Sign Up with React and Firebase

Let's implement the sign-up functionality using Firebase.

WE'LL COVER THE FOLLOWING

We've set up all the routes for our application, configured Firebase and implemented the authentication API for our Firebase class. We've also made Firebase available for use within our React components.

Now it's time to use the authentication functionalities in our React components, which we'll build from scratch. Most of the code is in one block because the components are not too small and splitting them up step-by-step might be too verbose. Nevertheless, we will examine each code block afterward. The code blocks for forms can become repetitive, so they will be explained once well.

Creating the Sign-Up Page

Let's start with the sign-up page (registration page). It consists of the *page*, a *form*, and a *link*.

The form is used to sign up a new user to our application through a *username*, *email*, and *password*. The link will be used on the sign-in page (login page) later if a user has no account yet. It is a redirect to the sign-up page, but not used on the sign-up page itself.

Implement the `src/components/SignUp/index.js` file the following way:

```
import React, { Component } from 'react';
```

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

import * as ROUTES from '../../constants/routes';


const SignUpPage = () => (
  <div>
    <h1>SignUp</h1>
    <SignUpForm />
  </div>
);

class SignUpForm extends Component {
  constructor(props) {
    super(props);
  }

  onSubmit = event => {

  }

  onChange = event => {

  };

  render() {
    return (
      <form onSubmit={this.onSubmit}>

      </form>
    );
  }
}

const SignUpLink = () => (
  <p>
    Don't have an account? <Link to={ROUTES.SIGN_UP}>Sign Up</Link>
  </p>
);

export default SignUpPage;

export { SignUpForm, SignUpLink };
```

SignUp/index.js

The `SignUpForm` component is the only React class component in this file because it has to manage the form state in React's local state. There are two pieces missing in the current `SignUpForm` component:

- The form content in the render method in terms of input fields to capture the information (email address, password, etc.) of a user.
- The implementation of the `onSubmit` class method when a user signs up eventually.

The Initial State

The Initial State

First, let's initialize the state of the component. It will capture user information such as username, email, and password. There will be a second password field/state for password confirmation.

In addition, there is an error state to capture an error object in case the sign-up request to the Firebase API fails. The state is initialized by an **object destructuring**. This way, we can use the initial state object to reset the state after a successful sign-up.

```
...

const INITIAL_STATE = {
  username: '',
  email: '',
  passwordOne: '',
  passwordTwo: '',
  error: null,
};

class SignUpForm extends Component {
  constructor(props) {
    super(props);

    this.state = { ...INITIAL_STATE };
  }

  ...

}

...
```

SignUp/index.js

Form Fields

Let's implement all the input fields to capture the information in the `render` method of the component. The input fields need to update the local state of the component by using an `onChange` handler.

```
..

class SignUpForm extends Component {

  ...

  onChange = event => {
    this.setState({ [event.target.name]: event.target.value });
```

```
  };

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

    return (
      <form onSubmit={this.onSubmit}>
        <input
          name="username"
          value={username}
          onChange={this.onChange}
          type="text"
          placeholder="Full Name"
        />
        <input
          name="email"
          value={email}
          onChange={this.onChange}
          type="text"
          placeholder="Email Address"
        />
        <input
          name="passwordOne"
          value={passwordOne}
          onChange={this.onChange}
          type="password"
          placeholder="Password"
        />
        <input
          name="passwordTwo"
          value={passwordTwo}
          onChange={this.onChange}
          type="password"
          placeholder="Confirm Password"
        />
        <button type="submit">Sign Up</button>

        {error && <p>{error.message}</p>}
      </form>
    );
  }
}

...
```

SignUp/index.js

Let’s take the last implemented code block apart. All the input fields implement the unidirectional data flow of React; thus, each input field gets a value from the local state and updates the value in the local state with an `onChange` handler.

The input fields are controlled by the local state of the component and don't control their own states. They are **controlled components**.

In the last part of the form, there is an optional error message from an error object. The error objects from Firebase have this message property by default, so we can rely on it to display the proper text to the user. However, the message is only shown when there is an actual error using a **conditional rendering**.

In the next lesson, we'll implement form validation.