: Yup.string().required('Фамилия обязательна'),
  }),
});
```

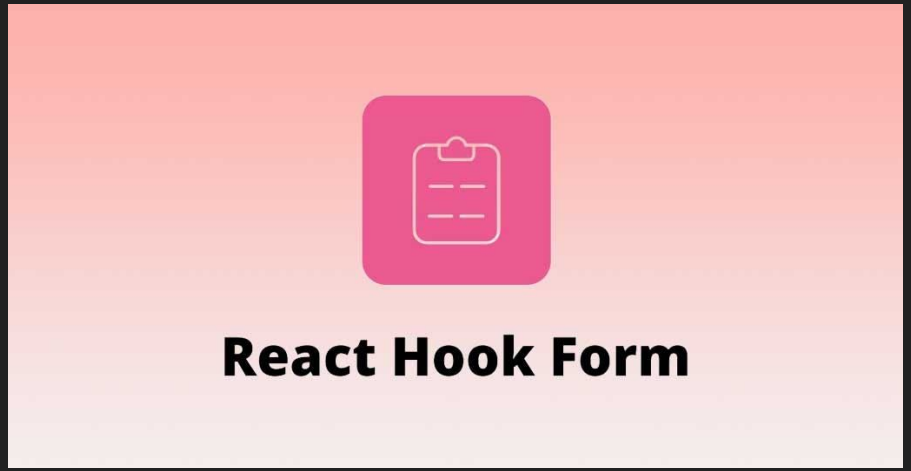
React-hook-form (неконтролируемые компоненты)

React Hook Form — это библиотека для управления состоянием форм в React, которая использует преимущества хуков для упрощения работы с формами. Она минимизирует количество перерисовок, обеспечивает простую интеграцию с валидаторами и поддерживает работу с контролируемыми и неконтролируемыми компонентами.

Основная цель React Hook Form — сделать работу с формами максимально простой и эффективной, предоставляя мощные инструменты для управления состоянием, валидации и обработки данных.

Основными концепциями библиотеки являются:

- **register:** используется для регистрации поля формы и подключения его к внутренним механизмам.
- **handleSubmit:** обрабатывает отправку формы и выполняет валидацию данных.
- **errors:** объект, содержащий ошибки валидации для каждого поля.
- **reset:** сбрасывает значения формы.
- **watch:** позволяет следить за изменениями полей формы.



Установка:
npm install react-hook-form
Для интеграции yup
npm install @hookform/resolvers

Объяснение:

useForm: Это основной хук библиотеки. Он возвращает методы, такие как register, handleSubmit, errors и т.д.

register: Метод для регистрации поля формы, который связывает его с внутренними механизмами библиотеки. В данном примере register("name", { required: true }) указывает, что поле "name" обязательно для заполнения.

handleSubmit: Этот метод используется для обработки отправки формы. Он проверяет валидацию и, если всё в порядке, передает данные формы в обработчик onSubmit.

errors: Объект, который содержит ошибки валидации для каждого поля формы. Если в поле есть ошибка, она отображается пользователю.

```jsx
import React from "react";
import { useForm } from "react-hook-form";

function SimpleForm() {
  const { register, handleSubmit, watch, formState: { errors } } = useForm();
  const onSubmit = (data) => {
    console.log(data); // Данные формы при успешной отправке
  };
  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <div>
        <label>Имя:</label>
        <input {...register("name", { required: true })} />
        {errors.name && <span>Поле имя обязательно!</span>}
      </div>
      <div>
        <label>Электронная почта:</label>
        <input {...register("email", {
          required: "Поле обязательно",
          pattern: {
            value: /^\S+@\S+$/i,
            message: "Неверный формат email"
          }
        })} />
        {errors.email && <span>{errors.email.message}</span>}
      </div>
      <input type="submit" />
    </form>
  );
}
```

yup + react-hook-form

Yup schema: Определяется схема валидации с помощью Yup, в которой прописаны правила для каждого поля.

yupResolver: Используется для интеграции схемы валидации Yup с react-hook-form.

Ошибки валидации: Если поле не проходит валидацию, errors содержит сообщение ошибки, которое можно вывести пользователю.

