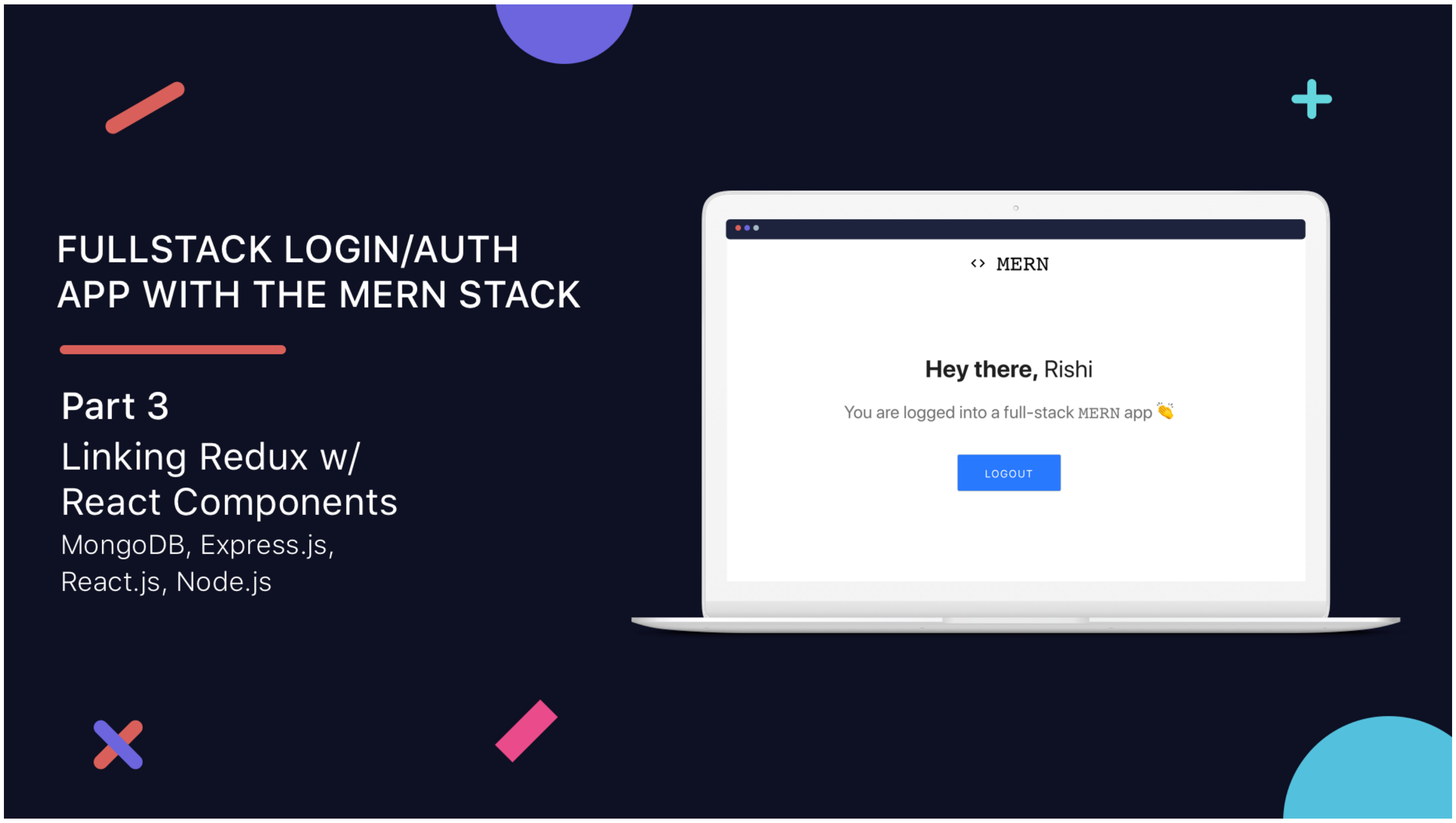
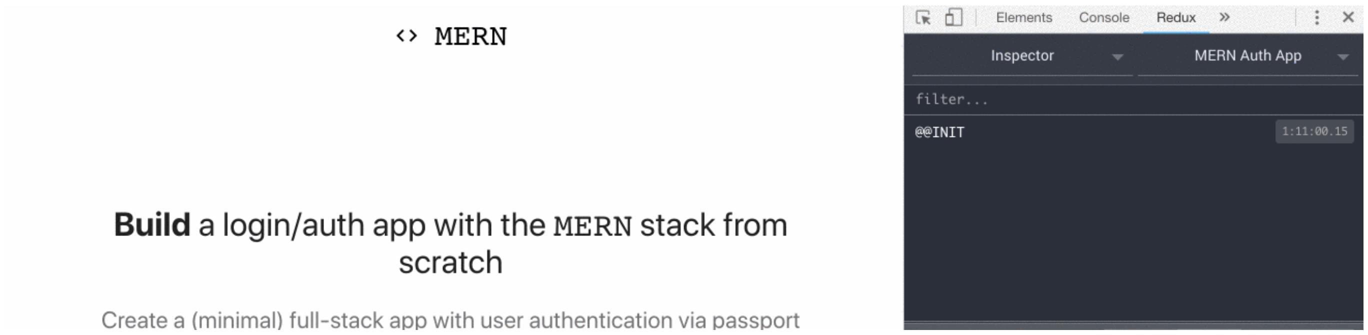
Build a Login/Auth App with the MERN Stack — Part 3 (Linking Redux with React Components)

