| **import \* as React from 'react';** |  |
| --- | --- |
|  | **import { Employee } from '@prisma/client';** |
|  | **import { Container, FormControl, InputLabel, FormHelperText, Input, TextField, Button, Box } from '@mui/material';** |
|  | **import { Controller, SubmitHandler, useForm } from "react-hook-form";** |
|  | **import \* as yup from "yup";** |
|  | **import { yupResolver } from "@hookform/resolvers/yup";** |
|  | **import { useRouter } from 'next/router';** |
|  | **import axios from "axios"** |
|  |  |
|  | **const schema = yup.object().shape({** |
|  | **name: yup.string().required(),** |
|  | **email: yup.string().email().required(),** |
|  | **salary: yup.number().required(),** |
|  | **// startDate: yup.dat,** |
|  | **title: yup.string().required(),** |
|  | **});** |
|  |  |
|  |  |
|  | **type Props = {** |
|  | **employees: Employee[]** |
|  | **}** |
|  |  |
|  | **type Inputs = yup.InferType<typeof schema>** |
|  |  |
|  | **const CreateEmployeeForm: React.FunctionComponent<Props> = ({** |
|  | **employees** |
|  | **}) => {** |
|  |  |
|  | **const { register, handleSubmit, watch, formState: { errors, isSubmitting }, control, reset } = useForm<Inputs>({** |
|  | **resolver: yupResolver(schema)** |
|  | **});** |
|  |  |
|  | **const router = useRouter()** |
|  |  |
|  | **const delay = (time: number) => new Promise((res) => setTimeout(res, time))** |
|  |  |
|  | **const onSubmit: SubmitHandler<Inputs> = async (data) => {** |
|  | **// await delay(1000)** |
|  | **// console.log(data);** |
|  | **// Call the axios api** |
|  | **try {** |
|  | **await axios.post("/api/createEmployee", {** |
|  | **name: data.name,** |
|  | **email: data.email,** |
|  | **salary: data.salary,** |
|  | **startDate: new Date(),** |
|  | **title: data.title** |
|  | **})** |
|  | **} catch (error) {** |
|  | **console.log(error)** |
|  | **}** |
|  | **// Call the api with data** |
|  | **reset()** |
|  | **router.push("/")** |
|  | **}** |
|  |  |
|  |  |
|  | **const formItems = [** |
|  | **{** |
|  | **name: "name",** |
|  | **label: "Name",** |
|  | **helperText: "Employee full name.",** |
|  | **type: "name"** |
|  | **},** |
|  | **{** |
|  | **name: "email",** |
|  | **label: "Email address",** |
|  | **helperText: "Employee email address.",** |
|  | **type: "email"** |
|  | **},** |
|  | **{** |
|  | **name: "title",** |
|  | **label: "Title | Designation",** |
|  | **helperText: "Employee Designation.",** |
|  | **type: ""** |
|  | **},** |
|  | **{** |
|  | **name: "salary",** |
|  | **label: "Salary",** |
|  | **helperText: "Employee annual package.",** |
|  | **type: "number"** |
|  | **}** |
|  | **]** |
|  |  |
|  | **return (** |
|  | **<Container>** |
|  | **<Box display={"flex"} flexDirection="column" justifyContent={"center"} maxWidth="80%" margin="auto">** |
|  | **{** |
|  | **formItems.map(f => (** |
|  | **<Controller** |
|  | **key={`${f.name}-id`}** |
|  | **// @ts-ignore** |
|  | **name={f.name as keyof Inputs}** |
|  | **control={control}** |
|  | **render={({ field: { onChange, value }, fieldState: { error } }) => {** |
|  | **console.log(error)** |
|  | **return (** |
|  | **<FormControl** |
|  | **disabled={isSubmitting}** |
|  | **error={!!error}** |
|  | **sx={{** |
|  | **my: 2** |
|  | **}}** |
|  | **>** |
|  | **<InputLabel htmlFor={f.name}>{f.label}</InputLabel>** |
|  | **<Input sx={{marginLeft: "12px"}} type={f.type} onChange={onChange} value={value} id={f.name} aria-describedby={`${f.name}-helper`} />** |
|  | **<FormHelperText id={`${f.name}-helper`}>{f.helperText}{error ? `- ${error.message}` : ""}</FormHelperText>** |
|  | **</FormControl>** |
|  | **)** |
|  | **}}** |
|  | **// @ts-ignore** |
|  | **defaultValue=""** |
|  | **/>** |
|  | **))** |
|  | **}** |
|  |  |
|  | **<Box my={2} display={"flex"} justifyContent={"center"}>** |
|  | **<Button disabled={isSubmitting} onClick={handleSubmit(onSubmit)}>Submit</Button>** |
|  | **<Button disabled={isSubmitting} onClick={() => reset()} variant={"outlined"}>Reset</Button>** |
|  | **</Box>** |
|  |  |
|  | **</Box>** |
|  | **</Container>** |
|  | **);** |
|  | **}** |
|  |  |
|  | **export default CreateEmployeeForm** |

