Компоненты React. Работа с формами. Formik

React Router

```
<RootProvider>
  <div className="App" style={{ height: '100%' }}>
    <Router>
      <Routes>
        <Route path="/" element={<LoginPage />} />
        <Route path="/home" element={<UserPage />} />
        <Route path="/project" element={<ProjectPage />} />
      </Routes>
    </Router>
  </div>
</RootProvider>
```

```
const navigate = useNavigate()

…

navigate('/home')
```

Axios

```
export const axiosInstanse = axios.create({
    baseURL: 'url'
})
```

PropTypes

- optionalArray: PropTypes.array,
- optionalBool: PropTypes.bool,
- optionalFunc: PropTypes.func,
- optionalNumber: PropTypes.number,
- optionalObject: PropTypes.object,
- optionalString: PropTypes.string,
- optionalSymbol: PropTypes.symbol,
- optionalNode: PropTypes.node,
- optionalElement: PropTypes.element,
- optionalMessage: PropTypes.instanceOf(Message),
- optionalEnum: PropTypes.oneOf(['News', 'Photos']),
- optionalUnion: PropTypes.oneOfType([

  PropTypes.string,

  PropTypes.number,

  PropTypes.instanceOf(Message)

]),

- optionalObjectOf: PropTypes.objectOf(PropTypes.number),

- optionalObjectWithShape: PropTypes.shape({
  color: PropTypes.string,
  fontSize: PropTypes.number
}),

- optionalObjectWithStrictShape: PropTypes.exact({
  name: PropTypes.string,
  quantity: PropTypes.number
}),

- optionalArrayOf: PropTypes.arrayOf(PropTypes.number),

Кастомные валидаторы

```
customProp: function(props, propName, componentName) {
   if (!/matchme/.test(props[propName])) {
            return new Error(
                     'Проп ' + propName + ' компонента' +
                     + componentName + ' имеет неправильное значение!'
     );
   }
},
```

Значение пропов по умолчанию

Значение будет задано в случае отсутствия задания пропа

```
Component.defaultProps = {
    text: 'Кнопка'
}
```

Стилизация

- Inline-стилизация

```
<h1 style={{color: "red"}}>Hello Style!</h1>
```

- JS object

```
const mystyle = {
        color: "white",
        backgroundColor: "DodgerBlue",
        padding: "10px",
        fontFamily: "Arial"
};
…
<h1 style={mystyle}>Hello Style!</h1>
```

- JS-файлы

```
import {styles} from "./styles";
<p style = {styles.ErrorMessage}>Текст</p>
```

- CSS-файлы

```
import './App.css';
```

- CSS-модули

```
import styles from './mystyle.module.css';
<h1 className={styles.bigblue}>Hello Car!</h1>;
```

- Styled Components

```
import styled from 'styled-components';
const Button = styled.button
                `font-size: 1.5em;background-color: black;color: white;`;
const Input = styled.input` `
….
<Input placeholder="…" type="text" />
```

Библиотеки компонентов

- Material UI (https://material-ui.com)
- React-bootstrap (https://react-bootstrap.github.io/)
- Reactstrap (https://reactstrap.github.io/)
- Ant design (https://ant.design/)
- Semantic UI (https://react.semantic-ui.com/)
- Blueprint (https://blueprintjs.com/)
- … и многие другие

React-bootsrap

- Установка

npm install react-bootstrap bootstrap

- Использование компонентов

```
import { Button } from 'react-bootstrap’;
…
<Button variant="primary" onClick=“alert(‘Нажатие на
        кнопку’)">Primary</Button>
```

- Стилизация

```
<Button className="remove-btn“>Primary</Button>
<Button style={{marginLeft: 2%}}>Primary</Button>
```

React-bootstrap-table2

```
import BootstrapTable from 'react-bootstrap-table-next';
import 'bootstrap/dist/css/bootstrap.min.css'
import 'react-bootstrap-table-next/dist/react-bootstrap-table2.min.css';
import filterFactory, { textFilter } from 'react-bootstrap-table2-filter';
…
const columns = [{
  dataField: 'id',
  text: 'Product ID',
}, {
  dataField: 'labels',
  text: 'Labels',
  formatter: (cell) => {
    return <>{cell.map(label => <li>{label}</li>)}</>
  },
}];
…
<BootstrapTable keyField='id' data={ products } columns={ columns } filter={ filterFactory() }
striped hover />
```

