Sign-Up Form Validation

Let's validate the information from all the fields in our sign-up form.

WE'LL COVER THE FOLLOWING

In the last lesson, we were almost done with the sign-up form of our application.

One piece in the form is missing: validation. Let's use an `isInvalid` boolean to enable or disable the submit button.

```
...

class SignUpForm extends Component {

  ...

  render() {
    const {
      username,
      email,
      passwordOne,
      passwordTwo,
      error,
    } = this.state;

    const isInvalid =
      passwordOne !== passwordTwo ||
      passwordOne === '' ||
      email === '' ||
      username === '';

    return (
      <form onSubmit={this.onSubmit}>
        <input
        ...
        <button disabled={isInvalid} type="submit">
          Sign Up
        </button>
        {error && <p>{error.message}</p>}
      </form>
    );

  }
}

...
```

SignUp/index.js

Sign-Up Criteria

The user is only allowed to sign up if :

- both passwords are the same;
- the username, email and at least one password are filled with a string.

This is password confirmation in a common signup process.

We can visit the `/signup` route in our browser after starting our application to confirm that the form shows up with all its input fields. We should be able to type into it (confirming that the local state updates are working) and to enable the **submit** button by providing all input fields with a string (confirmation that the validation works).

What's missing in the component is the `onSubmit()` class method, which will pass all the form data to the Firebase authentication API via the authentication interface in the Firebase class:

```
...

class SignUpForm extends Component {

  ...

  onSubmit = event => {
    const { username, email, passwordOne } = this.state;

    this.props.firebase
      .doCreateUserWithEmailAndPassword(email, passwordOne)
      .then(authUser => {
        this.setState({ ...INITIAL_STATE });
      })
      .catch(error => {
        this.setState({ error });
      });

    event.preventDefault();
  };
  ...
}


...
```

SignUp/index.js

The code is not working yet, but let's break down what we have so far. All the necessary information passed to the authentication API can be restructured from the local state. You will only need one password property because both password strings should be the same after the validation.

Next, call the sign-up function defined in the previous section in the Firebase class, which takes the email and the password property. The username is not used for the sign-up process but will be used later.

If the request resolves successfully, we can set the local state of the component to its initial state and empty the input fields. If the request is rejected, you will run into the catch block and set the error object in the local state. An error message should show up in the form due to the conditional rendering in your component's render method.

Also, the `preventDefault()` method on the event prevents a reload of the browser which would otherwise be natural when using submit in a form. Note that the signed up user object from the Firebase API is available in the callback function of the `then` block in our request. You will use it later with the username.

The Firebase Instance

We can see that one essential piece is missing: We haven't made the Firebase instance available in the SignUpForm component's props yet. Let's change this by utilizing our Firebase Context in the `SignUpPage` component and passing the Firebase instance to the `SignUpForm`.

```
import React, { Component } from 'react';
import { Link } from 'react-router-dom';

import { FirebaseContext } from '../Firebase';
import * as ROUTES from '../../constants/routes';

const SignUpPage = () => (
  <div>
```

```
    <h1>SignUp</h1>
    <FirebaseContext.Consumer>
      {firebase => <SignUpForm firebase={firebase} />}

    </FirebaseContext.Consumer>
  </div>
);

const INITIAL_STATE = { ... };

class SignUpForm extends Component {
  ...
}

...
```

SignUp/index.js

Now the registration of a new user should work.

However, there can be another improvement on how we access the Firebase instance here. We'll find out in the next lesson.