08-Short-Link-App

New User Validation - Video

- what we want to do first is to validate the user object just before it gets created
- so we are going to do this in the server/main.js and we will be using the Accounts class,
- it takes a function as an argument
- and lets also import the named export Accounts
- this method will stop the process if something is invalid
- so for this example we are going to allow it to get created, so all we need to do is return true
- for this we will just explore the ***user*** object

```
import { Meteor } from 'meteor/meteor';
import SimpleSchema from 'simpl-schema';
import { Accounts } from 'meteor/accounts-base';

Meteor.startup(() => {
  // code to run on server at startup

  Accounts.validateNewUser((user)=>{
    console.log('this is the user ', user);
    return true;
  });
```

- before continuing we are going to disable the built in browser validation, all we have to do is add a noValidate to our forms in Login.js and Signup.js

```
                <h1>Join Short Lnk</h1>

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

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

-

```
                <h1>Short Lnk</h1>

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

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

                login form here
```

- now head over to the browser and create a new user and typing some random email and password
- now head over to the terminal and this is what you will see

```
I20171016-15:47:39.657(-7)? this is the user  { createdAt: Mon Oct 16 2017 15:47:39 GMT-0700 (PD
T),
I20171016-15:47:39.658(-7)?   _id: 'YA3pYxLhA5unrWRK5',
I20171016-15:47:39.659(-7)?   services: { password: { bcrypt: '$2a$10$eareZjNwaMjLdlH7Wu4c4ufKi0
xulIeKSbNCCG486kpSvGSGvz3si' } },
I20171016-15:47:39.660(-7)?   emails: [ { address: 'jose@example.com', verified: false } ] }
```

- lets go back to server/main.js and we are going to take the email information from the user object
- lets create a const variable
- if you notice the array[0] is because we want to grab just one item, and we want to access the address property

```
  Accounts.validateNewUser((user)=>{
    const email = user.emails[0].address
    console.log('this is the user ', user);
    return true;
  });
```

- all we have to do is create a schema that validates the email
- as you notice we use a different way to create a new SimpleSchema, this is just a different way without a variable, and also we used the ES6 shorthand when using email

```
  Accounts.validateNewUser((user)=>{
    const email = user.emails[0].address


    new SimpleSchema({
      email:{
        type: String,
        regEx: SimpleSchema.RegEx.Email
      }
    }).validate({email});
    console.log('this is the user ', user);
    return true;
  });
```

- now we can go test this out
- first try to create a new user by using a bogus email on the browser
- and we get an internal server error
- but if we go over to the terminal we will see a ***'createUser' ClientError: Email must be a valid email address***
- we are going to be exploring a little bit with Meteor errors
- we will be using a try and catch clause
- first go back to the server main.js

```
import { Meteor } from 'meteor/meteor';
import SimpleSchema from 'simpl-schema';
import { Accounts } from 'meteor/accounts-base';


Meteor.startup(() => {
  // code to run on server at startup


  Accounts.validateNewUser((user)=>{
    const email = user.emails[0].address


    new SimpleSchema({
      email:{
        type: String,
        regEx: SimpleSchema.RegEx.Email
```

```
    }
  }).validate({email});
  console.log('this is the user ', user);
  return true;
});

try {
  throw new Error('Some message here');
} catch (e){
  console.log(e);
}
```

- and the some message here will appear in the terminal

```
I20171016-16:06:35.410(-7)? [Error: Some message here]
=> Meteor server restarted
```

- and now we are going to throw a Meteor error
- the Meteor.Error takes two arguments, the first is one is some code like 404 or even a string but for now we will stick with 400, and the second argument is your message, the reason it was invalid

```
Accounts.validateNewUser((user)=>{
  const email = user.emails[0].address

  new SimpleSchema({
    email:{
      type: String,
      regEx: SimpleSchema.RegEx.Email
    }
  }).validate({email});
  console.log('this is the user ', user);
  return true;
});

try {
  throw new Meteor.Error(400, 'Please enter a valid email.');
} catch (e){
```

```
    console.log(e);
  }
```

- now if we go back to the terminal this is what we get

```
I20171016-16:11:41.507(-7)? { [Error: Please enter a valid email. [400]]
I20171016-16:11:41.508(-7)?   isClientSafe: true,
I20171016-16:11:41.508(-7)?   error: 400,
I20171016-16:11:41.508(-7)?   reason: 'Please enter a valid email.',
I20171016-16:11:41.508(-7)?   details: undefined,
I20171016-16:11:41.509(-7)?   message: 'Please enter a valid email. [400]',
I20171016-16:11:41.509(-7)?   errorType: 'Meteor.Error' }
=> Meteor server restarted
```

- so in order to use this try and catch clause, all we have to do is wrap our try and catch clause on our new SimpleSchema like so

```
import { Meteor } from 'meteor/meteor';
import SimpleSchema from 'simpl-schema';
import { Accounts } from 'meteor/accounts-base';


Meteor.startup(() => {
  // code to run on server at startup


  Accounts.validateNewUser((user)=>{
    const email = user.emails[0].address


    try{
      new SimpleSchema({
        email:{
          type: String,
          regEx: SimpleSchema.RegEx.Email
        }
      }).validate({ email });
    }catch (e) {
      throw new Meteor.Error(400, e.message )
    }
```

```
    return true;
  });
```

- now lets try create a new user in the browser and use a bogus email and password
- and now we get this message : ***Email must be a valid email address***
- now we are going to take care of the password
- inside our Signup.js component we are going to ba adding validation in onSubmit()

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


      let email = this.refs.email.value.trim();
      let password = this.refs.password.value.trim();
      if(password.length < 9){
          return this.setState({error: 'Password must be more than 8 characters long.'})
      }
     Meteor.loginWithPassword({email}, password, (err)=>{
        if(err){
          this.setState({error: 'Unable to login.  Check email and password'});
        }else{
          this.setState({error: ''});
        }
     })
  }
```

- lets go to the browser and check if this works by typing a short password
- and we get this message ***Password must be more than 8 characters long.***