**React-Hook-Form**

* Use npm i react-hook-form
* Use useForm<FormValues>() from the library
* We have to manage form state like data has default, either interacted or not.
* We got a method called ‘register’ from the form variable which returns four methods which can be used for managing a current variable state e.g., const {name, ref, onChange, onBlur} = register(“username”), provides these values to the input field, but instead use <input {…register(“username”)}/>
* You can visualize form state using react hook form devtools, use npm i @hookform/devtools
* Import DevTool from the library and write this as component after the </form> tag, we can also get control method from useForm method and pass this prop to the control prop of DevTool
* React hook form itself manages unnecessary re-renders called by input field
* We also got ‘handleSubmit’ from useForm(), pass this to form onSubmit={handleSubmit(onSubmit)}
* This also requires the formValues datatypes so create an array of form variables datatypes and write onSubmit = (data: FormValues) => {}
* In <form noValidate> which prevents the browser to not handle any further validation itself
* In {…required(“id-name”,{

required: “error message”,

})}

* We can also do pattern validation using this react hook form:

{

…required(“email”, {

pattern: {

value: [regex value for email] /^((a-zA-Z0-9.\_%-)+@(a-zA-Z0-9.-)+\.(a-zA-Z)(2,))$/

message: ‘Invalid email format’

}

})

}

* We also got formState from useForm(), we got const {errors} = formState and we can display the error message below the input tag like <p>{errors.username?.message}</p>
* We can also pass to the register, validate: (fieldValue) => {checking for specific validation} or in case we wanted multiple types of validation, we can simply write like validate: { notAdmin: ()=> {}, notBlackListed: ()=> {}}
* We canm add default value names like in useForm({

defaultValues: {}})

* In case we got nested fields, we can simply write …register(“social.twitter”)
* In case we want array of two numbers to be stored under same id, user …register(“phoneNumbers.0”) and phoneNumbers.1 for two fields
* Now import from react-hook-form, useFieldArray, const {fields, append, remove} = useFieldArray(

{

Name: ‘phNumbers,

control

}

), then in the jsx write this {fields.map((field, index)=>{

<div key={field.id}>

<input {…register(“phNumbers.${index}.number”)}

</div>

})}, in order to add number user onClick(()=>append({number: “”})), and then for removing, we call onClick(()=>remove(index)), call this inside where you render fields.map()

* For numeric values, inside …register(“”, valueAsNumber: true) for not storing the number as a string, and for dates, use valueAsDate:true for exactly storing date values
* For again not calling the watch continuously, we can simply call watch callback, using useEffect(()=>{const subscription = watch(value)=> {console.log(value)} return ()=> subscription.unsubscribe(),,,, this will simply not rerencder the input fields of form},[watch *’From useForm()’*]
* For obtaining values of a form, use getValues method provided by react-hook-form
* For setting values of a form, use setValue method
* You can also import touchedFields and dirtyFields, isDirty and isValid, isSubmitting, isSubmitted, isSubmitSuccessful, SubmitCount, Esubmitting from formState of useForm()
* For resetting call reset from useForm
* Mode can be set to mode: “onBlur” or “onTouched” or “onChange” or “all”
* For manually triggering, use trigger method from useForm()