

A decorative wavy line in orange and white, running vertically along the left side of the slide.

React

Forms & Validations

Why use **Formik**?

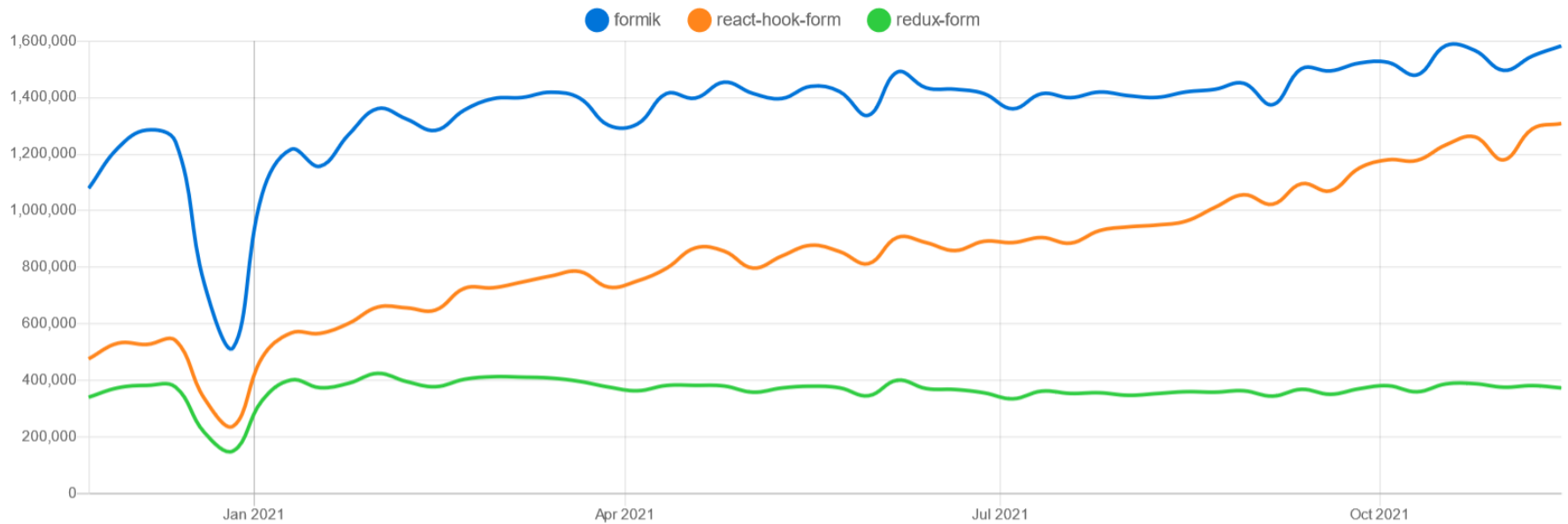
- Forms are really verbose in [React](#). To make matters worse, most form helpers do way too much magic and often have a significant performance cost associated with them.
- Formik is a small library that helps you with the 3 most annoying parts:
 - Getting values in and out of form state
 - Validation and error messages (yup)
 - Handling form submission
- Website: <https://formik.org/>
- To install formik package, type: `npm i formik`

Why use Yup?

- Yup makes our lives far easier for validating the data our apps consume and manipulate, in an abstract way that does not interfere with the rest of our business logic.
- Website: <https://github.com/jquense/yup>
- Api: <https://github.com/jquense/yup#api>
- To install Yup, type: `npm i yup`

- npm trends: <https://www.npmtrends.com/formik-vs-react-hook-form-vs-redux-form>

