Использование useForm из библиотеки <https://react-hook-form.com/> в сочетании с библиотекой zod и TypeScript значительно упрощает процесс управления формами и валидации, предоставляя мощные инструменты для типизации данных и строгой проверки. zod — это библиотека для построения схем, которая позволяет валидировать данные и генерировать типы TypeScript на основе этих схем.

Давайте рассмотрим, как использовать useForm с zod и TypeScript подробно.

**Установка зависимостей**

Для начала необходимо установить библиотеки react-hook-form и zod, а также адаптер для их интеграции @hookform/resolvers:

npm install react-hook-form zod @hookform/resolvers

Пример реализации формы с zod и TypeScript

Пример кода:

import React from 'react';

import { useForm } from 'react-hook-form';

import { z } from 'zod';

import { zodResolver } from '@hookform/resolvers/zod';

**// 1. Задаем Zod-схему для валидации формы**

const formSchema = z.object({

  name: z

    .string()

    .min(2, { message: 'Имя должно содержать минимум 2 символа' })

    .max(30, { message: 'Имя не может быть длиннее 30 символов' }),

  email: z

    .string()

    .email({ message: 'Введите корректный email' }),

  age: z

    .number({ invalid\_type\_error: 'Возраст должен быть числом' })

    .min(18, { message: 'Возраст должен быть не менее 18 лет' }),

});

**// 2. Генерируем типы на основе схемы**

type FormData = z.infer<typeof formSchema>;

**// 3. Конфигурируем useForm с использованием zodResolver**

function MyForm() {

  const {

    register,

    handleSubmit,

    formState: { errors, isSubmitting },

    reset

  } = useForm<FormData>({

    resolver: zodResolver(formSchema), // подключаем валидацию через zod

    mode: 'onSubmit', // режим валидации (onChange, onBlur, onSubmit и т.д.)

  });

  const onSubmit = (data: FormData) => {

    console.log('Данные формы:', data);

    reset(); // очищаем форму после успешной отправки (опционально)

  };

  return (

    <form onSubmit={handleSubmit(onSubmit)}>

      {/\* Поле: Имя \*/}

      <div>

        <label htmlFor="name">Имя:</label>

        <input

          {...register('name')}

          id="name"

          placeholder="Введите имя"

        />

        {errors.name && <p style={{ color: 'red' }}>{errors.name.message}</p>}

      </div>

      {/\* Поле: Email \*/}

      <div>

        <label htmlFor="email">Email:</label>

        <input

          {...register('email')}

          id="email"

          placeholder="Введите email"

        />

        {errors.email && <p style={{ color: 'red' }}>{errors.email.message}</p>}

      </div>

      {/\* Поле: Возраст \*/}

      <div>

        <label htmlFor="age">Возраст:</label>

        <input

          type="number"

          {...register('age', { valueAsNumber: true })} // преобразование значения в число

          id="age"

        />

        {errors.age && <p style={{ color: 'red' }}>{errors.age.message}</p>}

      </div>

      {/\* Кнопка отправки \*/}

      <button type="submit" disabled={isSubmitting}>Отправить</button>

    </form>

  );

}

export default MyForm;

**Пошаговое описание работы**

**Создание схемы с помощью zod:**

   Мы определяем правила валидации для каждого поля через z.object. Например, указываем, что поле name должно содержать не менее двух символов, а поле email должно соответствовать формату email. Схема также автоматически генерирует понятные сообщения об ошибках для каждого правила, или мы можем задавать их вручную.

const formSchema = z.object({

  name: z.string().min(2, { message: 'Имя должно содержать минимум 2 символа' }),

  email: z.string().email({ message: 'Введите корректный email' }),

  age: z.number().min(18, { message: 'Возраст должен быть не менее 18 лет' }),

});

**Генерация типов:**

   Мы используем z.infer для генерации TypeScript типа FormData, который точно повторяет структуру нашей схемы:

     type FormData = z.infer<typeof formSchema>;

   Теперь TypeScript автоматически проверяет, что данные соответствуют заданной схеме.

