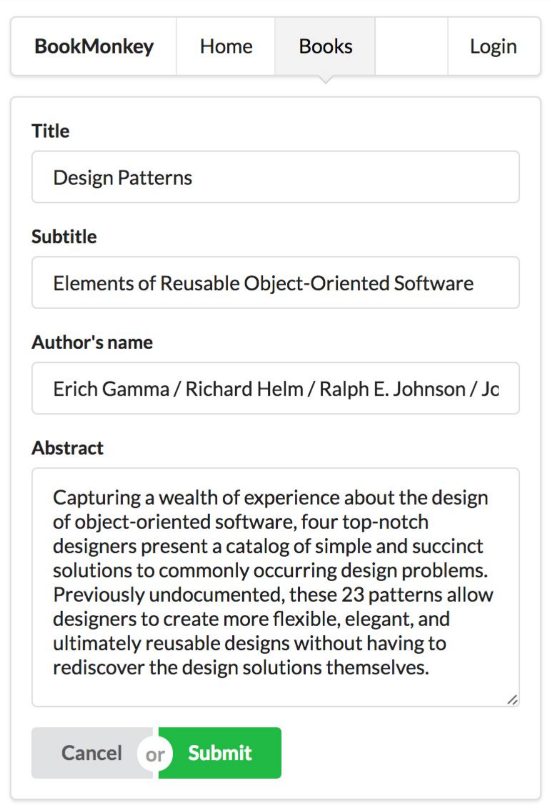
Workshop

React Forms

Task

What do you know about forms?

Handle user input in your application



Title

Design Patterns

Subtitle

Elements of Reusable Object-Oriented Software

Author's name

Erich Gamma / Richard Helm / Ralph E. Johnson / Jc

Abstract

Capturing a wealth of experience about the design of object-oriented software, four top-notch designers present a catalog of simple and succinct solutions to commonly occurring design problems. Previously undocumented, these 23 patterns allow designers to create more flexible, elegant, and ultimately reusable designs without having to rediscover the design solutions themselves.

Why / What you'll learn

- Manage user input to create and edit data
- Create forms with React
- Different types of input validation

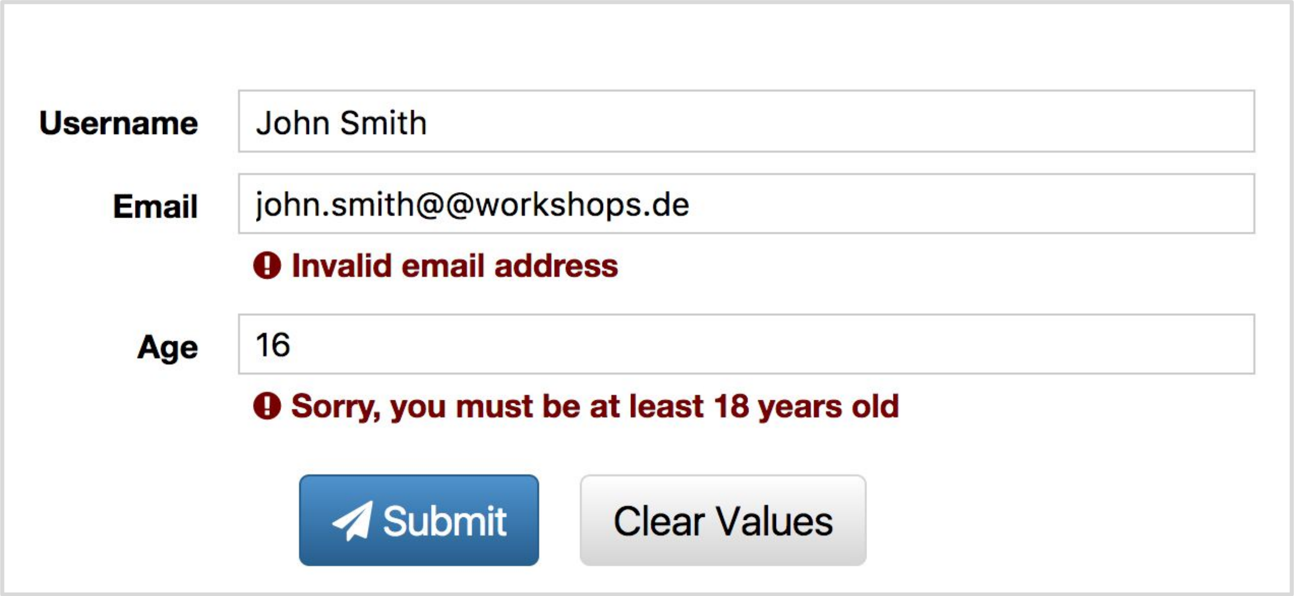
Form Validation

Validation shows the users what they are doing wrong and how to fix it as soon as possible!

