FULLSTACK DEVELOPMENT (MMT-B2020)

FHS (TMAYRHOFER.LBA@FH-SALZBURG.AC.AT)

REACT HOOKS RECAP

“Hooks allow you to reuse stateful logic without changing your component hierarchy. React Docs”

REACT HOOKS RECAP

- Introduced recently to reduce boilerplate
- Makes it possible to use state in functional components
  - Previously one had to convert between functional/class components when state introduced
- hooks are prefixed with use
- Can't be called inside loops, conditions or nested

REACT HOOKS RECAP
USESTATE

```
const App = () => {
  const [count, setCount] = useState(0)
  const handleIncrement = () => setCount(count + 1)

  return (
    <div>
      <div>{count}</div>
      <button onClick={handleIncrement}>Increment by 1</button>
    </div>
  )
}
```

REACT HOOKS RECAP
EXTRACT INTO CUSTOM HOOK

```
const useCounter = () => {
  const [count, setCount] = useState(0);
  const handleIncrement = () => setCount(count + 1);
  return { count, handleIncrement };
}

const App = () => {
  const {count,handleIncrement} = useCounter();

  return (
    <div>
      <div>{count}</div>
      <button onClick={handleIncrement}>Increment by 1</button>
    </div>
  );
}
```

REACT STATE
UNIDIRECTIONAL DATAFLOW

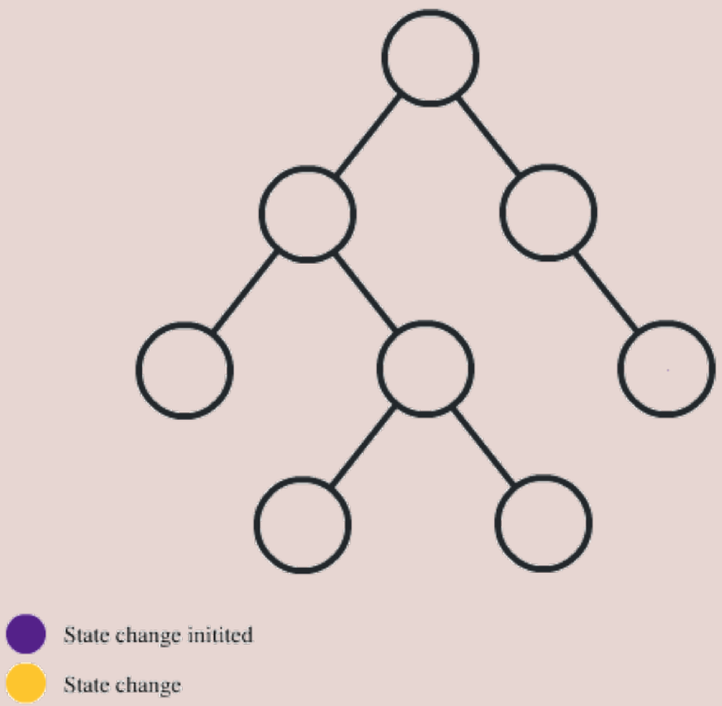
» Props only flow from parent to children

» Parent is responsible to update data

  » might provide callbacks to do so

» set state rerenders all children of component

REACT STATE
UNIDIRECTIONAL DATAFLOW



“Source”

VIRTUAL DOM

» makes DOM updates faster

» after setState subtree is rerendered in memory

» compares DOM to in memory representation

» applies DOM changes when needed

FORMS WITH REACT HOOKS

```
const App = () => {
  const [username, setUsername] = useState('');
  //                              ^^^^^^^^^^^^
  // define a new state with an initial value of empty string

  return (
    <div>
      <input onChange={(evt) => setUsername(evt.target.value)} value={username}>
      { /*                                  ^^^^^^^^^^^^^^^^^^ */}
      { /* set the state of the username */}
      <button onClick={() => console.log({ username })}>Submit form</button>
    </div>
  );
}
```

FORMS IN REACT

CONTROLLED VS UNCONTROLLED COMPONENTS

CONTROLLED COMPONENTS

- HTML form elements maintain own state
  - eg. input, textarea, ...
- React usually keeps state in their own components
  - component state/HTML state can get out of sync
- in controlled components react is the single source of truth

CONTROLLED COMPONENTS

» React has ownership of state

  » result: typing in the component does not have any effect

```
const Input = () => {
  return <input name="username" value="" />
  //                            ^^^^^^^^
  // if value is defined input becomes controlled
}
```

CONTROLLED COMPONENTS

```
const Input = () => {
  const [username, setUsername] = useState('')
  return <input
    name="username"
    onChange={(evt) => setUsername(evt.target.value)}
    //                 ^^^^^^^^^^^^
    // 1) whenever onChange setUsername is called with new value
    value={ username }
    // 2) setUsername triggers a rerender with the new username
  />
}
```

UNCONTROLLED COMPONENTS

» the browser keeps ownership of form state

```
const Input = () => {
  return <input name="username" />
  //                ^^^^^^^^
  // if no value attribute is defined on an input
  // the state is uncontrolled and managed by the
  // browser
}
```

HANDLE ERRORS IN COMPONENTS

```
const SignUpForm = ({ onSubmit }) => {
  const [username, setUsername] = useState('')

  return (
    <form>
      <input
        name="username"
        onChange={(evt) => setUsername(evt.target.value)}
        value={ username }
      />
      { username.length === 0 && ( // when username is 0 display error
        <span>Username can't be blank</span>
      )}
      <button type="submit">Sign up</button>
    <form/>
  )
}
```

TASK (15 MINUTES)

- adapt your sign-up form
  - convert your components to controlled components
  - display error messages when username or password is blank
- Do you find any issues in your code?