**Интеграция zod с useForm:**

   Мы подключаем схему zod к useForm с помощью zodResolver из библиотеки @hookform/resolvers. Это связывает нашу схему с формой и позволяет выполнять валидацию через Zod.

      const { register, handleSubmit, formState: { errors, isSubmitting } } = useForm({

     resolver: zodResolver(formSchema),

   });

**Обработка значений:**

   Поля формы регистрируются с помощью register, связывая их с определенными ключами схемы:

            Ошибки из zod доступны через объект errors из formState.

**Пример отображения ошибок:**

      {errors.name && {errors.name.message}}

**Ручная обработка данных:**

   Функция onSubmit вызывается только если валидация проходит успешно. Данные автоматически преобразуются в соответствии со схемой и передаются в обработчик.

      const onSubmit = (data: FormData) => {

     console.log(data);

   };

**Дополнительные функции:**

   reset: позволяет сбросить форму к изначальному состоянию после отправки.

   Параметр valueAsNumber: используется для преобразования значения из строки в число для числовых полей.

**Режимы валидации**

React Hook Form позволяет настроить, когда должна происходить валидация. Это задается через параметр mode:

onSubmit (по умолчанию): Валидация выполняется только при отправке формы.

onBlur: Валидация происходит при потере фокуса с поля.

onChange: Проверка выполняется на каждое изменение.

all: Комбинирует все три режима.

**Пример:**

const { register, handleSubmit } = useForm({

  resolver: zodResolver(formSchema),

  mode: 'onBlur',

});

import React from 'react';

import { useForm } from 'react-hook-form';

import { z } from 'zod';

import { zodResolver } from '@hookform/resolvers/zod';

import { useMutation } from '@tanstack/react-query';

import axios from 'axios';

**// 1. Определяем Zod-схему для данных формы**

const formSchema = z.object({

  name: z

    .string()

    .min(2, { message: 'Имя должно содержать минимум 2 символа' })

    .max(30, { message: 'Имя не может быть длиннее 30 символов' }),

  email: z

    .string()

    .email({ message: 'Введите корректный email' }),

  age: z

    .number({

      invalid\_type\_error: 'Возраст должен быть числом',

    })

    .min(18, { message: 'Возраст должен быть не менее 18 лет' }),

});

**// 2. Генерируем типы данных на основе Zod-схемы**

type FormData = z.infer<typeof formSchema>;

**// 3. Настройка React Hook Form с Zod валидацией**

function MyFormWithMutation() {

  const {

    register,

    handleSubmit,

    formState: { errors },

    reset,

  } = useForm<FormData>({

    resolver: zodResolver(formSchema), // Используем zod для валидации

    mode: 'onSubmit', // Валидация только при отправке формы

  });

**// 4. Настройка useMutation**

  const mutation = useMutation({

    mutationFn: async (data: FormData) => {

      // Симулируем API-запрос с помощью axios

      const response = await axios.post('/api/submit', data);

      return response.data;

    },

    onSuccess: (data) => {

      console.log('Успех:', data);

      alert('Данные успешно отправлены!');

      reset(); // Сброс формы на начальное состояние

    },

    onError: (error: any) => {

      console.error('Ошибка запроса:', error.message || error);

      alert('Произошла ошибка при отправке данных.');

    },

  });

**// 5. Обработчик отправки формы**

  const onSubmit = (data: FormData) => {

    mutation.mutate(data); // Вызываем useMutation с данными формы

  };

  return (

    <form onSubmit={handleSubmit(onSubmit)}>

      {/\* Поле: Имя \*/}

      <div>

        <label htmlFor="name">Имя:</label>

        <input

          {...register('name')}

          id="name"

          placeholder="Введите имя"