```jsx
import React from "react";
import { useForm } from "react-hook-form";
import { yupResolver } from "@hookform/resolvers/yup";
import * as Yup from "yup";
// Определение схемы валидации с помощью Yup
const schema = Yup.object().shape({
  name: Yup.string().required("Поле имя обязательно"),
  email: Yup.string().email("Неверный формат email").required("Поле email обязательно"),
});
function FormWithYupValidation() {
  const { register, handleSubmit, formState: { errors } } = useForm({
    resolver: yupResolver(schema) // Используем yupResolver для интеграции схемы валидации
  });
  const onSubmit = (data) => {
    console.log(data);
  };
  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <div>
        <label>Имя:</label>
        <input {...register("name")} />
        {errors.name && <span>{errors.name.message}</span>}
      </div>
      <div>
        <label>Электронная почта:</label>
        <input {...register("email")} />
        {errors.email && <span>{errors.email.message}</span>}
      </div>
      <input type="submit" />
    </form>
  );
}
```

Formik (контролируемые компоненты)

Formik — это одна из самых популярных библиотек для работы с формами в React. Она предназначена для упрощения процесса создания, управления и валидации форм. В отличие от react-hook-form, которая полагается на неконтролируемые компоненты, Formik чаще работает с контролируемыми компонентами, что даёт больше гибкости, но может повлиять на производительность при работе с большими формами.

Основные концепции в Formik:

- **Formik компонент:** Это компонент высшего порядка, который предоставляет методы и состояние формы через props, что делает его мощным и гибким для использования.
- **useFormik хук:** Альтернативный способ работы с формами через хуки, предоставляя вам полный контроль над формой.
- **Field компонент:** Компонент для рендеринга полей формы, который автоматически связывается с Formik.
- **Валидация**: Возможность валидации формы как с помощью встроенных функций, так и с использованием внешних библиотек, таких как Yup.

Установка:
npm install formik

Базовый пример

Formik компонент: Это контейнер для управления формой. Он предоставляет все необходимые методы для работы с формой, такие как значения полей, ошибки, статус отправки и т.д.

initialValues: Начальные значения для каждого поля формы.

validationSchema: Схема валидации на основе Yup, которая обеспечивает проверку данных перед отправкой формы.

onSubmit: Функция, которая вызывается при отправке формы. В ней обрабатываются данные, введённые пользователем.

Field: Компонент для рендеринга полей формы. Он автоматически связывается с состоянием формы и управляет значениями полей и событиями.

ErrorMessage: Компонент для отображения сообщений об ошибках для каждого конкретного поля формы.

```jsx
import React from "react";
import { Formik, Form, Field, ErrorMessage } from "formik";
import * as Yup from "yup";
function SimpleForm() {
  const validationSchema = Yup.object({
    name: Yup.string().required("Поле имя обязательно"),
    email: Yup.string().email("Неверный формат email").required("Поле email обязательно"),
  });
  return (
    <Formik
      initialValues={{ name: "", email: "" }} // Начальные значения
      validationSchema={validationSchema} // Схема валидации
      onSubmit={(values, { setSubmitting }) => {
        console.log(values);
        setSubmitting(false);
      }}
    >
      {({ isSubmitting }) => (
        <Form>
          <div>
            <label htmlFor="name">Имя:</label>
            <Field type="text" name="name" />
            <ErrorMessage name="name" component="div" />
          </div>

          <div>
            <label htmlFor="email">Электронная почта:</label>
            <Field type="email" name="email" />
            <ErrorMessage name="email" component="div" />
          </div>

          <button type="submit" disabled={isSubmitting}>Отправить</button>
        </Form>
      )}
    </Formik>
  );
}
export default SimpleForm;
```

Использование хука useFormik

useFormik: Хук, предоставляющий все необходимые методы и состояния для управления формой.

formik.handleSubmit: Метод для обработки отправки формы.

formik.handleChange: Метод для обработки изменений в полях формы.

formik.handleBlur: Метод для обработки события "потеря фокуса" (blur) для поля формы.

formik.values: Текущие значения полей формы.

formik.errors и formik.touched: Используются для отображения ошибок валидации только для тех полей, которые были изменены или затронуты пользователем.