DO YOU SEE ANY ISSUES WITH THE CODE

```
const SignUpForm = ({ onSubmit }) => {
  const [username, setUsername] = useState('')

  return (
    <form>
      <input
        name="username"
        onChange={(evt) => setUsername(evt.target.value)}
        value={ username }
      />
      { username.length === 0 && (
        <span>Username can't be blank</span>
      )}
      <button type="submit">Sign up</button>
    <form/>
  )
}
```

DO YOU SEE ANY ISSUES WITH THE CODE

 DON'T SPOIL YOURSELF AND LOOK AT THE NEXT SLIDES 

DO YOU SEE ANY ISSUES WITH THE CODE

I MEAN REALLY, STOP HERE 😄

DO YOU SEE ANY ISSUES WITH THE CODE

 SERIOUSLY 

DO YOU SEE ANY ISSUES WITH THE CODE

- errors are shown even if a user didn't focus the input
- form can be submitted even if it contains errors
  - sign-in button is not disabled
- adding complex validations is tedious

FORM LIBRARIES WHICH MAKE YOUR LIFE EASIER

- there are multiple libraries which help with validation
  - formik
  - react-hook-form
  - react final form

FORMIK

» Form library which can be used with hooks

» uses controlled components

» npm install formik yup

FORMIK
EXAMPLE

```
import { useFormik } from "react-hook-form";

const SignInForm = () => {
  const formik = useFormik({
    initialValues: { username: '' },
    onSubmit: values => console.log(values),
  });

  return (
    <form onSubmit={formik.handleSubmit}>
      <input
        name="username"
        onChange={formik.handleChange}
        value={formik.values.username}
      />
      {/* ... */}
    </form>
  )
}
```

FORMIK
WITH ERRORS

```
import { useFormik } from "react-hook-form";
import {object, string} from 'yup'

const validationSchema = object({
  username: string().min(3)
})

const SignInForm = () => {
  const formik = useFormik({
    initialValues: { username: '' },
    validationSchema: validationSchema,
    // verify form with schema  ^^^^^^^^^^

  });

  return (
    <form onSubmit={formik.handleSubmit}>
      <input
        name="username"
        onChange={formik.handleChange}
        value={formik.values.username}
      />
      { formik.errors.username }
      {/* display the error */}
    </form>
  )
}
```

TASK 20 MINUTES

» convert your Sign Up form to use react hooked forms

ROUTING

REACT ROUTER

» dynamic routing library for

» react native

» react web

» Documentation

INSTALLATION

```
npm install react-router-dom --save
```

USAGE

```
import { BrowserRouter , Routes, Route, Link } from 'react-router-dom'
import Homepage from './components/homepage'
import SignIn from './components/sign-in'

const App = () => {
  return (
    <BrowserRouter> { /* creates a new routing context */ }
      <Routes> { /* render only one route */ }
        { /* define routes and pass component as element prop to the route */ }
        <Route path='/sign-in' element={<SignIn />} />
        <Route path='/' element={<Homepage />} />
        { /* if no route matches redirect to 'Homepage' */ }
        <Route path="*" element={<Navigate to='/' />} />
      </Routes>
    </BrowserRouter>
  )
}
```

DEFINE NESTED ROUTES

```
import { BrowserRouter, Routes, Route, Navigate } from 'react-router-dom'

const UserProfile = () => {
  const params = useParams();
//               ^^^^^^^^^^
// access to dynamic params from URL
  return <h1>User {params.userId}</h1>;
}

const Routes = () => (
  <BrowserRouter>
    <Routes>
      <Route path='user/:userId' element={<UserProfile />}/>
      {/*                ^^^^^^ */}
      {/* define dynamic URL segment */}
      {/* .... */}
    </Routes>
  </BrowserRouter>
)
```

DEFINE NESTED ROUTES

```
import { BrowserRouter, Routes, Route, Navigate } from 'react-router-dom'
// ... other imports

const Routes = () => (
  <BrowserRouter>
    <Routes>
      { /* define nested routes */ }
      <Route path='user'>
        <Route path='profile' element={<Profile />} />
        <Route path="*" element={<Navigate to='profile' />} />
        { /* redirects to user/profile */ }
      </Route>
      {/* .... */}
    </Routes>
  </BrowserRouter>
)
```

ADD LINKS FROM HTML

```
import { Link } from 'react-router-dom'

const Routes = () => (
  <nav>
    <Link to='/'>Home</Link>
    <Link to='/sign-in'>Sign in</Link>
  </nav>
)
```

ADD REDIRECTS FROM JS

```
 import {useNavigate} from 'react-router-dom'

const Profile = () => {
  const navigate = useNavigate()
//                 ^^^^^^^^^^^^
// get access to navigate function

  return <button onClick={() => navigate('/') }>Logout</button>
}
```

TASK 20 MINUTES

- Start the application npm run start
  - npm install react-router-dom
  - add 2 routes
  - sign-up/
    - renders the SignUp component
  - sign-in/
    - renders a SignIn component (needs to be built)

FEEDBACK

» Questions: tmayrhofer.lba@fh-salzburg.ac.at

» https://s.surveyplanet.com/x1ibwm85