07-Short-Link-App

Showing Meteor Error Messages- Video

- first lets explore the error message if we try to login without email and password

```
errorClass {isClientSafe: true, error: 400, reason: "Match failed", details: undefined, message:
 "Match failed [400]", …}
```

- so we are going to be implementing the reason why it didn't work to login
- so lets start with an if statement
- so go to Signup.js

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

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

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

           }else{

           }
        });

    }
```

- if it does give an error we have to update the state so use setState

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

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

```
        Accounts.createUser({email, password}, (err)=>{
           if(err){
            this.setState({error: err.reason});
           }else{

           }
        });

    }
```

- if there isn't one we are going to clear the state

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

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

        Accounts.createUser({email, password}, (err)=>{
           if(err){
            this.setState({error: err.reason});
           }else{
            this.setState({error: ''});
           }
        });

    }
```

- now if you go to /signup and try to create account without a email or password you should get an error message above
- and if I just put in a password I get a different error above, so everything is working
- now let's do the samething with Login.js but instead of Accounts.createUser, we are going to be using Meteor.loginWithPassword like so...

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

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

      Meteor.loginWithPassword({email}, password, (err)=>{
        if(err){
          this.setState({error: err.reason});
        }else{
          this.setState({error: ''});
        }
      })
    }
```

- lets replace our default error message to something more meaninful

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

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

      Meteor.loginWithPassword({email}, password, (err)=>{
        if(err){
          this.setState({error: 'Unable to login.  Check email and password'});
        }else{
          this.setState({error: ''});
        }
      })
    }
```

Schema Validation - Video

- we want to make sure the user types in a valid email and a valid password, we are going to be using a library called simple schema
- so lets install it over in our terminal, open up a new tab and type
- make sure to type it without the e in simple, because that is different library

```
meteor npm install simpl-schema@0.0.3 --save
```

- now check the package.json in your folder and make sure is there

```
{
  "name": "short-lnk",
  "private": true,
  "scripts": {
    "start": "meteor run"
  },
  "dependencies": {
    "babel-runtime": "6.18.0",
    "meteor-node-stubs": "~0.2.0",
    "react": "^15.4.1",
    "react-dom": "^15.4.1",
    "react-router": "^3.0.0",
    "simpl-schema": "0.0.3"
  }
}
```

- now lets go and import our default export and call it SimpleSchema, and this is a constructor function, so we are going to be calling with the new keyword
- so go over to server main.js

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

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

- lets now practice with this
- now to create a schema all we have to do is create a variable
- it take one argument that is an object- we provide all the properties we want for it, and we provide the rules for those properties
- and of the most popular rule is what type, is it a string? a number? a boolean?

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

Meteor.startup(() => {
```

```
  // code to run on server at startup

const petSchema = new SimpleSchema({
  name: {

  }
});

});
```

- so for this we are going to be using String

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

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

const petSchema = new SimpleSchema({
  name: {
    type: String
  }
});

});
```

- now lets validate to see if its working, with Shema we can use the validate() method

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

  const petSchema = new SimpleSchema({
    name: {
      type: String
    }
```

```
  });
  petSchema.validate({
    name: 'Jose'
  });

});
```

- and as you can see it also take one argument, and it is an object, so in this case we Jose as the string.
- over in the terminal you won't see any changes, you will get an error message if the type is incorrect
- we can also use min and max for a size of a string

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

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

  const petSchema = new SimpleSchema({
    name: {
      type: String,
      min: 1,
      max: 200
    }

  });
  petSchema.validate({

  });

});
```

- we can also provide other properties like age

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

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

  const petSchema = new SimpleSchema({
    name: {
      type: String,
      min: 1,
      max: 200
    },
    age:{
      type: Number
    }

  });
  petSchema.validate({
    name: 'spot'
  });

});
```

- and we can also give it min or max properties

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

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

  const petSchema = new SimpleSchema({
    name: {
      type: String,
      min: 1,
      max: 200
    },
    age:{
```

```
      type: Number,
      min: 0
    }

  });
  petSchema.validate({
    name: 'spot',
    age: -3
  });

});
```

- so if we give this a -3 like you above, it will give an error, the min has to be at least 0
- we can also optional properties, and it won't give you an error

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

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

  const petSchema = new SimpleSchema({
    name: {
      type: String,
      min: 1,
      max: 200,
      optional: true
    },
    age:{
      type: Number,
      min: 0
    }

  });
  petSchema.validate({
    name: 'spot',
```

```
    age: -3
  });

});
```

- now you can delete the name under petSchema.validate and you won't get any errors
- remember that everything defined is required unless you say other wise
- to learn more about this check on google node simpl schema
- we are going to be using regEx, which is good for validating emails, urls, phone numbers which pretty specific patterns
- so lets start by creating a new property called contactNumber and it is going to be an object
- and this will include the regEx

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

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

  const petSchema = new SimpleSchema({
    name: {
      type: String,
      min: 1,
      max: 200,
      optional: true
    },
    age:{
      type: Number,
      min: 0
    },
    contactNumber:{
      type: String,
      optional: true,
      regEx: SimpleSchema.RegEx.Phone
    }

  });
  petSchema.validate({
```

```
    name: 'spot',
    age: 21
  });

});
```

- now lets validate that number

```
  });
  petSchema.validate({
    name: 'spot',
    age: 21,
    contactNumber: '1234'
  });

});
```

- and the above will work, but if you the following it won't work

```
  });
  petSchema.validate({
    name: 'spot',
    age: 21,
    contactNumber: '12#$34'
  });

});
```

- and this is the error you will get in the terminal

```
ClientError: Contact number failed regular expression validation
```

- as a challenge we did an employeeSchema

```
  const employeeSchema = new SimpleSchema({
    name: {
      type: String,
      min: 1,
```

```
      max: 200
    },
    hourlyWage:{
      type: Number,
      min: 0
    },
    email:{
      type: String,
      regEx: SimpleSchema.RegEx.Email,
      optional: true
    }
  });

  employeeSchema.validate({
    name:'Charles',
    hourlyWage: 25,
    email: 'charles@email.com'
  });

});
```

-