

FULLSTACK DEVELOPMENT (MMT-B2018)

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>
  );
}
```

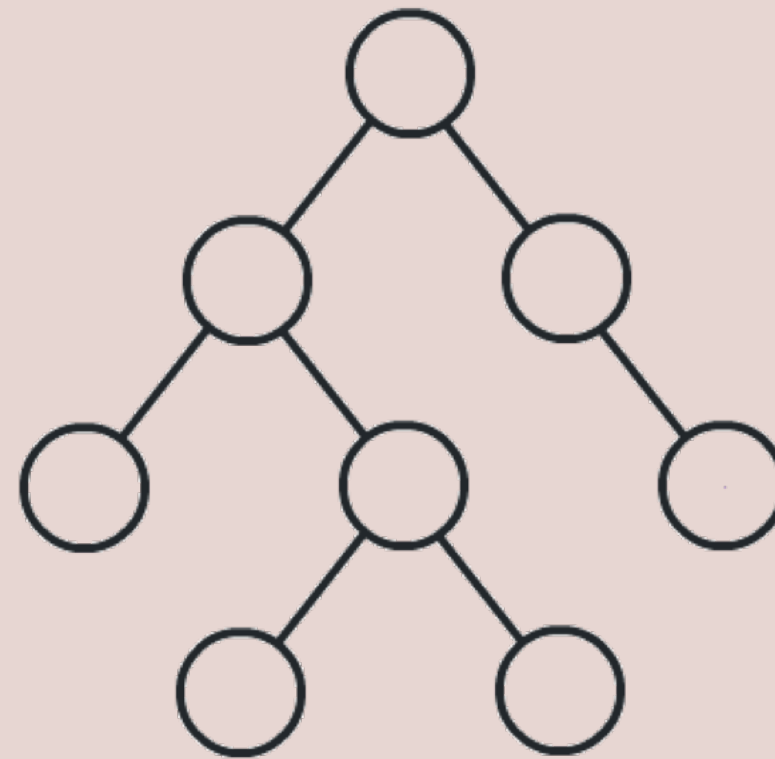
REACT STATE

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



- State change initiated
- State change

“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 as Router, Route, Switch, Redirect } from "react-router-dom";
import Homepage from './components/homepage'
import SignIn from './components/sign-in'
```

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

ROUTE PRIORITY (WITHOUT EXACT)

```
// path === "/" => renderes Homepage  
// path === "/sign-in" => renderes Homepage  
const Routes = () => (  
  <Switch>  
    <Route path="/" component={Homepage} />  
    <Route path="/sign-in" component={SignIn} />  
  </Switch>  
)
```


ROUTE PRIORITY (WITHOUT EXACT)

```
// path === "/" => renderes Homepage  
// path === "/sign-in" => renderes SignIn  
const Routes = () => (  
  <Switch>  
    <Route path='/sign-in' component={SignIn} />  
    <Route path="/" component={Homepage} />  
  </Switch>  
)
```

ROUTE PRIORITY (WITH EXACT)

```
// path === "/" => renderes Homepage  
// path === "/sign-in" => renderes sign-in  
const Routes = () => (  
  <Switch>  
    <Route exact path="/" component={Homepage} />  
    <Route exact path="/sign-in" component={SignIn} />  
  </Switch>  
)
```

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 { withRouter } from 'react-router-dom'

const SignIn = withRouter(({ history }) => {
  const onSubmit = (evt) => {
    evt.preventDefault()
    history.push('/')
  }

  return (
    <form onSubmit={onSubmit}>
      { /* ... */ }
    </form>
  )
})
```

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)

HOMEWORK

HOMEWORK 1

- » Add formik to your form components
- » onSubmit => values should be logged to the console
 - » UserSignIn -> onSubmit => { username, password }
 - » UserSignUp -> onSubmit => { username, password }
 - » MoneyTransactionCreate
 - » users => { id, name }
 - » onSubmit => { debtorId, creditorId, amount }

FEEDBACK

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

» <https://s.surveyplanet.com/x1ibwm85>