External State (in stateful components)	View
Input Fields : String	ErrorLabels : String
1. email	1. email
2. password	2. password
3. confirmPassword	3. confirmPassword
4. age (optional)	4. age
Checkbox : Bool	
5. acceptPolicy	5. acceptPolicy

stateless input devices / component

Buttons

1. Submit
2. Cancel

Validation

1. (instant feedback, after 1st submission) Every key stroke
2. Before submission

(translate / annotate external state / event into internal state / event

with internal representation / "business model" encoded in the type system,

so then under the protection of the type system when transforming the data)

e.g. Json. Decode

External State		Internal Representation	Validator
InputField : String	→	Email = String	Validator String Email
I~ : S~	→	Password = String	V~ String P~
I~ : S~	→	ConfirmPassword = String	V~ String CP~
I~ : S~	→	Age = Int	V~ Int Age
Checkbox : Bool	→	PolicyAcceptance = Bool	V~ Bool PA~

Field raw a = Field raw (Validity a)

raw = | String | Bool

Validity a = | NotValidated | Validated (| Valid a | Invalid ErrMsg) ≅ | NotV~ | Valid a | Invalid ErrMsg

ErrMsg = String

Validator a b = a → Result ErrMsg b

(Internal) Event a = | OnSubmit | OnChange a | OnRelatedChange

← capture the desired behavior difference before and after 1st Submit.

↖ (potentially cyclic) dependencies / constraints between state variables

Current Model

Required Behavior

1. Insert feedback only after 1st submission attempt.
   (display Error labels in View)
   - ✓ i) one additional state variable:

     attemptedOnce : Bool

   - ii) Union type on top of current state/Model:

FirstAttempt Model
OtherAttempt Model

     (no structural changes, unfavorable)

other Design Choices

1. submit button is disabled until all fields are valid
2. Insert feedback states (for each field except confirmPassword) after 1st valid input

Field is a named subdimension that

encodes one raw input (variable)

and the validation status of this piece of input/data.

Whether confirmPassword,

all validation computations are independent

so we can use Applicative to coordinate results.

email: (raw, NotValidated) ⟶ (raw, Valid a | Invalid ErrMsg)

password: (~, ~) ⟶ (~, V~ | I~)

⋮ ⋮

Pure (submit) <*> getValidity email <*> getValidity password <*> ...

If a large number of fields are validated

by the same validation logic/function,

can put them in a list and use 'traverse'.

dependency / group constraint

the validation result of confirmPassword depends on the validation result of password.

password = (___ , NotValidated | Invalid ~)

always → confirmPassword = (___ , NotValidated)

validation.reset