28-short-lnk-app video

Controlled and Uncontrolled Inputs

- lets go to AddLink.js to make this is a controlled component
- first we are going to be setting up state, so we need our constructor for that
- in our state we are going to keep track of our url value
- it going to start with an empty string, but for now as an example we are going to give it a value

```
import React from 'react';

import { Meteor } from 'meteor/meteor';


export default class AddLink extends React.Component{

    constructor(props){

        super(props);

        this.state = {

            url: 'Jose was here'

        }

    }
```

- **now** our form down below currently doesn't have the value attribute, so we are going to be adding that
- and the value for the value attribute is going to be our url that we just created

```
import React from 'react';

import { Meteor } from 'meteor/meteor';


export default class AddLink extends React.Component{

    constructor(props){

        super(props);

        this.state = {

            url: 'Jose was here'

        }

    }

    onSubmit(e){

        const url = this.refs.url.value.trim();

        e.preventDefault();


        if(url){
            Meteor.call('links.insert', url);

            this.refs.url.value = '';


        }

    }

    render(){

        return(

            <div>

                <p>Add Link</p>

                <form onSubmit={this.onSubmit.bind(this)}>

                    <input type="text" ref="url" placeholder="URL" value={this.state.url}/>

                    <button>Add Link</button>

                </form>

            </div>

        );

    }

}
```

- now over in the browser you should be able to see jose was here in the text field, and you won't be able to delete it was well - you will get a warning in the console about providing an onChange handler
- so when creating controlled components we have to provide onChange and value
- so lets register an onChange handler to that input
- and that is going to fire up method in our case we will call it onChange- we then are going to create that method up above

```
    render(){

        return(

            <div>

                <p>Add Link</p>

                <form onSubmit={this.onSubmit.bind(this)}>

                    <input type="text" ref="url" placeholder="URL" value={this.state.url} onChan
ge={this.onChange.bind(this)}/>

                    <button>Add Link</button>

                </form>

            </div>

        );

    }

}
```

- we then are going to create that method, and it's going to take an argument so by using that argument we can target that value
- but we are going to be changing the state everytime fireup that event so we need a setState method for that as well, and we know that setState take on argument, and that is an object
- we are going to use the trim() method, so that way the user is not allowed to use spaces, and this is perfectly ok on this situation because we are just entering urls, and urls don't have spaces
- but we are going not going to use, we are going to let Meteor do the validation for us

```
import React from 'react';

import { Meteor } from 'meteor/meteor';


export default class AddLink extends React.Component{

    constructor(props){

        super(props);

        this.state = {

            url: 'Jose was here'

        }

    }

    onSubmit(e){

        const url = this.refs.url.value

        e.preventDefault();


        if(url){

            Meteor.call('links.insert', url);

            this.refs.url.value = '';


        }

    }

    onChange(e){

        this.setState({

           url: e.target.value.trim()

        })

    }


    render(){

        return(

            <div>

                <p>Add Link</p>

                <form onSubmit={this.onSubmit.bind(this)}>
```

```
                    <input type="text" ref="url" placeholder="URL" value={this.state.url} onChan
ge={this.onChange.bind(this)}/>
                    <button>Add Link</button>
                </form>
            </div>
        );
    }
}
```

- next up we are going to be changing the onSubmit method
- we are no longer going to fetch the url off of the DOM, the state is taking care of that so we are going to make that const url equal to the state

```
export default class AddLink extends React.Component{
    constructor(props){
        super(props);
        this.state = {
            url: 'Jose was here'
        }
    }
    onSubmit(e){
        const url = this.state.url
        e.preventDefault();

        if(url){
            Meteor.call('links.insert', url);
            this.refs.url.value = '';


        }
    }
```

- we can also use ES6 distructuring for this, and switch that line with this.

```
    onSubmit(e){
        const { url } = this.state
        e.preventDefault();

        if(url){
            Meteor.call('links.insert', url);
            this.refs.url.value = '';


        }
    }
```