Why / What you'll learn

- Sometime a user needs to be guided through a form
- Provide the user of your form a better UX
- Write and use functions for errors and warnings
- Create a reusable input field that renders errors and warnings

Validation - Example



Username	John Smith
Email	john.smith@@workshops.de
	❗ Invalid email address
Age	16
	❗ Sorry, you must be at least 18 years old

Form Validation - Strategies

There are multiple ways to achieve Form Validation (Client-side):

- **Built-in form validation**
  - Uses HTML5 form validation features.
- **JavaScript - The constraint validation API**
  - More and more browsers now support the constraint validation API, and it's becoming reliable.
- **JavaScript - Custom Implementation**
  - Sometimes the constraint validation API is not enough.

Built-in form validation

HTML5 Built-in Validators

In HTML5 there are **built-in validators** that can be used with the Built-in form validation and constraint **validation API**

HTML5 Built-in Validators

- **type**
  - The type attribute of an input is also a validator.
  - E.g. email, number, color, date, datetime-local, month, number, range, password
- **required**
  - A value is required
- **minlength, maxlength**
  - The minimal or maximal length of the input value
- **Pattern**
  - The input value has to match the given regular expression

Built-in form validation

<code>

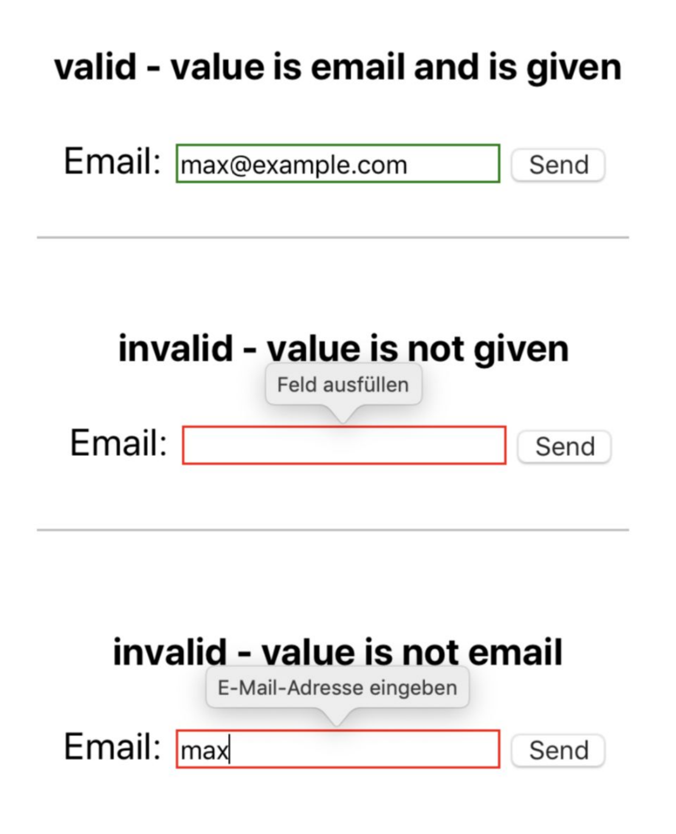
Example usage of build-in form validations

```
export function SimpleForm() {
    const [email, setEmail] = useState<string>('');

    function handleChange({ target: { value } }: React.ChangeEvent<HTMLInputElement>) {
        setEmail(value);
    }
    return (
        <form onSubmit={sendForm}>
            <label htmlFor="userEmail">Email: </label>
            <input id="userEmail"
                name="userEmail"
                type="email"
                required
                value={email}
                onChange={handleChange} />
            <button>Send</button>
        </form>);
}
```

