React Hooks & Form

Hook, React-Bootstrap, Formik & Yup, react-toastify

What are Hooks?

- Hooks are a new addition to React in version 16.8 that allows you use state and other React features, like lifecycle methods, without writing a class.
- Classes in JavaScript encourage multiple levels of inheritance that quickly increase overall complexity and potential for errors.
- The benefits of hooks are:
  - How we managing state has become easier to reason about
  - Our code is significantly simplified, and more readable
  - It’s easier to extract and share stateful logic in our apps

Convert a React Class to React Hook

10 steps to quickly change a class component to a functional component with hooks:

- Change class to function

```
class Major extends Component {
```

→

```
const Major = (props) => {
```

- Remove the constructor
- Remove the render() method, keep the return
- Add const before all methods

- Set initial state with useState()

```
import React, { useState, Fragment } from "react";
```

```
state = {
  majors: [],
};
```

```
const [majors, setMajors] = useState([]);
```

- Remove this.state throughout the component

```
{this.state.majors.map((major, idx)
```

```
{majors.map((major, idx) =>
```

- Remove all references to 'this' throughout the component
- Change this.setState() … instead, call the function that you named in the previous step to update the state…

```
this.setState({ majors: res.data });
```

```
setMajors(res.data);
```

- Replace compentDidMount with useEffect

```
import React, { useState, Fragment, useEffect } from "react";
```

```
componentDidMount() {
  majorService.getAll().then((res) => {
    if (res.errorCode === 0) {
      setMajors(res.data);
    }
  });
}
```

```
useEffect(() => {
  majorService.getAll().then((res) => {
    if (res.errorCode === 0) {
      setMajors(res.data);
    }
  });
 }, []);
```

- Replace componentDidUpdate with useEffect

```
const [id, setId] = useState(1);
useEffect(() => {
  if(id === '' || id === null) return;
  majorService.getAll().then((res) => {
      if (res.errorCode === 0) {
        setMajors(res.data);
      }
  });
 }, [id]);
```

Integrating “react-bootstrap”

- Website:

  https://react-bootstrap.github.io/

- Installing package

  npm i react-bootstrap

- Adding stylesheet in “src/index.js” file

  import 'bootstrap/dist/css/bootstrap.min.css’;

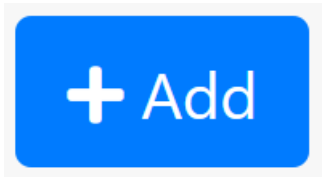
Create Modal using “react-bootstrap”

```
import { Button, Modal } from "react-bootstrap";
```

```
const Major = (props) => {
  const [majors, setMajors] = useState([]);
  const [modalShow, setModalShow] = useState(false);

  const handleModalClose = () => setModalShow(false);
  const handleModalShow = (e, dataId) => setModalShow(false);
```

```
<Modal show={modalShow} onHide={handleModalClose} backdrop="static" keyboard={false}>
  <Modal.Header closeButton>
    <Modal.Title>Major</Modal.Title>
  </Modal.Header>
  <form>
    <Modal.Body>
      <Input id="txtMajor" type="text" label="Major" maxLength="100" />
    </Modal.Body>
    <Modal.Footer>
      <Button variant="secondary" onClick={handleModalClose}>
        Close
      </Button>
      <Button variant="primary" type="submit">
        Save
      </Button>
    </Modal.Footer>
  </form>
</Modal>
```




```
<button
  type="button"
  className="btn btn-primary"
  data-toggle="modal"
  onClick={() => handleModalShow(null, 0)}
>
  <i className="fas fa-plus"></i> Add
</button>
```

```
<a
  href="/#" className="mr-1"
  onClick={(e) => handleModalShow(e, major.id)}
>
  <i className="fas fa-edit text-primary"></i>
</a>
```

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
  - Handling form submission
- Website: https://formik.org/
- To install formic 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.
- Reference: https://www.npmjs.com/package/yup
- To install Yup, type: npm i yup

Applying Formik & Yup in Component

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

```
const [majors, setMajors] = useState([]);

const formik = useFormik({
  initialValues: {
    name: "",
  },
  validationSchema: Yup.object({
    name: Yup.string().required("Required").min(5, "Must be 5 characters or more"),
  }),
  onSubmit: (values) => {
    console.log(values);
  },
});
```