- in the Meteor.call we are going to be adding a third argument, which will be taking two arguments, error and res
- and then we will do an if statement, if there is no error we are going to clear the value, if there is one we are going to leave the value on the field so the user can fix his typo
- and we are going to be using setState again and we are going to pass our url to equal an empty string
- so now we can delete the refs line of code because is no longer necessary as well as the ref attribute over in the input tag

```
    onSubmit(e){
        const { url } = this.state
        e.preventDefault();


        if(url){
            Meteor.call('links.insert', url, (err, res)=>{
                if(!err){
                    this.setState({url:''});
                }
            });
        }
    }
    onChange(e){
        this.setState({
           url: e.target.value
        })
    }


    render(){
        return(
            <div>
                <p>Add Link</p>
                <form onSubmit={this.onSubmit.bind(this)}>
                    <input type="text" placeholder="URL" value={this.state.url} onChange={this.o
nChange.bind(this)}/>
```

```
                    <button>Add Link</button>
                </form>
            </div>
        );
    }
}
```

- over in our constructor lets delete the line jose was here to make our url state an empty string

```
import React from 'react';
import { Meteor } from 'meteor/meteor';

export default class AddLink extends React.Component{
    constructor(props){
        super(props);
        this.state = {
            url: ''
        }
    }
    onSubmit(e){
        const { url } = this.state
        e.preventDefault();

        if(url){
            Meteor.call('links.insert', url, (err, res)=>{
                if(!err){
                    this.setState({url:''});
                }
            });
        }
    }
    onChange(e){
        this.setState({
           url: e.target.value
        })
    }
    render(){
        return(
            <div>
                <p>Add Link</p>
                <form onSubmit={this.onSubmit.bind(this)}>
                    <input type="text" placeholder="URL" value={this.state.url} onChange={this.o
nChange.bind(this)}/>
                    <button>Add Link</button>
                </form>
            </div>
        );
    }
}
```

- we are going to be using this tecnique in the LinksListFilters.js
- first lets change the arrow function to the es6 class base component

```
import React from 'react';
import { Session } from 'meteor/session';

export default class LinksListFilters extends React.Component{
    render(){
        return (
            <div>
            <label>
                <input type="checkbox" onChange={(e)=>{
                    console.log(e.target.checked);
                    Session.set('showVisible', !e.target.checked);
                }}/>
                show hidden links
            </label>
        </div>
        );
    }
}
```

- lets add our construtor, and pass the props
- and now we can add our default state, showVisible will be false

- now instead of using value like we used in the text field, we are going to be using checked, and this takes a boolean and in our case we are going to use this.state.showVisible but in our case we want to check our box when the showVisible is false so we ! in front of it

```
import React from 'react';
import { Session } from 'meteor/session';

export default class LinksListFilters extends React.Component{
    constructor(props){
        super(props)
        this.state = {
            showVisible : false
        }
    }
    render(){
        return (
            <div>
            <label>
                <input type="checkbox" checked={!this.state.showVisible} onChange={(e)=>{
                    console.log(e.target.checked);
                    Session.set('showVisible', !e.target.checked);
                }}/>
                show hidden links
            </label>
        </div>
        );
    }
}
```

- now by default let switch the showVisible to true

```
import React from 'react';
import { Session } from 'meteor/session';

export default class LinksListFilters extends React.Component{
    constructor(props){
        super(props)
        this.state = {
            showVisible : true
        }
    }
    render(){
        return (
            <div>
            <label>
                <input type="checkbox" checked={!this.state.showVisible} onChange={(e)=>{
                    console.log(e.target.checked);
                    Session.set('showVisible', !e.target.checked);
                }}/>
                show hidden links
            </label>
        </div>
        );
    }
}
```

- our challenge will be to use two life cycle components, componentDidMount and ComponentWillUnmount
- in our componentDidMount is to setup a tracket.autorun call
- first let import Tracker
- we created a variable called tracker only because we want to use in ComponentWillUnmount

```
import React from 'react';
import { Session } from 'meteor/session';
import { Links } from '../api/links'
import { Tracker } from 'meteor/tracker';


export default class LinksListFilters extends React.Component{
    constructor(props){
        super(props);
        this.state = {
            showVisible : true
        }
    }
    componentDidMount(){
        this.tracker = Tracker.autorun(()=>{
            this.setState({
                showVisible: Session.get('showVisible')
            });
        })
    }
    componentWillUnmount(){
        this.tracker.stop()
    }
```

-