Validations

With any web application, we would like to make sure the data being inserted into our database is clean. We do not want erroneous or incomplete data to make its way into our database. In order to prevent this, we can validate the data beforehand.

We will be performing our validations on the backend. This will be accomplished by making a request to our API. If the response contains an error, then we will render the errors on our frontend. For example, let's say we have an API that creates books. The `BookSchema` looks like the following:

```
const mongoose = require('mongoose');
const BookSchema = new mongoose.Schema({
    title: {
        type: String,
        required: [
            true,
            "Title is required"
        ]
    },
    numberOfPages: {
        type: Number,
        required: [
            true,
            "Pages is required"
        ]
    }
}, { timestamps: true });
module.exports = mongoose.model('Book', BookSchema);
```

We have a title, number of pages and a timestamp. Both the title and number of pages is required.

Our controller is using a different syntax to export our function as a key (`create:`) in an object instead of exporting each individual function independently as we have done before. You can write as many keys and functions as you would like. The controller for our Books will look like this:

```
const Book = require('../models/book.model');
module.exports = {
    create: (request, response) => {
        const { title, pages } = request.body;
        Book.create({
            title: title,
            numberOfPages: pages
        })
            .then(book => response.json(book))
            .catch(err => response.status(400).json(err))
    }
}
```

We are making this a 400 response so that our `axios.post` request will catch it as an error. If we make a request **without** the `title` or `numberOfPages`, we will see a response like this:

```
{
    "errors": {
        "title": {
            "message": "Title is required",
            "name": "ValidatorError",
            "properties": {
                "message": "Title is required",
                "type": "required",
                "path": "title"
            },
            "kind": "required",
            "path": "title"
        },
        "numberOfPages": {
            "message": "Pages is required",
            "name": "ValidatorError",
            "properties": {
                "message": "Pages is required",
                "type": "required",
                "path": "numberOfPages"
            },
            "kind": "required",
            "path": "numberOfPages"
        }
    },
    "_message": "Book validation failed",
    "message": "Book validation failed: numberOfPages: Pages is required, title: Title is
required",
    "name": "ValidationError"
}
```

The response we have contains all of the information we need to let the client know that the request has failed. When we make a request in our React project, we can check if there are errors in the response and conditionally render other information.

So, our React component will need to understand the structure of this response and then prepare to display them. Let's see what our component will look like:

MERN PART-TIME ONLINE

Full Stack MERN

Full Stack MERN

Advanced MERN

- Overview ✓
- Reusing Components ✓
- Reusing Components (Part II) ✓
- Presentational vs Container ✓
- Refactor, Reuse, Recycle ✓
- Material-UI
- **Validations**
- Authors
- Team Manager
- Belt Exam Rubric ✓
- Sports Demo

```
import React, { useState } from 'react';
import axios from 'axios';
const Main = () {
    const [title, setTitle] = useState("");
    const [pages, setPages] = useState(0);
    //Create an array to store errors from the API
    const [errors, setErrors] = useState([]);
    const onSubmitHandler = e => {
        e.preventDefault();
        //Send a post request to our API to create a Book
        axios.post('http://localhost:8000/api/books', {
            title,
            pages
        })
            .then(res=>console.log(res)) // If successful, do something with the response.
            .catch(err=>{
                const errorResponse = err.response.data.errors; // Get the errors from
err.response.data
                const errorArr = []; // Define a temp error array to push the messages in
                for (const key of Object.keys(errorResponse)) { // Loop through all errors
and get the messages
                    errorArr.push(errorResponse[key].message)
                }
                // Set Errors
                setErrors(errorArr);
            })
    }
    return (
            <form onSubmit={onSubmitHandler}>
                {errors.map((err, index) => (
                    <p key="{index}">{err}</p>
                ))}
                <p>
                    <label>Title</label>
                    <input type="text" onChange={e => setTitle(e.target.value)} />
                </p>
                <p>
                    <label>Pages</label>
                    <input type="text" onChange={e => setPages(e.target.value)} />
                </p>
                <input type="submit" />
            </form>
        </div>
    )
}
export default Main;
```

We are going through our response to find all the errors if our response sends back a 400 response. If there are errors, we will render all of them via the `map` method above the form inputs.

An alternate way of presenting the errors is to put each error next to its corresponding input.

This would be done using the following approach:

```
import React, { useState } from 'react';
import axios from 'axios';
const Main = () {
    const [title, setTitle] = useState("");
```