Week 10/11 - Working with forms

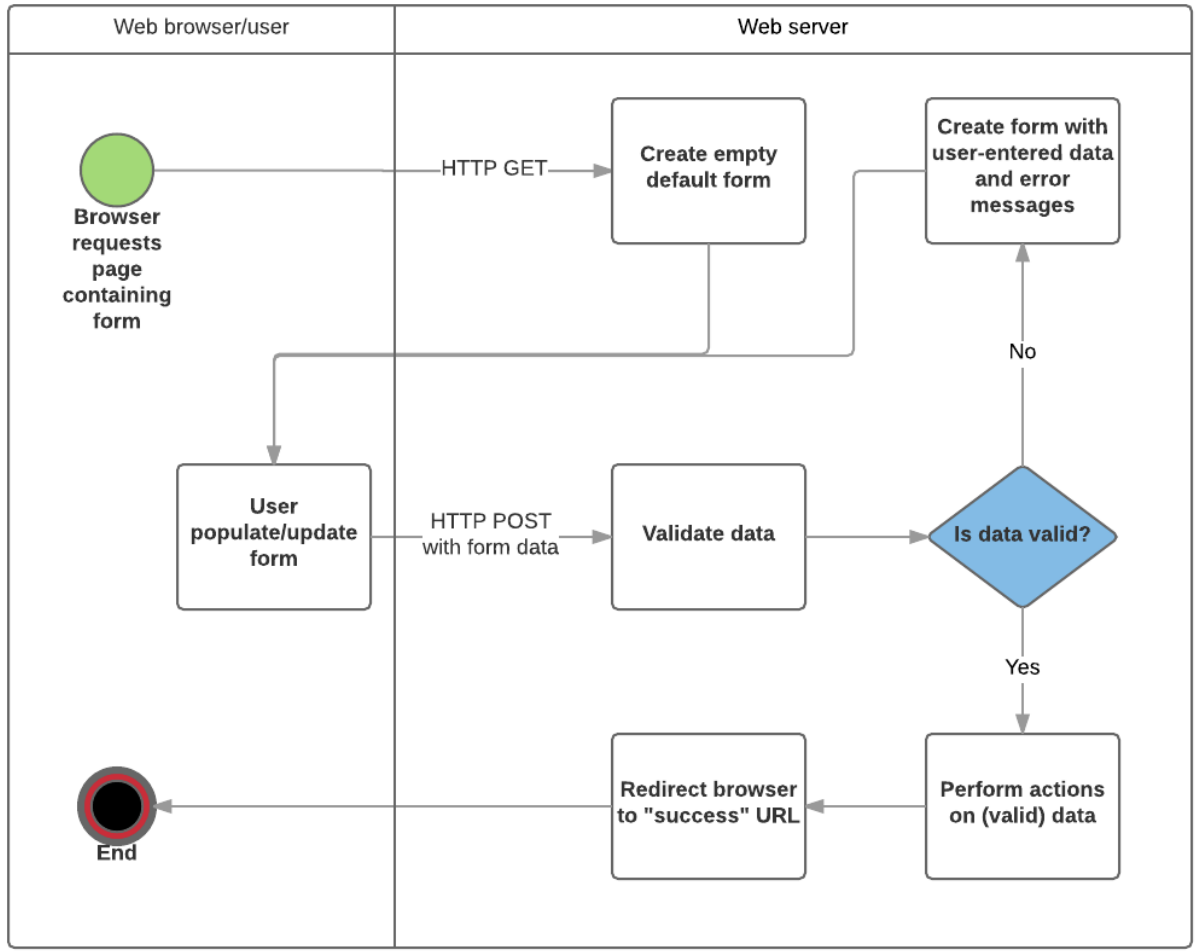
Lukáš Grolig et al.