Описание столбца

```
{
  dataField: ‘id',
  text: ‘ID’,
  sort: true,
  filter: textFilter(),
  footer: columnData => columnData.reduce((acc, item) => acc + item, 0)
  formatter: (cell) => {
    return <>{cell.map(label => <li>{label}</li>)}</>
  },
  sortFunc: (a, b, order, dataField, rowA, rowB) => {
    if (order === 'asc') return a - b;
    else return b - a;
  },
  filterRenderer: (onFilter, column) => .....
},
```

Виды фильтров

- TextFilter - textFilter()
- SelectFilter - selectFilter({options: selectOptions})
- MultiSelectFilter - multiSelectFilter({options: selectOptions})
- NumberFilter – numberFilter()
- DateFilter – dateFilter()
- CustomFilter

```
customFilter({
  type: FILTER_TYPES.NUMBER,  // default is FILTER_TYPES.TEXT
  comparator: Comparator.EQ, // only work if type is FILTER_TYPES.SELECT
  caseSensitive: false, // default is true
})
```

Выбор строк

```
const selectRow = {
  mode: 'checkbox',
  clickToSelect: true,
  onSelect: (row, isSelect, rowIndex, e) => {…},
  style: (row, rowIndex) => { return ...; },
  nonSelectable: [1, 3 ,5],
  hideSelectColumn: true,
  nonSelectableClasses: (row, rowIndex) => { return ...; }
};

<BootstrapTable ... selectRow={ selectRowProp } />
```

Редактирование ячейки

```
import cellEditFactory from 'react-bootstrap-table2-editor';
import { Type } from 'react-bootstrap-table2-editor';
const columns = [
  ..., {
    …
    editor: {
      type: Type.SELECT | Type.TEXTAREA | Type.CHECKBOX | Type.DATE,
      getOptions: (setOptions) => {
                    setTimeout(() => setOptions([...]), 1500);    }
]
…
<BootstrapTable … cellEdit={ cellEditFactory({ mode: 'click',
    onStartEdit: (row, column, rowIndex, columnIndex) => { console.log('start to edit!!!'); },
    beforeSaveCell: (oldValue, newValue, row, column) => { console.log('Before Saving Cell!!'); },
    afterSaveCell: (oldValue, newValue, row, column) => { console.log('After Saving Cell!!'); }
    blurToSave: true})
} />
```

Расширяемые строки

```
const expandRow = {
  renderer: row => (
    <div>....</div>
  )
};

<BootstrapTable
  keyField='id'
  data={ products }
  columns={ columns }
  expandRow={ expandRow }
/>
```

Yup

- Создание схемы для всех типов данных

let schema = yup.mixed();

- mixed.clone(): Schema
- mixed.concat(schema: Schema): Schema
- mixed.validate(value: any, options?: object): Promise<any, ValidationError>
- mixed.validateSync(value: any, options?: object): any
- mixed.isValid(value: any, options?: object): Promise<boolean>
- mixed.isValidSync(value: any, options?: object): boolean
- mixed.isType(value: any): boolean
- mixed.strict(isStrict: boolean = false): Schema

- mixed.default(value: any): Schema
- mixed.default(): Any
- mixed.nullable(isNullable: boolean = true): Schema
- mixed.required(message?: string | function): Schema
- mixed.notRequired(): Schema
- mixed.defined(): Schema
- mixed.oneOf(arrayOfValues: Array<any>, message?: string | function):

Schema Alias: equals

- mixed.notOneOf(arrayOfValues: Array<any>, message?: string | function)
- mixed.test(name: string, message: string | function, test: function): Schema
- mixed.transform((currentValue: any, originalValue: any) => any): Schema

Yup.string

- string.required(message?: string | function): Schema
- string.length(limit: number | Ref, message?: string | function): Schema
- string.min(limit: number | Ref, message?: string | function): Schema
- string.max(limit: number | Ref, message?: string | function): Schema
- string.matches(regex: Regex, message?: string | function): Schema
- string.matches(regex: Regex, options: { message: string, excludeEmptyString: bool }): Schema
- string.email(message?: string | function): Schema
- string.url(message?: string | function): Schema
- string.uuid(message?: string | function): Schema
- string.ensure(): Schema
- string.trim(message?: string | function): Schema
- string.lowercase(message?: string | function): Schema
- string.uppercase(message?: string | function): Schema

