11 — Formulários e validação

11 — Formulários e validação

> react-hook-form + zod, sempre via `useValidatedForm`. Schema é a fonte única do formato: o tipo do formulário é **inferido** dele, nunca escrito à mão. Campos do formulário usam os mesmos nomes `_Prefixo` do backend.

`useValidatedForm`

```
// src/hooks/useValidatedForm.tsimport { useForm, UseFormProps, UseFormReturn } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";
import { z } from "zod";
interface UseValidatedFormProps<T extends z.ZodType> extends Omit<UseFormProps<z.infer<T>>, "resolver"> {
  schema: T;
}
export const useValidatedForm = <T extends z.ZodType>({
  schema,
  ...formProps
}: UseValidatedFormProps<T>): UseFormReturn<z.infer<T>> =>
  useForm<z.infer<T>>({
    resolver: zodResolver(schema),
    mode: "onChange", // valida em tempo real
    ...formProps
  });
```

Use sempre este hook, nunca `useForm` direto. Ele garante o resolver e o `mode: "onChange"` uniformes.

`mode: "onChange"` valida a cada digitação. Para formulário grande e caro de validar, sobrescreva para `"onBlur"` — o spread permite.

schemas.ts

Um arquivo por tela, ao lado do componente.

```
// src/pages/Fornecedor/Cadastro/schemas.ts
import { z } from "zod";
export const fornecedorSchema = z.object({  _Nome: z.string().min(1, "Informe o nome").max(120,
"Máximo de 120 caracteres"),
  _CNPJ: z
    .string()
    .min(1, "Informe o CNPJ")
    .regex(/^\d{2}\.\d{3}\.\d{3}\/\d{4}-\d{2}$/, "CNPJ inválido"),  _Email: z.string().email("E-
mail inválido").optional().or(z.literal("")),
  _Ativo: z.boolean(),
  _DataCadastro: z.string().min(1, "Informe a data")
});
export type FornecedorFormData = z.infer<typeof fornecedorSchema>;
export const filtroFornecedorSchema = z.object({
  _Nome: z.string(),
  _Ativo: z.boolean()
});
export type FiltroFornecedorData = z.infer<typeof filtroFornecedorSchema>;
```

Regras:

- **Nunca escreva a interface do formulário à mão.** Sempre `z.infer<typeof schema>`. Escrever as duas cria divergência silenciosa.
- Toda regra tem mensagem em português — ela vai direto para a tela.
- Nomes de campo espelham o backend (`_Nome`), evitando mapeamento no submit.
- Um schema por formulário. Tela com dois formulários independentes tem dois schemas.

Campo opcional que aceita vazio

```
_Email: z.string().email("E-mail inválido").optional().or(z.literal(""));
```

`.optional()` sozinho não basta: um `TextField` limpo devolve `""`, não `undefined`, e `""` falha na validação de e-mail.

O formulário

Duas formas, ambas válidas.

Com `FormField` (preferível)

```
import { FormField } from "@components/common";
<FormField name="_Nome" control={form.control} label="Nome" size="small" fullWidth />;
```

Encapsula o `Controller` e liga `error`/`helperText` ao `fieldState`. Use quando o campo é um `TextField` comum.

Com `Controller` (quando precisa do componente MUI cru)

```
<Controller
  name="_Nome"
  control={form.control}
  render={({ field, fieldState }) => (
    <TextField
      {...field}
      label="Nome"
      size="small"
      fullWidth
```

```
      error={!!fieldState.error}
      helperText={fieldState.error?.message}
    />
  )}
/>
```

Mais verboso, mas necessário quando o componente não é um `TextField` ou exige props que o `FormField` não repassa.

> Evite usar `Controller` direto quando `FormField` bastaria — **prefira `FormField`** e reserve `Controller` para os casos que realmente precisam do componente MUI cru.

Submit

```
const form = useValidatedForm({
  schema: fornecedorSchema,  defaultValues: { _Nome: "", _CNPJ: "", _Ativo: true, _DataCadastro:
dataHoje }
});
const handleSalvar = async (data: FornecedorFormData) => {
  await salvarAsync({
    Fornecedor: {
      _OID: String(fornecedorData?.Fornecedor._OID ?? ""),
      _Nome: data._Nome,
      _CNPJ: data._CNPJ
    },
    msgSucesso: "Fornecedor salvo com sucesso!"
  });
};
<Button onClick={form.handleSubmit(handleSalvar)}>Salvar</Button>;
```

`form.handleSubmit(fn)` só chama `fn` se a validação passar. **Não valide manualmente antes** — é o que o resolver faz.

`defaultValues` é obrigatório

Sempre forneça `defaultValues` com todos os campos. Sem ele, o campo nasce não-controlado e vira controlado na primeira digitação — o React emite warning e o `reset()` não limpa direito.

Reset

```
const handleNovoRegistro = () => {
  form.reset(); // volta aos defaultValues  filtroForm.reset({ _Nome: "", _Ativo: true }); //
valores explicitos
};
```

Inputs com máscara

`CnpjInput`, `PhoneInput`, `CurrencyInput` e `DatePickerInput` de `common/` — ver 08.

Moeda precisa de conversão no submit:

```
import { CurrencyInput, parseCurrencyValue } from "@components/common";
const handleSalvar = (data: FormData) => {  salvar({ Fornecedor: { _Limite:
parseCurrencyValue(data._Limite) } });
};
```

O campo guarda o texto formatado (`"1.234,56"`); o backend quer número. `parseCurrencyValue` faz a ponte. Esquecer isso envia string e o backend rejeita.

Datas

O backend Curio espera data com tempo:

```
const dataFormatada = `${data._DataExpiracao}T00:00:00.0`;
```

O campo do formulário guarda `"2026-08-18"`; o request precisa de `"2026-08-18T00:00:00.0"`. Faça a conversão **no handler**, não no schema — o schema descreve o formulário, não o payload.

Para gerar a data de hoje no formato do input:

```
const hoje = new Date();const dataHoje = `${hoje.getFullYear()}-${String(hoje.getMonth() + 1).padStart(2, "0")}-${String(hoje.getDate()).padStart(2, "0")}`;
```

Não use `toISOString().split("T")[0]` — converte para UTC e retorna o dia anterior à noite no fuso brasileiro.

Validação vinda do servidor

O backend valida de novo. Um `ValidationError` do Curio traz `detail: string[]`, já convertido em notificação pelo `queryClient` (07).

Não tente espelhar toda regra de servidor no zod. O zod cobre o que dá para verificar no cliente (obrigatório, formato, tamanho); regra de negócio é do servidor.

Erros comuns

Sintoma	**Causa**
"changing an uncontrolled input"	Faltou o campo em `defaultValues`
`reset()` não limpa	Idem
Submit não dispara e nada aparece	Validação falhou num campo sem `FormField` / `helperText` visível
Backend rejeita a data	Faltou o sufixo `T00:00:00.0`
Backend recebe moeda como texto	Faltou `parseCurrencyValue`
Tipo do form diverge do schema	Interface escrita à mão em vez de `z.infer`

Revision #2
Created Thu, Aug 20, 2026 5:42 PM by Geraldo Barbosa
Updated Tue, Aug 25, 2026 4:53 PM by Geraldo Barbosa