Outline

- Traditional forms
- React forms
- React-hook-form
- JSON Forms

Basic form

```
<form action="/pet" method="post">
  <label>
    Name:
    <input type="text" name="name" />
  </label>
  <input type="submit" value="Submit" />
</form>
```



Server-side handling

Express.js form handling

```
const { body, validationResult } = require('express-validator');

app.post(
  '/pet',
  // name must be at least 5 chars long, is trimmed and sanitized
  body('name').not().isEmpty().isLength({ min: 5 }).trim().escape(),
  (req, res) => {
    // Finds the validation errors in this request and wraps them in an object with handy func
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    // Do something
  },
);
```

React form (the old way)

```
class NameForm extends React.Component {
  // ...

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text" value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

React form (the old way)

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    alert('A name was submitted: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    // ...
  }
```

React Form using hook

```
import React, { useState } from "react";

export function NameForm(props) {
  const [name, setName] = useState("");

  const handleSubmit = (e) => {
      e.preventDefault();
      alert(`Submitting Name ${name}`)
  }
  return (
    <form onSubmit={handleSubmit}>
      <label>
        Frirst Name:
        <input
          type="text"
          value={name}
          onChange={e => setName(e.target.value)}
        />
      </label>
      <input type="submit" value="Submit" />
    </form>
  );
```

React-hook-form

```
import * as React from "react";
import { useForm } from "react-hook-form";
import Headers from "./Header";

export default function App() {
  const { register, handleSubmit } = useForm();
  const onSubmit = (data) => alert(JSON.stringify(data));

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <Headers />

      <input {...register("firstName")} placeholder="First name" />
      <input {...register("lastName")} placeholder="Last name" />
      <select {...register("category")}>
        <option value="">Select...</option>
        <option value="A">Category A</option>
        <option value="B">Category B</option>
      </select>

      <input type="submit" />
    </form>
```

using Typescript

```
type FormData = {
  firstName: string;
  lastName: string;
};

const { register, setValue, handleSubmit, formState: { errors } }
        = useForm<FormData>();
```

Controlled components

```
import React from "react";
import { useForm, Controller } from "react-hook-form";
import { TextField, Checkbox } from "@material-ui/core";

function App() {
  const { handleSubmit, control, reset } = useForm();
  const onSubmit = data => console.log(data);

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <Controller
        name="MyCheckbox"
        control={control}
        defaultValue={false}
        rules={{ required: true }}
        render={({ field }) => <Checkbox {...field} />}
      />
      <input type="submit" />
    </form>
  );
}
```

Validation

```
import React from "react";
import { useForm } from "react-hook-form";

export default function App() {
  const { register, handleSubmit } = useForm();
  const onSubmit = data => console.log(data);

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input {...register("firstName", { required: true, maxLength: 20 })} />
      <input {...register("lastName", { pattern: /^[A-Za-z]+$/i })} />
      <input type="number" {...register("age", { min: 18, max: 99 })} />
      <input type="submit" />
    </form>
  );
}
```

Accessible form

```
import React from "react";
import { useForm } from "react-hook-form";

export default function App() {
  const { register, handleSubmit, formState: { errors } } = useForm();
  const onSubmit = (data) => console.log(data);

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <label htmlFor="name">Name</label>

      {/* use aria-invalid to indicate field contain error */}
      <input
        id="name"
        aria-invalid={errors.name ? "true" : "false"}
        {...register('name', { required: true, maxLength: 30 })}
      />

      {/* use role="alert" to announce the error message */}
      {errors.name && errors.name.type === "required" && (
        <span role="alert">This is required</span>
      )}
```

Validation using YUP

```
import React from "react";
import { useForm } from "react-hook-form";
import { yupResolver } from '@hookform/resolvers/yup';
import * as yup from "yup";

interface IFormInputs {
  firstName: string
  age: number
}

const schema = yup.object({
  firstName: yup.string().required(),
  age: yup.number().positive().integer().required(),
}).required();

export default function App() {
  const { register, handleSubmit, formState: { errors } } = useForm<IFormInputs>({
    resolver: yupResolver(schema)
  });
  const onSubmit = (data: IFormInputs) => console.log(data);

  return (
```

YUP - more complex object schema

```
import { object, string, number, date, InferType } from 'yup';

let userSchema = object({
  name: string().required(),
  age: number().required().positive().integer(),
  email: string().email(),
  website: string().url().nullable(),
  createdOn: date().default(() => new Date()),
});
```