        />

        {errors.name && <p style={{ color: 'red' }}>{errors.name.message}</p>}

      </div>

      {/\* Поле: Email \*/}

      <div>

        <label htmlFor="email">Email:</label>

        <input

          {...register('email')}

          id="email"

          placeholder="Введите email"

        />

        {errors.email && <p style={{ color: 'red' }}>{errors.email.message}</p>}

      </div>

      {/\* Поле: Возраст \*/}

      <div>

        <label htmlFor="age">Возраст:</label>

        <input

          type="number"

          {...register('age', { valueAsNumber: true })} // Преобразуем числовые значения

          id="age"

          placeholder="Введите возраст"

        />

        {errors.age && <p style={{ color: 'red' }}>{errors.age.message}</p>}

      </div>

      {/\* Отображение состояния мутации \*/}

      {mutation.isLoading && <p>Отправка данных...</p>}

      {mutation.isError && <p style={{ color: 'red' }}>Произошла ошибка при отправке данных.</p>}

      {/\* Кнопка отправки \*/}

      <button type="submit" disabled={mutation.isLoading}>

        {mutation.isLoading ? 'Отправка...' : 'Отправить'}

      </button>

    </form>

  );

}

export default MyFormWithMutation;

**Пошаговое объяснение**

Zod-схема (z.object) для валидации формы:

   Мы описываем схему данных, которые хотим отправить, используя zod. Это позволяет задать строгие правила валидации (например, минимальное количество символов, формат email, и т. д.):

      const formSchema = z.object({

     name: z.string().min(2).max(30),

     email: z.string().email(),

     age: z.number().min(18),

   });

     Благодаря функции z.infer, мы генерируем тип FormData из схемы для TypeScript:

      type FormData = z.infer;

**Подключение React Hook Form с Zod через zodResolver:**

   Метод useForm принимает resolver, чтобы передать схему валидации:

      const { register, handleSubmit } = useForm({

     resolver: zodResolver(formSchema),

   });

     Это значит, что React Hook Form будет проверять данные согласно Zod-схеме перед вызовом onSubmit.

**useMutation для отправки данных на сервер:**

   Мы используем useMutation для обработки запросов и их состояний:

      const mutation = useMutation({

     mutationFn: async (data) => {

       const response = await axios.post('/api/submit', data);

       return response.data;

     },

     onSuccess: () => {

       reset(); // Сбрасываем форму

     },

     onError: (error) => {

       alert('Ошибка: ' + error.message);

     },

   });

     Здесь:

   mutationFn — это код отправки данных (в данном случае POST-запрос через axios).

   onSuccess — вызывается, если запрос выполнен успешно.

   onError — вызывается, если запрос завершился ошибкой.

**Связывание onSubmit с React Hook Form и useMutation:**

   Когда данные формы проходят валидацию, мы вызываем mutation.mutate(data):

      const onSubmit = (data: FormData) => {

     mutation.mutate(data);

   };

  Состояния мутации (mutation.isLoading, mutation.isError):

   isLoading: Указывает, что запрос находится в процессе выполнения.

   isError: Указывает, что запрос завершился ошибкой.

   Эти состояния отображаются в интерфейсе для улучшенного UX:

          {mutation.isLoading && Отправка данных...}

     {mutation.isError && Ошибка при отправке данных.}

    Добавление кнопки блокировки на время отправки данных:

        {mutation.isLoading ? 'Отправка...' : 'Отправить'}

Преимущества этого подхода:

Zod + TypeScript:

   Автоматическая типизация данных формы.

   Эффективная и декларативная валидация входных данных.

useMutation для API-запросов:

   Простое управление состояниями запроса с минимальным кодом.

   Очистка формы после успешной отправки.

   Обработка серверных ошибок прямо в интерфейсе.

React Hook Form:

   Интеграция с другими библиотеками, такими как zod и react-query.

   Легкость в работе с формами, минимальная работа с состоянием вручную.