Using CSS-Pseudo classes with validation

```
input {
    outline: none;
}
input:valid {
    border: 1px solid green;
}
input:invalid {
    border: 1px solid red;
}
```



valid - value is email and is given

Email: max@example.com

invalid - value is not given

Feld ausfüllen

Email:

invalid - value is not email

E-Mail-Adresse eingeben

Email: max

Handle Submit in Forms

Overwrite the default event on Submit. Otherwise you trigger an Request.

```
<form onSubmit={onSubmit}>
<!-- ... -->
<input type="submit" value="Submit"/>
</form>

onSubmit (event) {
    // do something with this.state
    event.preventDefault();
}
```

Task

Create a form with built-in validation

Disadvantages of Built-in Form Validation

- **No immediate user guidance**
  - Error messages are shown on form submit.
  - There is no way to immediately show an input hint, error or success message depending on the inputs validity and / or touched state - while the user is typing.
- **The error messages are pre-styled and pre-defined**
  - We'd like to show a custom message with our custom design and behavior
- **No Cross-field validation**
  - Validation happens on input element basis.

Validate on with our own strategy

Validate on with our own strategy

Let us define an own validation strategy:

- Each input is in the state: (un-)touched and (in-)valid
- The error message depends on the kind of the error.
- Disable each submit trigger (submit button) if the form is in an invalid state.

Validation Form - novalidate

<code>

Disable build-in browser-specific HTML5 validation for a form

```
<!-- "noValidate" with a capital 'V'. -->
<form onSubmit={handleSubmit} novalidate></form>
```

Validate on with our own strategy

<code>

Disable built-in validation

```
export function SimpleForm() {
    return (
      <form onSubmit={sendForm} novalidate>
            {emailError && (<p className="errorMessage">{emailError}</p>)}
            {!emailError && email && (<p className="successMessage">Thank you for your email.</p>)}
            {!emailError && !email && (<p className="hintMessage">Please input an email.</p>)}
            <label htmlFor="userEmail">Email: </label>
            <input id="userEmail"
                name="userEmail"
                type="email"
                required
                value={email}
                onChange={e => handleChange(e)} />
            <button type="submit" disabled="disabled">Send</button>
        </form>
   )
}
```

Validate on with our own strategy

A reference can be bound directly to a DOM Node.

```
import React, { useState, useRef } from 'react';
import './SimpleForm.css';


export function SimpleForm() {
    const [email, setEmail] = useState('');
    const [emailError, setEmailError] = useState('');
    const submitButtonRef = useRef(undefined);
    const formRef = useRef(undefined);

    // ..
```

Validate on with our own strategy

<code>

Bind the reference to the DOM Node

```
export function SimpleForm() {
    return (
      <form ref={formRef} onSubmit={sendForm} novalidate>
            {emailError && (<p className="errorMessage">{emailError}</p>)}
            {!emailError && email && (<p className="successMessage">Thank you for your email.</p>)}
            {!emailError && !email && (<p className="hintMessage">Please input an email.</p>)}
            <label htmlFor="userEmail">Email: </label>
            <input id="userEmail"
                name="userEmail"
                type="email"
                required
                value={email}
                onChange={e => handleChange(e)} />
            <button ref={submitButtonRef} disabled="disabled">Send</button>
        </form>
   )
}
```

Validate on with our own strategy

<code>

Define custom error messages for our form

```
export function SimpleForm() {
    return (
      <form onSubmit={sendForm} novalidate>
            {emailError && (<p className="errorMessage">{emailError}</p>)}
            {!emailError && email && (<p className="successMessage">Thank you for your email.</p>)}
            {!emailError && !email && (<p className="hintMessage">Please input an email.</p>)}
            <label htmlFor="userEmail">Email: </label>
            <input id="userEmail"
                name="userEmail"
                type="email"
                required
                value={email}
                onChange={e => handleChange(e)} />
            <button type="submit" disabled="disabled">Send</button>
        </form>
   )
}
```