Using Formik in Form

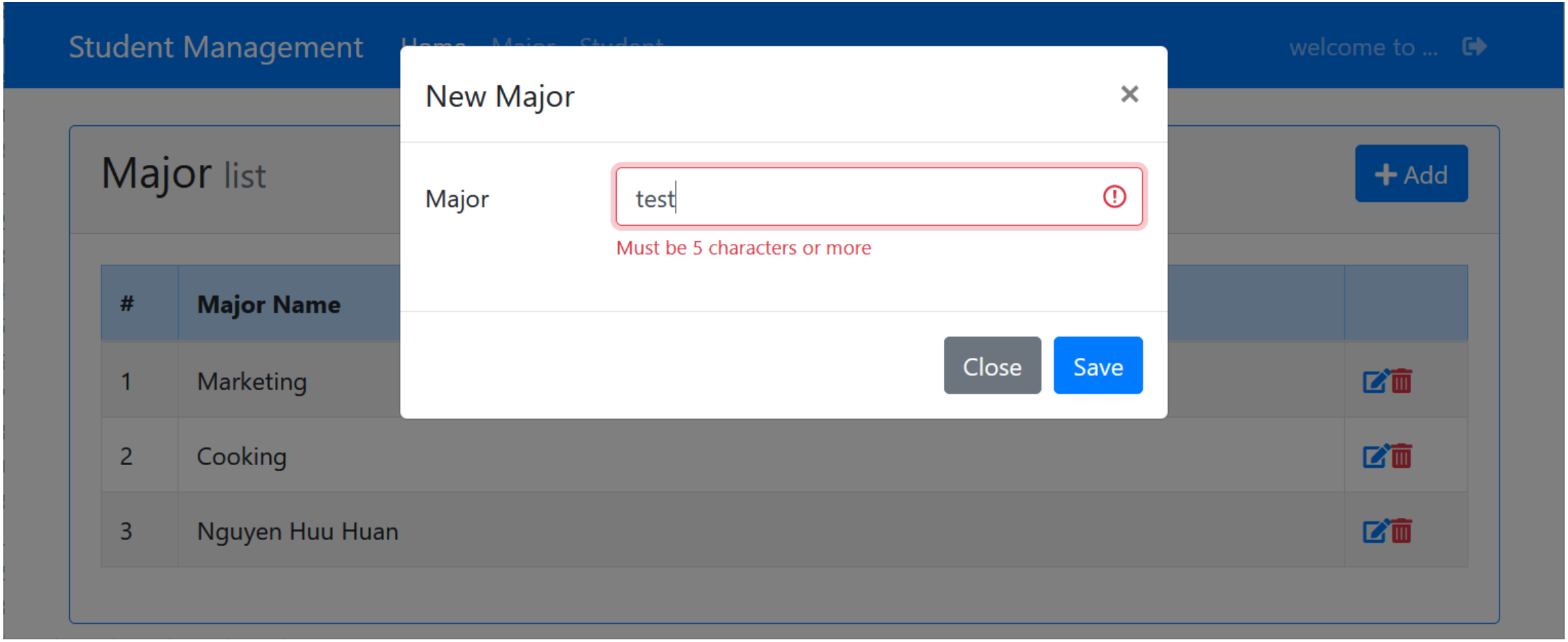
```
<form onSubmit={formik.handleSubmit}>
  <Modal.Body>
    <Input id="txtMajor" type="text" label="Major" maxLength="100"
      frmField={formik.getFieldProps("name")}
      err={formik.touched.name && formik.errors.name}
      errMessage={formik.errors.name}
    />
  </Modal.Body>
  <Modal.Footer>
    <Button variant="secondary" onClick={handleModalClose}>
      Close
    </Button>
    <Button variant="primary" type="submit">
      Save
    </Button>
  </Modal.Footer>
</form>
```