```jsx
function SimpleFormWithHook() {
  const formik = useFormik({
    initialValues: { name: "", email: "" },
    validationSchema: Yup.object({
      name: Yup.string().required("Поле имя обязательно"),
      email: Yup.string().email("Неверный формат email").required("Поле email обязательно"),
    }),
    onSubmit: (values) => {
      console.log(values);
    },
  });
  return (
    <form onSubmit={formik.handleSubmit}>
      <div>
        <label>Имя:</label>
        <input
          type="text"
          name="name"
          onChange={formik.handleChange}
          onBlur={formik.handleBlur}
          value={formik.values.name}
        />
        {formik.touched.name && formik.errors.name ? (
          <div>{formik.errors.name}</div>
        ) : null}
      </div>
      <div>
        <label>Электронная почта:</label>
        <input
          type="email"
          name="email"
          onChange={formik.handleChange}
          onBlur={formik.handleBlur}
          value={formik.values.email}
        />
        {formik.touched.email && formik.errors.email ? (
          <div>{formik.errors.email}</div>
        ) : null}
      </div>
      <button type="submit">Отправить</button>
    </form>
  );
}
export default SimpleFormWithHook;
```

Redux-Form

Redux Form — это библиотека для управления формами в React с использованием Redux для хранения состояния формы. Она предоставляет возможность интеграции с глобальным состоянием приложения, что делает её особенно полезной, если вам нужно управлять состоянием форм централизованно, через Redux. Однако, в последние годы Redux Form потерял свою популярность по сравнению с более лёгкими решениями, такими как react-hook-form и Formik, так как он добавляет немалую нагрузку на производительность из-за постоянного обновления глобального состояния.

Установка:
npm install redux-form

Подготовка

```
// Функция валидации
const validate = (values) => {
  const errors = {};
  if (!values.name) {
    errors.name = 'Name is required';
  }
  if (!values.email) {
    errors.email = 'Email is required';
  } else if (!/\S+@\S+\.\S+/.test(values.email)) {
    errors.email = 'Invalid email address';
  }
  return errors;
};
```

```
// Компонент поля
const renderField = ({ input, label, type, meta: { touched, error } }) => (
  <div>
    <label>{label}</label>
    <div>
      <input {...input} placeholder={label} type={type} />
      {touched && (error && <span>{error}</span>)}
    </div>
  </div>
);

import { createStore, combineReducers } from 'redux';
import { reducer as formReducer } from 'redux-form';

const rootReducer = combineReducers({
  form: formReducer, // подключение редьюсера формы
  // другие редьюсеры
});

const store = createStore(rootReducer);
```

Компонент формы

Field: компонент, который связывает поле ввода с Redux Store. Он управляет значениями, ошибками и состоянием фокуса для каждого поля формы.

reduxForm: функция, которая оборачивает компонент формы и подключает его к Redux Store. Она добавляет метод handleSubmit и другие полезные функции.

validate: функция для валидации формы. Она получает текущие значения формы и возвращает объект с ошибками.

```
// Форма
const MyForm = (props) => {
  const { handleSubmit } = props;

  const onSubmit = (values) => {
    console.log('Form Values:', values);
  };

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <Field name="name" type="text" component={renderField} label="Name" />
      <Field name="email" type="email" component={renderField} label="Email" />
      <button type="submit">Submit</button>
    </form>
  );
};

// Оборачиваем компонент в reduxForm
export default reduxForm({
  form: 'myForm', // уникальное имя формы
  validate, // подключение валидации
})(MyForm);
```

Ресурсы

Код урока (github репозиторий) - ТЫК

Документация Yup - ТЫК

Документация React-hook-form - ТЫК

Документация Formik - ТЫК

Документация Redux-Form - ТЫК