Downloads in past 1 Year ▾



Applying Formik & Yup

```
import { useFormik } from "formik";  
import * as Yup from "yup";
```

```
const Major = () => {  
  const [majors, setMajors] = useState<IMajor[]>([]);  
  const [showModal, setShowModal] = useState(false);  
  
  const handleCloseModal = () => setShowModal(false);  
  const handleShowModal = () => setShowModal(true);  
  
  const frm = useFormik({  
    initialValues: { id: 0, name: "", },  
    validationSchema: Yup.object({  
      id: Yup.number().required(),  
      name: Yup.string().required("Required").min(3, ">= 3 characters"),  
    }),  
    onSubmit: (values) => {  
      handleSave(values);  
    },  
  });
```

```
const handleSave = (data: IMajor) => {  
  if (data.id === 0) {  
    majorService.add(data).then((res) => {  
      if (res.errorCode === 0) {  
        loadData();  
        handleCloseModal();  
        toast.success("Add successfully");  
      } else toast.error(res.message);  
    });  
  } else {  
    majorService.update(data.id, data).then((res) => {  
      if (res.errorCode === 0) {  
        loadData();  
        handleCloseModal();  
        toast.success("Update successfully");  
      } else toast.error(res.message);  
    });  
  }  
};
```

```
const showEditModal = (e: any, id: number) => {  
  if (e) e.preventDefault();  
  if (id > 0) {  
    majorService.get(id).then((res) => {  
      frm.setValues(res.data);  
      handleShowModal();  
    });  
  } else {  
    frm.resetForm();  
    handleShowModal();  
  }  
};
```

Using Formik in Modal

```
<Modal show={showModal} onHide={handleCloseModal}
  size="lg" backdrop="static" keyboard={false}>
  <Modal.Header closeButton>
    <Modal.Title>{frm.values.id > 0 ? "Update" : "New"} Major</Modal.Title>
  </Modal.Header>
  <Modal.Body>
    <div className="row">
      <Input label="Major name" type="text" id="txtMajor"
        frmField={frm.getFieldProps("name")}
        errMessage={frm.errors.name} />
    </div>
  </Modal.Body>
  <Modal.Footer>
    <Button variant="secondary" onClick={handleCloseModal}>Close</Button>
    <Button variant="primary"
      disabled={!frm.dirty || !frm.isValid}
      onClick={() => frm.handleSubmit()}>
      Save
    </Button>
  </Modal.Footer>
</Modal>
```


Modifying **Input.tsx** component

```
type InputAttributes = InputHTMLAttributes<HTMLInputElement>;

interface Props extends DetailedHTMLProps<InputAttributes, HTMLInputElement> {
  id: string;
  label: string;
  labelSize?: number;
  rows?: number;
  inputRef?: any;
  required?: boolean;
  lastRow?: boolean;
  frmField?: object;
  errorMessage?: string | undefined;
}

> const Input: FC<Props> = ({ ...
  });

export default Input;
```

```
> interface Props extends DetailedHTMLProps<InputAttributes, HTMLInputElement> { ...
}


const Input: FC<Props> = ({
  inputRef,
  id,
  label,
  labelSize = 3,
  rows = 1,
  className = "",
  required = false,
  lastRow = false,
  frmField,
  errMessage,
  ...others
}) => {
  const labelClass = `col-sm-${labelSize} col-form-label ${required ? "required" : ""}`;
  const inputClass = `form-control ${className} ${errMessage ? "is-invalid" : ""}`;
  return (
```

```
const labelClass = `col-sm-${labelSize} col-form-label ${required ? "required" : ""}`;
const inputClass = `form-control ${className} ${errorMessage ? "is-invalid" : ""}`;
return (
  <div className={`row ${lastRow ? "" : "mb-3"}`>
    <label htmlFor={id} className={labelClass}>{label}</label>
    <div className="col-sm">
      {rows > 1 ? (
        <textarea id={id} ref={inputRef} rows={rows}
          {...(others as TextareaHTMLAttributes<HTMLTextAreaElement>)}
          className={inputClass}
          {...frmField}
        ></textarea>
      ) : (
        <input id={id} ref={inputRef} {...others} className={inputClass} {...frmField} />
      )}
      {errorMessage ? <div className="invalid-feedback">{errorMessage}</div> : ""}
    </div>
  </div>
);
```

Testing Form

Student Management Major Student welcome to ...

New Major



Major Name 



>= 5 characters



Close Save



#	Major
1	IT
2	Ma
3	Networking
4	Business Administration

+ Add

Exercise

- Validate instructor component before submit.

Student Management - Main - Instructors - Student

New Instructor

Instructor Id * Student Id

Full name * Last name First name

Gender * ☐ Male ☐ Female

Phone * Phone number

Email Email address

Close Save

#	Instr
1	
2	
3	
4	

Add

✎ ✖

✎ ✖

✎ ✖

✎ ✖



THE END