BTI425/WEB422 - Web Programming for Apps and Services

Lecture Recap:

Week 6 – Forms Introduction

Agenda

- React Forms
- Validation

HTML Form - Review

- Form elements (/controls/fields):
  - Text-Boxes
    - Single line: <input type=text /> or type= number, email, date, color, tel, …
    - Multiple lines: <textarea></textarea>
  - Selections
    - Single selection:
      - <input type=radio /> in group
      - <select> without multiple attribute, plus <option>s
    - Multiple selections:
      - <input type=checkbox /> in group — not used as multiple selection in Angular form
      - <select> with multiple attribute, plus <option>s
  - Buttons and others

Data types in Angular/React

- Form elements vs data types (in React App):

Element type	Form element	Data type
Textbox	<input type='text' />	string
"	<textarea></textarea>	string
Selection	<input type='radio' /> group	string
"	<select> + <option>s	string
"	<input type='checkbox' /> group	not used
"	a single <input type='checkbox' />	boolean
"	<select multiple> + <option>s	string[]

JavaScript Object - Review

- Property Accessors
  - Dot Notation
    - Syntax: Object.property
    - Example:

```
var person = { firstName: "John", lastName : "Doe" };
var lname = person.lastName   // "Doe"
```

  - Bracket Notation
    - Syntax: Object['property']
    - Example:

```
var person = { firstName: "John", lastName : "Doe" };
var lname = person["lastName"];  // "Doe"
```

- Copy JS Objects
  - using spread ... – shallow copy

```
var p2 = { …person };
```

  - using JSON – deep copy

```
var p2 = JSON.parse(JSON.stringify(person));
```

React Forms

- Common web forms:
  - <form> element: setting action, method, and optional enctype attributes:
    - method="POST", action="URL", which is a special route on a server
  - the body of the http request contains the "urlencoded" data of the form, in querystring-like format
  - the server is responsible to use the "urlencoded" middleware, i.e., built-in "body-parser" to decode/parse this data in object -> to be persisted to a data store
- Forms in a front-end app, e.g., React app:
  - Explicitly handling the "submit" event for the form and preventing the default action of automatically sending a HTTP "POST" request
  - Obtaining updated form values and generating JSON formatted data to reflect the current values of the form fields and submitting the form an AJAX/HTTP request.

Controlled Components in React

- User input vs React component state
  - In HTML, form elements such as <input>, <textarea>, and <select> typically maintain their own state and update it based on user input.
  - In React, the state of all corresponding form elements are maintained by the state property of the component and mutable/updated with setState() only.
- Controlled Component: the component is able to make the React state be the 'single source of truth':
  - The state always reflects the users' input
  - The state is mutable and always rendered on the html form
- Example - the SimpleFrom component

React Hook Form

- React Hook Form - the UserFrom component
  - Registering Multiple Form Controls
  - For handling submit and get/set values:

    const { register, handleSubmit, setValue, getValues, watch } = useForm(…)

  - "Watch" Form Values // different from "getValues"

```
const watchUserName = watch('userName');
const watchAllFields = watch();
```

in JSX/UI

```
<p>User Name: {watchUserName}</p>
<p>All Fields: {JSON.stringify(watchAllFields)}</p>
```

Form Validation

- Using "browser native validation" attributes:
  - required, min, max, minLength, maxLength, pattern

- Adding Validation Rules to "register"

  e.g. `<input {...register("lastName", { pattern: /^[A-Za-z]+$/i })} /><br /><br />`

- Custom Validation Rules

  e.g. `type="number" {...register("age", { min: 18, max: 99, validate: { onlyEven: v => v % 2 == 0 } })} />`

- Showing Errors

  `const { ……, formState: { errors } } = useForm({...});`

  … …

The End