Step 1 : For validation, install these packages



Step 2 : create the form component

Entity class

"title": "Fjallraven - Foldsack No. 1 Backpack, Fits 15 Laptops",

"price": 109.95,

"description": "Your perfect pack for everyday use and",

"category": "men's clothing",

"image":"https://fakestoreapi.com/img/81fPKd-2AYL.\_AC\_SL1500\_.jpg",

 <>

      <form onSubmit={handleSubmit}>

        <div>

          <label htmlFor="title">Product Title</label>

          <input

            type="text"

            name="title"

            id="title"/>

        </div>

        <div>

          <label htmlFor="price">Product Price</label>

          <input

            type="text"

            name="price"

            id="price"/>

        </div>

        <div>

          <label htmlFor="description">Product Description</label>

          <input

            type="text"

            name="description"

            id="description"/>

        </div>

        <div>

          <label htmlFor="category">Product Category</label>

          <input

            type="text"

            name="category"

            id="category"/>

        </div>

        <div>

          <label htmlFor="image">Product Image</label>

          <input

            type="text"

            name="image"

            id="image"/>

        </div>

        <div className="button-group">

          <button type="submit">Register</button>

          <button type="button">Cancel</button>

        </div>

    </form>

    </>

Step 3 : initialise all the form data

const AddProduct = () => {

  const p1 = {

    title: "",

    price: 0,

    description: "",

    category: "",

    image: "",

  };

Step 4 : make a Validation.js file (NOTE its a .js file not a .jsx as it is not a component)

Validation.js

import \* as yup from "yup";

export const registartionValidation = yup.object({

  title: yup

    .string()

    .matches(/^[a-zA-Z0-9 !@#$%^&\*(),.?":{}|<>\_\-+=]\*$/, "Enter a valid title with special characters and digits allowed" )

    .min(3, "Title must be at least 3 characters")

    .max(200, "Title must be at most 200 characters")

    .required("Title is required"),

    price: yup

    .number()

    .typeError("Price must be a number")

    .min(1, "Min Price must be greater than 0")

    .max(100000, "Max Price must be less than or equal to 100000")

    .required("Price is Mandatory"),

  description: yup

    .string()

    .matches(/^[a-zA-Z0-9 '!@#$%^&\*(),.?":{}|<>\_\-+=]\*$/, "Enter a valid description" ),

  category: yup

    .string()

    .matches(/^[a-zA-Z0-9 '!@#$%^&\*(),.?":{}|<>\_\-+=]\*$/, "Enter a valid category" )

    .required("Category is Mandatory"),

  image: yup

    .string()

    .url("Enter a valid URL")

    .matches(

      /\.(jpeg|jpg|png|gif|bmp|svg)$/i,

      "Enter a valid image URL (jpeg, jpg, png, gif, bmp, svg)"

    )

    .required("Image URL is Mandatory"),

});

Step 5: write in North hemisphere (done in step 3)

import React, { useState } from 'react';

import './AddProduct.css';

import { useFormik } from 'formik';

import { registartionValidation } from '../components/Validation';

const AddProduct = () => {

  const p1 = {

    title: "",

    price: 0,

    description: "",

    category: "",

    image: "",

  };

  //const [product, setProduct] = useState(p1);

  const {

    errors,

    values, // Changed 'value' to 'values'

    touched,

    handleBlur,

    handleSubmit,

    handleChange,

  } = useFormik({

    initialValues: p1,  // Correctly set initialValues

    validationSchema: registartionValidation,

    onSubmit: function () {

      saveData();

    },

  });

async function saveData() {

    try {

      const response = await fetch('http://localhost:8087/products/addProduct', {

        method: 'POST',

        headers: {

          'Content-Type': 'application/json',

        },

        body: JSON.stringify(values)  // Use 'values' here instead of 'product'

      });

      if (response.ok) {

        const data = await response.json();

        alert("Product Added successfully!");

        console.log(data);

      } else {

        alert("Failed to Add Product");

      }

    } catch (error) {

      console.error("Error submitting the form", error);

    }

  }

Step 6 : add the bootstrap classes, values attributes in the text boxes & also show the error messages <p> for each textbox for incorrect value

return (

    <div className="container mt-5">

      <form className="form-container" onSubmit={handleSubmit}>

        <div className="col-md-4">

          <label htmlFor="title" className="form-label">

            Product Title

          </label>

          <input

            type="text"

            name="title"

            id="title"

            className="form-control"

            onChange={handleChange}

            onBlur={handleBlur}

            value={values.title}

          />

          {errors.title && touched.title && (<p className="text-danger">{errors.title}</p>)}

        </div>

        <div className="col-md-4">

          <label htmlFor="price" className="form-label">

            Product Price

          </label>

          <input

            type="text"

            name="price"

            id="price"

            className="form-control"

            onChange={handleChange}

            onBlur={handleBlur}

            value={values.price}

          />

          {errors.price && touched.price && (

            <p className="text-danger">{errors.price}</p>

          )}

        </div>

        <div className="col-md-4">

          <label htmlFor="description" className="form-label">

            Product Description

          </label>

          <input

            type="text"

            name="description"

            id="description"

            className="form-control"

            onChange={handleChange}

            onBlur={handleBlur}

            value={values.description}

          />

          {errors.description && touched.description && (

            <p className="text-danger">{errors.description}</p>

          )}

        </div>

        <div className="col-md-4">

          <label htmlFor="category" className="form-label">

            Product Category

          </label>

          <input

            type="text"

            name="category"

            id="category"

            className="form-control"

            onChange={handleChange}

            onBlur={handleBlur}

            value={values.category}

          />

          {errors.category && touched.category && (

            <p className="text-danger">{errors.category}</p>

          )}

        </div>

        <div className="col-md-4">

          <label htmlFor="image" className="form-label">

            Product Image URL

          </label>

          <input

            type="text"

            name="image"

            id="image"

            className="form-control"

            onChange={handleChange}

            onBlur={handleBlur}

            value={values.image}

          />

          {errors.image && touched.image && (

            <p className="text-danger">{errors.image}</p>

          )}

        </div>

        <div className="d-flex justify-content-start gap-2 col-md-4 mt-5 mb-5">

          <button type="submit" className="btn btn-primary"> Register </button>

          <button type="button" className="btn btn-secondary"> Cancel  </button>

        </div>

      </form>

    </div>

  );

};

export default AddProduct;

Formik

is a hook provided by Formik to manage form state and handle form submission.

It returns an object with several properties and methods to manage form fields, validation, and submission.

**Formik Properties**:

* values: An object containing the current values of form fields.
* errors: An object containing error messages corresponding to each field.
* touched: An object containing information about which fields have been "touched" (i.e., focused and then blurred).
* handleBlur: A function to handle the blur event for each field to mark it as touched.
* handleChange: A function to handle the change event for each field and update the corresponding value.
* handleSubmit: A function to handle the form's submission.
* setFieldValue: A function to manually set the value of a specific field.

**validationSchema**:

* The validationSchema is an object created using Yup to define validation rules for the form fields.