Yup.number

- number.min(limit: number | Ref, message?: string | function): Schema
- number.max(limit: number | Ref, message?: string | function): Schema
- number.lessThan(max: number | Ref, message?: string | function): Schema
- number.moreThan(min: number | Ref, message?: string | function): Schema
- number.positive(message?: string | function): Schema
- number.negative(message?: string | function): Schema
- number.integer(message?: string | function): Schema
- number.truncate(): Schema
- number.round(type: 'floor' | 'ceil' | 'trunc' | 'round' = 'round'): Schema

Yup.date

- date.min(limit: Date | string | Ref, message?: string | function): Schema
- date.max(limit: Date | string | Ref, message?: string | function): Schema

Yup.array

- array.of(type: Schema): Schema
- array.required(message?: string | function): Schema
- array.min(limit: number | Ref, message?: string | function): Schema
- array.max(limit: number | Ref, message?: string | function): Schema
- array.ensure(): Schema
- array.compact(rejector: (value) => boolean): Schema

Yup.object

- object.shape(fields: object, noSortEdges?: Array<[string, string]>): Schema
- object.from(fromKey: string, toKey: string, alias: boolean = false): Schema
- object.noUnknown(onlyKnownKeys: boolean = true, message?: string |

function): Schema

- object.camelCase(): Schema
- object.constantCase(): Schema

Методы для работы со схемой

- yup.reach(schema: Schema, path: string, value?: object, context?: object): Schema
- yup.addMethod(schemaType: Schema, name: string, method: ()=> Schema): void
- yup.ref(path: string, options: { contextPrefix: string }): Ref
- yup.lazy ((value: any) => Schema): Lazy

Формы

```
const [validated, setValidated] = useState(false);
const handleSubmit = (event) => {
const form = event.currentTarget;
if (form.checkValidity() === false) {
  event.preventDefault();
  event.stopPropagation();
}
setValidated(true);
};
……
<Form noValidate validated={validated} onSubmit={handleSubmit}>
  <Form.Group as={Row} controlId="formHorizontalEmail">
    <Form.Label column sm={2}>Email</Form.Label>
    <Col sm={10}><Form.Control type="email" placeholder="Email" /></Col>
  </Form.Group>
  ….
  <Form.Group as={Row}><Col sm={{ span: 10, offset: 2 }}><Button type="submit">Sign in</Button></Col></Form.Group>
</Form>
```

Formik

```
<Formik onSubmit={(values) => this.login(values)} initialValues={{name: '', password: ''}}>
{({ handleSubmit, handleChange, handleBlur, values, touched, isValid, errors, }) => (
<Form noValidate onSubmit={handleSubmit}>
<Form.Group as={Row} controlId="userName">
                <Form.Label column md={4}>Имя пользователя</Form.Label>
                <Col md={8}>
                                <Form.Control
                                                type="text" placeholder="Имя" name="name" value={values.name}
                                                                onChange={handleChange} />
                </Col>
</Form.Group>

<Form.Group as={Row}>
                <Col sm={{ span: 10, offset: 5 }}><Button size="lg" type="submit">Войти</Button></Col>
</Form.Group>
</Form>
)}
</Formik>
```

Валидация

```
const SignupSchema = Yup.object().shape({
    firstName: Yup.string()
      .min(2, 'Too Short!')
      .max(50, 'Too Long!')
      .required('Required'),
    email: Yup.string().email('Invalid email').required('Required'),
});
...
<Formik initialValues={{...}} validationSchema={SignupSchema} onSubmit={values => {console.log(values);}} >
    {({ handleSubmit, handleChange, handleBlur, values, touched, isValid, errors, }) => (
        <Form noValidate>
          <Form.Group as={Row} controlId="firstName">
            <Form.Label column md={4}>Имя</Form.Label>
            <Col md={8}>
                <Form.Control
                      type="text" placeholder="Имя" name="firstName" value={values.firstName} onChange={handleChange}
                          isInvalid={!!(errors.firstName && touched.firstName)} />
                 <Form.Control.Feedback type="invalid">{errors.firstName}</Form.Control.Feedback>
            </Col>
          </Form.Group>
....
        </Form>
}}
</Formik>
```

Валидация выполняется

1. После изменения данных
   - handleChange
   - setFieldValue
   - setValues
2. После изменения фокуса
   - handleBlur
   - setTouched
   - setFieldTouched
3. При отправке формы
   - handleSubmit
   - submitForm
4. При вызове специальных методов
   - validateForm
   - validateField

Хуки Formik

- Создание собственного поля:

```
const [field, meta, helpers] = useField(props);
```

- Получение состояния формы

```
const formik = useFormik(object)
```

- Получение состояния формы через контекст

```
const { values, submitForm } = useFormikContext();
```