Create a (minimal) full-stack app with user authentication via passport and JWTs

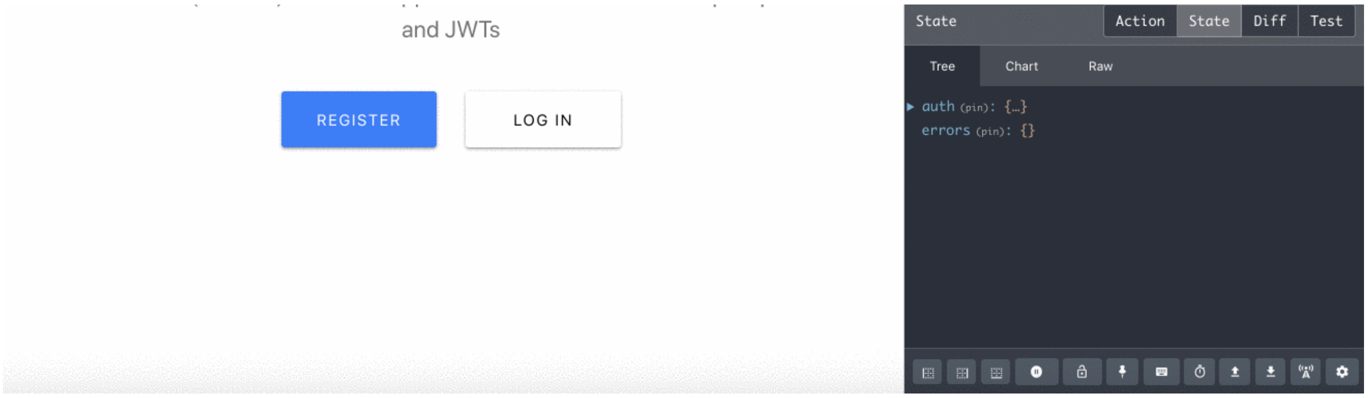
Rishi Prasad Follow
Dec 5, 2018 · 9 min read


FULLSTACK LOGIN/AUTH APP WITH THE MERN STACK
Part 3
Linking Redux w/ React Components
MongoDB, Express.js, React.js, Node.js
<> MERN
Hey there, Rishi
You are logged into a full-stack MERN app
LOGOUT


<> MERN

Build a login/auth app with the MERN stack from scratch

Create a (minimal) full-stack app with user authentication via passport



The finished product: a full-stack MERN app with Redux for state management (repo)

Before we get started

Read Part 1: Creating our Backend

Build a Login/Auth App with the MERN Stack — Part 1 (Backend)
Create a (minimal) full-stack app with user authentication via passport and JWTs.
blog.bitsrc.io

In Part 1, we

- Initialized our backend using `npm` and installed necessary packages
- Set up a MongoDB database using mLab
- Set up a server with `Node.js` and `Express`
- Created a database schema to define a `User` for registration and login purposes
- Set up three API routes, `register` , `login` , and `currentuser` using `passport` + `jsonwebtoken` s for authentication and `validator` for input validation
- Tested our API routes using Postman

Read Part 2: Frontend & Redux Setup

Build a Login/Auth App with the MERN Stack — Part 2 (Frontend &

Create a (minimal) full-stack app with user authentication via passport and JWTs

blog.bitsrc.io

In Part 2, we

- Set up our frontend using `create-react-app`
- Created static `components` for our Navbar, Landing, Login and Register pages
- Setup `Redux` for global state management

