04-Short-Link-App

Creating User Accounts

- we are going to be adding a new module called accounts-password
- go to your terminal and type the following

```
meteor add accounts-password
```

- to learn more about these module google the meteor documentation
- head over to Signup.js and import the accounts-password module

```
import React from 'react';
import { Link } from 'react-router';
import { Accounts } from 'meteor/accounts-base';

export default class Signup extends React.Component{
    constructor(props){
        super(props);
        this.state = {
            error: ' '
        };

    }
    onSubmit(e){
        e.preventDefault();
        this.setState({
            error: 'something went wrong'
        })
    }
```

- now lets put the Accounts to work
  - we are going to be using the createUser() method, and it takes 2 arguments , the 1st is an object, which for us is going to be email with an empty string , and password with an empty string, and the 2nd argument is a callback function, this callback function gets called with any error
  - and now all we have to do is grab the email and password from the values the form gets passed in the form below, in the past we would use something like this e.target.email.value but a React way to target a specific element.
  - React uses refs(references)
  -

```
    onSubmit(e){
        e.preventDefault();

        Accounts.createUser({email: ' ', password: ' '}, (err)=>{

        })
        // this.setState({
        //     error: 'something went wrong'
        // })
    }
    render(){
        return (
            <div>
                <h1>Joint Short Lnk</h1>

                {this.state.error ? <p>{this.state.error}</p>: undefined}

                <form onSubmit={this.onSubmit.bind(this)}>
                    <input type="email" ref="email" name="email" placeholder="Email"/>
                    <input type="password" ref="password" name="password"
placeholder="Password"/>
                    <button>Create Account</button>
                </form>



                <Link to="/">Already have an account?</Link>
            </div>
        );
    }
}
```

- now we can access those refs pulling out the values

```
    onSubmit(e){
        e.preventDefault();
```

```
        let email = this.refs.email.value.trim();
        let password = this.refs.password.value.trim();

        Accounts.createUser({email: ' ', password: ' '}, (err)=>{

        })
        // this.setState({
        //     error: 'something went wrong'
        // })
    }
    render(){
        return (
            <div>
                <h1>Joint Short Lnk</h1>

                {this.state.error ? <p>{this.state.error}</p>: undefined}

                <form onSubmit={this.onSubmit.bind(this)}>
                    <input type="email" ref="email" name="email" placeholder="Email"/>
                    <input type="password" ref="password" name="password"
placeholder="Password"/>
                    <button>Create Account</button>
                </form>
```

- notice that we used the trim(), and that is just get rid of any spaces the user used by accident when typing his information in
- now that we have email and password we can now pass it to createUser, and for this we are going to use the ES6 shorthand

```
    onSubmit(e){
        e.preventDefault();

        let email = this.refs.email.value.trim();
        let password = this.refs.password.value.trim();

        Accounts.createUser({email, password}, (err)=>{

        })
```

```
        // this.setState({
        //     error: 'something went wrong'
        // })
    }
```

- and now lets finish the error function by console login it
- now head over to the browser and type in some random email and password
- and on the console, lets check if we actually submitted that data, go ahead an type

```
require('meteor/meteor').Meteor.userId()
"A5pvXqou7S6ykPrhF"
```

- if we run the user() method

```
require('meteor/meteor').Meteor.user()
{_id: "A5pvXqou7S6ykPrhF", emails: Array(1)}
```

- we are now going to see what happens if we are logged out
- and that is available on Accounts- so in our console type

```
require('meteor/accounts-base')
{AccountsTest: {…}, Accounts: AccountsClient, Symbol(__esModule): true, AccountsClient: ƒ}
Accounts
:
AccountsClient {_options: {…}, connection: Connection, users: M…o.Collection, _onLoginHook: Hoo
k, _onLoginFailureHook: Hook, …}
AccountsClient
:
ƒ AccountsClient(options)
AccountsTest
:
{attemptToMatchHash: ƒ}
Symbol(__esModule)
:
true
__proto__
:
Object
```

- and now we want to access the Accounts method so we type the following
- and from here we are just interested in

```
require('meteor/accounts-base').Accounts
AccountsClient {_options: {…}, connection: Connection, users: M…o.Collection, _onLoginHook: Hoo
k, _onLoginFailureHook: Hook, …}
LOGIN_TOKEN_EXPIRES_KEY
:
"Meteor.loginTokenExpires"
LOGIN_TOKEN_KEY
:
"Meteor.loginToken"
USER_ID_KEY
:
"Meteor.userId"
changePassword
:
ƒ (oldPassword, newPassword, callback)
connection
:
Connection {onReconnect: null, _stream: L…t.ClientStream, _lastSessionId: "8tFhTpaPB7qojmWFX", _
versionSuggestion: "1", _version: "1", …}
createUser
:
ƒ (options, callback)
forgotPassword
:
ƒ (options, callback)
resetPassword
:
ƒ (token, newPassword, callback)
users
:
Mongo.Collection {_transform: null, _connection: Connection, _collection: LocalCollection, _nam
e: "users", _makeNewID: ƒ, …}
verifyEmail
:
```

ƒ (token, callback)

_accountsCallbacks

:

{}

_autoLoginEnabled

:

true

_hashPassword

:

ƒ (password)

_lastLoginTokenWhenPolled

:

"UGIJyE93DROuJ5KOX7wt95tDhKrvEYQXl4RemlF9neR"

_loggingIn

:

ReactiveVar {curValue: false, equalsFunc: undefined, dep: T…r.Dependency}

_loggingOut

:

ReactiveVar {curValue: false, equalsFunc: undefined, dep: T…r.Dependency}

_loginFuncs

:

{}

_loginServicesHandle

:

{subscriptionId: "tEP6zAtqvs7NPDGcB", stop: ƒ, ready: ƒ}

_onLoginFailureHook

:

Hook {nextCallbackId: 0, callbacks: {…}, bindEnvironment: false, exceptionHandler: "onLoginFailu
re callback"}

_onLoginHook

:

Hook {nextCallbackId: 0, callbacks: {…}, bindEnvironment: false, exceptionHandler: "onLogin call
back"}

_onLogoutHook

:

Hook {nextCallbackId: 0, callbacks: {…}, bindEnvironment: false, exceptionHandler: "onLogout cal
lback"}

```
_options
:
{}
_pageLoadLoginAttemptInfo
:
null
_pageLoadLoginCallbacks
:
[]
_pollIntervalTimer
:
3
_reconnectStopper
:
{stop: ƒ}
__proto__
:
AccountsCommon
```

- and now if we use the logout() method like so

```
require('meteor/accounts-base').Accounts.logout()
undefined
```

- and now if we try to run one of our previous methods we will get null

```
require('meteor/meteor').Meteor.userId()
null
```

- samething with the user() method, we get null again

```
require('meteor/meteor').Meteor.user()
null
```

- now lets check out the user we created, go over to the terminal, be sure to be in your directory and type
- and start up mongo

```
meteor mongo
```

- and once your mongo is running type

```
meteor:PRIMARY> db.users.find()
{ "_id" : "A5pvXqou7S6ykPrhF", "createdAt" : ISODate("2017-10-14T17:36:01.692Z"), "services" : {
 "password" : { "bcrypt" : "$2a$10$Nw.5lolOiptOuEmWRDL9HeWAxUmw48UMHj94OSvJzpVUTymWyjg1O" }, "re
sume" : { "loginTokens" : [ ] } }, "emails" : [ { "address" : "jose@jjmendoza.com", "verified" :
 false } ] }
```

- and as you can see get back the email we just typed in