YUP - string functions

```
string.required(message?: string | function): Schema;
string.length(limit: number | Ref, message?: string | function): Schema;
string.min(limit: number | Ref, message?: string | function): Schema;
string.max(limit: number | Ref, message?: string | function): Schema;
string.matches(regex: Regex, message?: string | function): Schema;
string.matches(regex: Regex, options: { message: string, excludeEmptyString: bool }): Schema;
string.email(message?: string | function): Schema;
string.url(message?: string | function): Schema;
string.uuid(message?: string | function): Schema;
string.ensure(): Schema;
string.trim(message?: string | function): Schema;
string.lowercase(message?: string | function): Schema;
string.uppercase(message?: string | function): Schema;
```

YUP - number functions

```
number.min(limit: number | Ref, message?: string | function): Schema;
number.max(limit: number | Ref, message?: string | function): Schema;
number.lessThan(max: number | Ref, message?: string | function): Schema;
number.moreThan(min: number | Ref, message?: string | function): Schema;
number.positive(message?: string | function): Schema;
number.negative(message?: string | function): Schema;
number.integer(message?: string | function): Schema;
number.truncate(): Schema;
number.round(type: 'floor' | 'ceil' | 'trunc' | 'round' = 'round'): Schema;
```

YUP - number functions

```
date.min(limit: Date | string | Ref, message?: string | function): Schema;
date.max(limit: Date | string | Ref, message?: string | function): Schema;
```

JSON Forms

JSON Forms is a declarative framework for efficiently building form-based web UIs. These UIs are targeted at entering, modifying and viewing data and are usually embedded within an application.

Writing HTML templates and Javascript for data binding by hand is error-prone, especially in applications of reasonable size.

JSON Forms utilizes the capabilities of JSON and JSON schema and provides a simple and declarative way of describing forms. Forms are then rendered with a UI library or framework, e.g. React or Angular.

JSON Forms

Any UI is defined by using two schemata:

- the data/JSON schema defines the underlying data to be shown in the UI (objects, properties, and their types);
- the UI schema defines how this data is rendered as a form, e.g. the order of controls, their visibility, and the layout.

Data schema

```
{
  "type":"object",
  "properties":{
    "name":{
      "type":"string"
    },
    "description":{
      "type":"string"
    },
    "done":{
      "type":"boolean"
    },
    "rating":{
      "type":"integer"
    }
  },
  "required":[
    "name"
  ]
}
```

UI schema


```
{
  "type":"VerticalLayout",
  "elements":[
    {
      "type":"Control",
      "scope":"#/properties/name"
    },
    {
      "type":"Control",
      "scope":"#/properties/description",
      "options":{
        "multi":true
      }
    },
    {
      "type":"Control",
      "label":"Rating",
      "scope":"#/properties/rating"
    },
    {
      "type":"Control",
      "label":"Done?"
```

Layouting

```
{
  "type": "Group",
  "label": "My Group",
  "elements": [
    {
      "type": "HorizontalLayout",
      "elements": [
        {
          "type": "VerticalLayout",
          "elements": [
            {
              "type": "Control",
              "label": "Name",
              "scope": "#/properties/name"
            },
            {
              "type": "Control",
              "label": "Birth Date",
              "scope": "#/properties/birthDate"
            }
          ]
        }
```

Enums

```
{
  "type": "object",
  "properties": {
    "comments": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "date": {
            "type": "string",
            "format": "date"
          },
          "message": {
            "type": "string",
            "maxLength": 5
          },
          "enum": {
            "type": "string",
            "enum": [
              "foo",
              "bar"
            ]
```

Rules

```
{
  "type": "object",
  "properties": {
    "name": {
      "type": "string"
    },
    "dead": {
      "type": "boolean"
    },
    "kindOfDead": {
      "type": "string",
      "enum": [
        "Zombie",
        "Vampire",
        "Ghoul"
      ]
    },
    "vegetables": {
      "type": "boolean"
    },
    "kindOfVegetables": {
      "type": "string"
```

```
{
  "type": "VerticalLayout",
  "elements": [
    {
      "type": "Control",
      "label": "Name",
      "scope": "#/properties/name"
    },
    {
      "type": "Group",
      "elements": [
        {
          "type": "Control",
          "label": "Is Dead?",
          "scope": "#/properties/dead"
        },
        {
          "type": "Control",
          "label": "Kind of dead",
          "scope": "#/properties/kindOfDead",
          "rule": {
            "effect": "ENABLE",
            "condition": {
              "scope": "#/properties/dead",
```

DEMO TIME

That's it for today!