| **import \* as React from 'react';** |  |
| --- | --- |
|  | **import { Employee } from '@prisma/client';** |
|  | **import { Container, FormControl, InputLabel, FormHelperText, Input, TextField, Button, Box } from '@mui/material';** |
|  | **import { Controller, SubmitHandler, useForm } from "react-hook-form";** |
|  | **import \* as yup from "yup";** |
|  | **import { yupResolver } from "@hookform/resolvers/yup";** |
|  | **import { useRouter } from 'next/router';** |
|  | **import axios from "axios"** |
|  |  |
|  | **const schema = yup.object().shape({** |
|  | **name: yup.string().required(),** |
|  | **email: yup.string().email().required(),** |
|  | **salary: yup.number().required(),** |
|  | **// startDate: yup.dat,** |
|  | **title: yup.string().required(),** |
|  | **});** |
|  |  |
|  |  |
|  | **type Props = {** |
|  | **employees: Employee[]** |
|  | **}** |
|  |  |
|  | **type Inputs = yup.InferType<typeof schema>** |
|  |  |
|  | **const CreateEmployeeForm: React.FunctionComponent<Props> = ({** |
|  | **employees** |
|  | **}) => {** |
|  |  |
|  | **const { register, handleSubmit, watch, formState: { errors, isSubmitting }, control, reset } = useForm<Inputs>({** |
|  | **resolver: yupResolver(schema)** |
|  | **});** |
|  |  |
|  | **const router = useRouter()** |
|  |  |
|  | **const delay = (time: number) => new Promise((res) => setTimeout(res, time))** |
|  |  |
|  | **const onSubmit: SubmitHandler<Inputs> = async (data) => {** |
|  | **// await delay(1000)** |
|  | **// console.log(data);** |
|  | **// Call the axios api** |
|  | **try {** |
|  | **await axios.post("/api/createEmployee", {** |
|  | **name: data.name,** |
|  | **email: data.email,** |
|  | **salary: data.salary,** |
|  | **startDate: new Date(),** |
|  | **title: data.title** |
|  | **})** |
|  | **} catch (error) {** |
|  | **console.log(error)** |
|  | **}** |
|  | **// Call the api with data** |
|  | **reset()** |
|  | **router.push("/")** |
|  | **}** |
|  |  |
|  |  |
|  | **const formItems = [** |
|  | **{** |
|  | **name: "name",** |
|  | **label: "Name",** |
|  | **helperText: "Employee full name.",** |
|  | **type: "name"** |
|  | **},** |
|  | **{** |
|  | **name: "email",** |
|  | **label: "Email address",** |
|  | **helperText: "Employee email address.",** |
|  | **type: "email"** |
|  | **},** |
|  | **{** |
|  | **name: "title",** |
|  | **label: "Title | Designation",** |
|  | **helperText: "Employee Designation.",** |
|  | **type: ""** |
|  | **},** |
|  | **{** |
|  | **name: "salary",** |
|  | **label: "Salary",** |
|  | **helperText: "Employee annual package.",** |
|  | **type: "number"** |
|  | **}** |
|  | **]** |
|  |  |
|  | **return (** |
|  | **<Container>** |
|  | **<Box display={"flex"} flexDirection="column" justifyContent={"center"} maxWidth="80%" margin="auto">** |
|  | **{** |
|  | **formItems.map(f => (** |
|  | **<Controller** |
|  | **key={`${f.name}-id`}** |
|  | **// @ts-ignore** |
|  | **name={f.name as keyof Inputs}** |
|  | **control={control}** |
|  | **render={({ field: { onChange, value }, fieldState: { error } }) => {** |
|  | **console.log(error)** |
|  | **return (** |
|  | **<FormControl** |
|  | **disabled={isSubmitting}** |
|  | **error={!!error}** |
|  | **sx={{** |
|  | **my: 2** |
|  | **}}** |
|  | **>** |
|  | **<InputLabel htmlFor={f.name}>{f.label}</InputLabel>** |
|  | **<Input sx={{marginLeft: "12px"}} type={f.type} onChange={onChange} value={value} id={f.name} aria-describedby={`${f.name}-helper`} />** |
|  | **<FormHelperText id={`${f.name}-helper`}>{f.helperText}{error ? `- ${error.message}` : ""}</FormHelperText>** |
|  | **</FormControl>** |
|  | **)** |
|  | **}}** |
|  | **// @ts-ignore** |
|  | **defaultValue=""** |
|  | **/>** |
|  | **))** |
|  | **}** |
|  |  |
|  | **<Box my={2} display={"flex"} justifyContent={"center"}>** |
|  | **<Button disabled={isSubmitting} onClick={handleSubmit(onSubmit)}>Submit</Button>** |
|  | **<Button disabled={isSubmitting} onClick={() => reset()} variant={"outlined"}>Reset</Button>** |
|  | **</Box>** |
|  |  |
|  | **</Box>** |
|  | **</Container>** |
|  | **);** |
|  | **}** |
|  |  |
|  | **export default CreateEmployeeForm** |