Validate on with our own strategy

<code>

Define our handleChange method

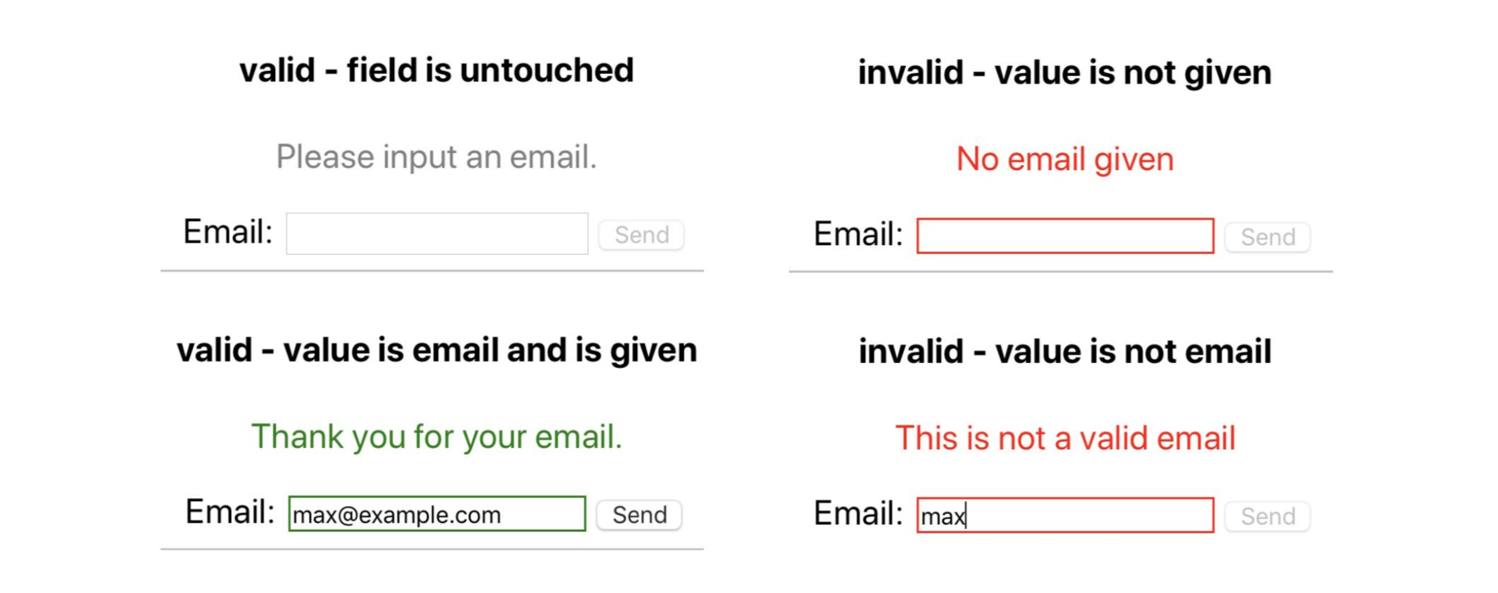
```
export function SimpleForm() {
    const [email, setEmail] = useState('');
    const [emailError, setEmailError] = useState('');
    const handleEmailChange = ({ target: { value } }: React.ChangeEvent<HTMLInputElement>) => {
      setEmail(value);
      const error = validateEmail(value);
      if (error) {
        setEmailError(error);
      } else {
        setEmailError(null);
      }
    };

    return (
      ...
   )
}
```

Validate on with our own strategy

<code>

See our own strategy in action in different states



valid - field is untouched

Please input an email.

Email:

invalid - value is not given

No email given

Email:

valid - value is email and is given

Thank you for your email.

Email: max@example.com

invalid - value is not email

This is not a valid email

Email: max

Delayed Validation

<code>

Validating on every keystroke is generally not great UX

```
export function SimpleForm() {
  const handleEmailValidation = ({ target: { value } }: React.ChangeEvent<HTMLInputElement>) => {
   const error = validateEmail(value);
   if (error) {
    setEmailError(error);
   } else {
    setEmailError(null);
   }
  };

  return (
   ...
   <input id="userEmail"
    ...
    onBlur={e => handleEmailValidation(e)}
    onChange={e => handleEmailChange(e)} />
  )
}
```

Validation Function

Validation Function

- Get an object of all inputs as values

```
values = { age: 16 }
```

- Return an object of all errors

```
errors = { age: 'Sorry, you must
be at least 18 years old' }
```

Validation Function

<code>

Simple function that accepts form values and returns an error object

```
const validate = values => {
  const errors = {}
  if (Number(values.age) < 18) {
    errors.age = 'Sorry, you must be at least 18 years old'
  }
  return errors;
}
```

Validation Function - Example Required

If an input is empty it is undefined

```
const validate = values => {
  const errors = {}
  if (!values.age) {
    errors.age = 'Required'
  }
  return errors
}
```

Task

**Create a form
with our own validation**

We teach.

workshops.de