In this final part, we will

- Link `Redux` to our components
- Display errors from our backend in our `React` forms
- Create protected routes (pages only certain users can access based on their authentication)
- Keep a user logged when they refresh or leave the page (in other words, until they either logout or the `jwt` expires)

Part 3: Linking Redux with React Components

1. Linking `Redux` to our `Register` component and displaying errors in our form

i. Using `connect()` from `react-redux`

`connect()` does just that; it connects our `React` components to our `Redux` `store` provided by the `Provider` component

We have to modify our `export default Register;` at the bottom of `Register.js` . Read the `connect` documentation for more clarification.

```
export default connect(
  mapStateToProps,
  { registerUser }
)(withRouter(Register));
```

You may also notice we wrapped our `Register` with a `withRouter()`. While it is easy to redirect within a `component` (can simply say `this.props.history.push('/dashboard')` for example), we can't do that by default within an `action`. To allow us to redirect within an `action`, we

- Used `withRouter` from `react-router-dom`, wrapping our component in our `export withRouter()`
- Will add a parameter to `this.props.history` within our call to `this.props.registerUser(newUser, this.props.history)` in our `onSubmit` event so we can easily access it within our action (step iv below)

ii. `mapStateToProps`

`mapStateToProps` allows us to get our `state` from `Redux` and map it to `props` which we can use inside components.

We'll add the following above our `export` at the bottom of `Register.js`.

```
const mapStateToProps = state => ({
  auth: state.auth,
  errors: state.errors
});
```

This allows us to call `this.props.auth` or `this.props.errors` within our `Register` component.

iii. Defining `propTypes`

Since we cannot define types in our constructor, it is considered good convention to do so using the `prop-types` package.

```
Register.propTypes = {
  registerUser: PropTypes.func.isRequired,
  auth: PropTypes.object.isRequired,
  errors: PropTypes.object.isRequired
};
```

iv. Tying it all together

All said and done, let’s make the following **bolded** additions to our `Register.js` React component. We’ll also display errors within our form here.

```
import React, { Component } from "react";
import { Link, withRouter } from "react-router-dom";
import PropTypes from "prop-types";
import { connect } from "react-redux";
import { registerUser } from "../../actions/authActions";
import classnames from "classnames";

class Register extends Component {
  constructor() {
    super();
    this.state = {
      name: "",
      email: "",
      password: "",
      password2: "",
      errors: {}
    };
  }

componentWillReceiveProps(nextProps) {
    if (nextProps.errors) {
      this.setState({
        errors: nextProps.errors
      });
    }
  }

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

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

const newUser = {
      name: this.state.name,
      email: this.state.email,
      password: this.state.password,
      password2: this.state.password2
    };

this.props.registerUser(newUser, this.props.history);
  };

render() {
    const { errors } = this.state;

return (
      <div className="container">
        <div className="row">
          <div className="col s8 offset-s2">
            <Link to="/" className="btn-flat waves-effect">
              <i className="material-icons
left">keyboard_backspace</i> Back to
              home
            </Link>
            <div className="col s12" style={{ paddingLeft: "11.250px"
}}>
              <h4>
                <b>Register</b> below
              </h4>
              <p className="grey-text text-darken-1">
                Already have an account? <Link to="/login">Log
in</Link>
              </p>
            </div>
            <form noValidate onSubmit={this.onSubmit}>
              <div className="input-field col s12">
                <input
                  onChange={this.onChange}
                  value={this.state.name}
                  error={errors.name}
                  id="name"
                  type="text"
                  className={classnames("", {
                    invalid: errors.name
                  })}
                />
                <label htmlFor="name">Name</label>
                <span className="red-text">{errors.name}</span>
              </div>
              <div className="input-field col s12">
                <input
                  onChange={this.onChange}
                  value={this.state.email}
                  error={errors.email}
                  id="email"
                  type="email"
                  className={classnames("", {
                    invalid: errors.email
                  })}
                />
                <label htmlFor="email">Email</label>
                <span className="red-text">{errors.email}</span>
              </div>
              <div className="input-field col s12">
                <input
                  onChange={this.onChange}
                  value={this.state.password}
                  error={errors.password}
                  id="password"
                  type="password"
                  className={classnames("", {
                    invalid: errors.password
                  })}
                />
                <label htmlFor="password">Password</label>
                <span className="red-text">{errors.password}</span>
              </div>
              <div className="input-field col s12">
                <input
                  onChange={this.onChange}
                  value={this.state.password2}
                  error={errors.password2}
                  id="password2"
                  type="password"
                  className={classnames("", {
                    invalid: errors.password2
                  })}
                />
                <label htmlFor="password2">Confirm Password</label>
                <span className="red-text">{errors.password2}</span>
              </div>
              <div className="col s12" style={{ paddingLeft:
"11.250px" }}>
                <button
                  style={{
                    width: "150px",
                    borderRadius: "3px",
                    letterSpacing: "1.5px",
                    marginTop: "1rem"
                  }}
                  type="submit"
                  className="btn btn-large waves-effect waves-light
hoverable blue accent-3"
                >
                  Sign up
                </button>
              </div>
            </form>
          </div>
        </div>
      </div>
    );
  }
}

Register.propTypes = {
  registerUser: PropTypes.func.isRequired,
  auth: PropTypes.object.isRequired,
  errors: PropTypes.object.isRequired
};

const mapStateToProps = state => ({
  auth: state.auth,
  errors: state.errors
});

export default connect(
  mapStateToProps,
  { registerUser }
)(withRouter(Register));
```