Modifying “Input.jsx” control

```
render() {
  const { inputRef, id, label, labelSize, frmField, err, errMessage, ...others } = this.props;
  const size = labelSize ? labelSize : 3;
  const classLeft = `col-sm-${size} col-form-label`;
  const classRight = `col-sm-${12 - size}`;
  const inputClass = `form-control ${err ? "is-invalid" : ""}`;

  return (
    <div className="form-group row">
      <label htmlFor={id} className={classLeft}>{label}</label>
      <div className={classRight}>
        {others["rows"] > 1 ? (
          <textarea ref={inputRef} id={id} className={inputClass} {...others} {...frmField}></textarea>
        ) : (
          <input ref={inputRef} id={id} className={inputClass} {...others} {...frmField} />
        )}
        {err ? <div className="invalid-feedback">{errMessage}</div> : null}
      </div>
    </div>
  );
}
```

Testing Form



Preparing Api Services in “majorService.js”

```
import api from './api';

const getAll = () => api.get(api.url.majors).then(res => res.data);
const get = (id) => api.get(`${api.url.majors}/${id}`).then(res => res.data);
const add = (data) => api.post(api.url.majors, data).then(res => res.data);
const update = (id, data) => api.put(`${api.url.majors}/${id}`, data).then(res => res.data);
const remove = (id) => api.delete(`${api.url.majors}/${id}`).then(res => res.data);

const majorService = { getAll, get, add, update, delete: remove };

export default majorService;
```

Creating “LoadData” function

```
const loadData = () => {
  majorService.getAll().then((res) => {
    if (res.errorCode === 0) {
      setMajors(res.data);
    }
  });
};
useEffect(() => {
  loadData();
}, []);

return (
```

Adding a new Major

```
const handleFormSubmit = (data) => {
  //TODO: check for ADDING or UPDATING
  majorService.add(data).then((res) => {
    if (res.errorCode === 0) {
      //TODO: show success message
      loadData();
      handleModalClose();
    } else {
      //TODO: show error message
    }
  });
};
```

```
const formik = useFormik({
  initialValues: {
    name: "",
  },
  // enableReinitialize: true,
  validationSchema: Yup.object({
    name: Yup.string()
              .required("Required")
              .min(5, "Must be 5 characters or more"),
  }),
  onSubmit: (values) => {
    handleFormSubmit(values);
  },
});
```

Test “Add” function

Deleting a Major

```
const deleteRow = (e, dataId) => {
  e.preventDefault();
  //TODO: show confirm dialogue
  majorService.delete(dataId).then((res) => {
    if (res.errorCode === 0) {
      //TODO: show success message
      loadData();
    } else {
      //TODO: show error message
    }
  });
};
```

```
<a href="/#" onClick={(e) => deleteRow(e, major.id)}>
  <i className="fas fa-trash-alt text-danger"></i>
</a>
```

Test “Delete” function

Updating an existing Major

```
const [majorId, setMajorId] = useState(0);
```

```
const handleModalShow = (e, dataId) => {
  if (e) e.preventDefault();
  setMajorId(dataId);
  if (dataId > 0) { // edit
    majorService.get(dataId).then((res) => {
      formik.setValues(res.data);
      setModalShow(true);
    });
  } else { // add
    formik.resetForm();
    setModalShow(true);
  }
};
```

```
const handleFormSubmit = (data) => {
  if (majorId === 0) { // add
    majorService.add(data).then((res) => {…
  } else { // update
    majorService.update(majorId, data).then((res) => {
      if (res.errorCode === 0) {
        //TODO: show success message
        loadData();
        handleModalClose();
      } else {
        //TODO: show error message
      }
    });
  }
};
```

Test “Update” function

Showing Notification

- In terminal, type: npm i react-toastify
- In “index.js” file, import & configure “react-toastify”

```
import { toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';

toast.configure({
  autoClose: 3000,
  draggable: false,
  position: 'top-right',
  hideProgressBar: false,
  newestOnTop: true,
  closeOnClick: true,
  rtl: false,
  pauseOnVisibilityChange: true,
  pauseOnHover: true
});
```

Applying “react-toastify” in “major.jsx”

- Modify “delete” function

```
const deleteRow = (e, dataId) => {
  e.preventDefault();
  //TODO: show confirm dialogue
  majorService.delete(dataId).then((res) => {
    if (res.errorCode === 0) {
      toast.warning("A data has been deleted!");
      loadData();
    } else {
      toast.error("Delete failed!");
    }
  });
};
```