ii. Linking `Redux` to our `Login` component and displaying errors in our form

Let's make the following **bolded** additions to our `Login.js` React component.

```
import React, { Component } from "react";
import { Link } from "react-router-dom";
import PropTypes from "prop-types";
import { connect } from "react-redux";
import { loginUser } from "../../actions/authActions";
import classnames from "classnames";

class Login extends Component {
  constructor() {
    super();
    this.state = {
      email: "",
      password: "",
      errors: {}
    };
  }

componentWillReceiveProps(nextProps) {
    if (nextProps.auth.isAuthenticated) {
      this.props.history.push("/dashboard"); // push user to
dashboard when they login
    }

if (nextProps.errors) {
      this.setState({
        errors: nextProps.errors
      });
    }
  }

onChange = e => {
    this.setState({ [e.target.id]: e.target.value });
  };

onSubmit = e => {
    e.preventDefault();

const userData = {
      email: this.state.email,
      password: this.state.password
    };

this.props.loginUser(userData); // since we handle the redirect
within our component, we don't need to pass in this.props.history as
a parameter
  };

render() {
    const { errors } = this.state;

return (
      <div className="container">
        <div style={{ marginTop: "4rem" }} className="row">
          <div className="col s8 offset-s2">
            <Link to="/" className="btn-flat waves-effect">
              <i className="material-icons
left">keyboard_backspace</i> Back to
              home
            </Link>
            <div className="col s12" style={{ paddingLeft: "11.250px"
}}>
              <h4>
                <b>Login</b> below
              </h4>
              <p className="grey-text text-darken-1">
                Don't have an account? <Link
to="/register">Register</Link>
              </p>
            </div>
            <form noValidate onSubmit={this.onSubmit}>
              <div className="input-field col s12">
                <input
```

```
                  onChange={this.onChange}
                  value={this.state.email}
                  error={errors.email}
                  id="email"
                  type="email"
                  className={classnames("", {
                    invalid: errors.email || errors.emailnotfound
                  })}
                />
                <label htmlFor="email">Email</label>
                <span className="red-text">
                  {errors.email}
                  {errors.emailnotfound}
                </span>
              </div>
              <div className="input-field col s12">
                <input
                  onChange={this.onChange}
                  value={this.state.password}
                  error={errors.password}
                  id="password"
                  type="password"
                  className={classnames("", {
                    invalid: errors.password ||
errors.passwordincorrect
                  })}
                />
                <label htmlFor="password">Password</label>
                <span className="red-text">
                  {errors.password}
                  {errors.passwordincorrect}
                </span>
              </div>
              <div className="col s12" style={{ paddingLeft:
"11.250px" }}>
                <button
                  style={{
                    width: "150px",
                    borderRadius: "3px",
                    letterSpacing: "1.5px",
                    marginTop: "1rem"
                  }}
                  type="submit"
                  className="btn btn-large waves-effect waves-light
hoverable blue accent-3"
                >
                  Login
                </button>
              </div>
            </form>
          </div>
        </div>
      </div>
    );
  }
}

Login.propTypes = {
  loginUser: PropTypes.func.isRequired,
  auth: PropTypes.object.isRequired,
  errors: PropTypes.object.isRequired
};

const mapStateToProps = state => ({
  auth: state.auth,
  errors: state.errors
});

export default connect(
  mapStateToProps,
  { loginUser }
)(Login);
```

Right now, when the user logs in, the app redirects us back to a blank page `"/dashboard"` per the first conditional statement of our **`componentWillReceiveProps(nextProps)`** lifecycle method. Next, we'll create our `Dashboard` component and make it a `PrivateRoute` so that only a logged in user can view it.

Creating our Dashboard component for when users log in

In our `component` directory, let's create a `dashboard` directory and within it, a `Dashboard.js` file.

```
➜  components mkdir dashboard && cd dashboard && touch Dashboard.js
```

Let's place the following in our `Dashboard.js` file.

```
import React, { Component } from "react";
import PropTypes from "prop-types";
import { connect } from "react-redux";
import { logoutUser } from "../../actions/authActions";

class Dashboard extends Component {
  onLogoutClick = e => {
    e.preventDefault();
  this.props.logoutUser();
 };

render() {
  const { user } = this.props.auth;

return (
   <div style={{ height: "75vh" }} className="container valign-
wrapper">
    <div className="row">
     <div className="col s12 center-align">
      <h4>
       <b>Hey there,</b> {user.name.split(" ")[0]}
       <p className="flow-text grey-text text-darken-1">
        You are logged into a full-stack{" "}
        <span style={{ fontFamily: "monospace" }}>MERN</span>
app 👏
       </p>
      </h4>
      <button
       style={{
        width: "150px",
        borderRadius: "3px",
        letterSpacing: "1.5px",
        marginTop: "1rem"
       }}
       onClick={this.onLogoutClick}
       className="btn btn-large waves-effect waves-light
hoverable blue accent-3"
      >
       Logout
      </button>
     </div>
    </div>
   </div>
  );
 }
}

Dashboard.propTypes = {
 logoutUser: PropTypes.func.isRequired,
 auth: PropTypes.object.isRequired
};

const mapStateToProps = state => ({
 auth: state.auth
});

export default connect(
 mapStateToProps,
 { logoutUser }
)(Dashboard);
```

Creating Protected Routes

There is no standard way of creating protected routes in `React`. We'll use the logic described in the below post by Tyler McGinnis to create authenticated routes (routes that only certain users can access based on their auth status).

Protected routes and authentication with React Router v4

Protected routes are an important part of any web application. In this post we'll break down the "Redirects (Auth)"...

tylermcginnis.com

In our `components` directory, let's create a directory and file for our private route.

```
➜  components mkdir private-route && cd private-route && touch
PrivateRoute.js
```

Let's place the following in `PrivateRoute.js`.

```
import React from "react";
import { Route, Redirect } from "react-router-dom";
import { connect } from "react-redux";
import PropTypes from "prop-types";

const PrivateRoute = ({ component: Component, auth, ...rest }) => (
  <Route
    {...rest}
    render={props =>
      auth.isAuthenticated === true ? (
        <Component {...props} />
      ) : (
        <Redirect to="/login" />
      )
    }
  />
);

PrivateRoute.propTypes = {
  auth: PropTypes.object.isRequired
};
```

```
const mapStateToProps = state => ({
  auth: state.auth
});

export default connect(mapStateToProps)(PrivateRoute);
```

Tying it all together in App.js

In this, we will

- Check `localStorage` for a token to keep the user logged in even if they close or refresh the app (e.g. until they log out or the token expires)
- Pull in our `Dashboard` component and define it as a `PrivateRoute`

Make the following bolded additions to `App.js`.

```
import React, { Component } from "react";
import { BrowserRouter as Router, Route, Switch } from "react-router-
dom";
import jwt_decode from "jwt-decode";
import setAuthToken from "./utils/setAuthToken";
import { setCurrentUser, logoutUser } from "./actions/authActions";

import { Provider } from "react-redux";
import store from "./store";

import Navbar from "./components/layout/Navbar";
import Landing from "./components/layout/Landing";
import Register from "./components/auth/Register";
import Login from "./components/auth/Login";
import PrivateRoute from "./components/private-route/PrivateRoute";
import Dashboard from "./components/dashboard/Dashboard";

// Check for token to keep user logged in
if (localStorage.jwtToken) {
  // Set auth token header auth
  const token = localStorage.jwtToken;
  setAuthToken(token);
  // Decode token and get user info and exp
  const decoded = jwt_decode(token);
  // Set user and isAuthenticated
  store.dispatch(setCurrentUser(decoded));

// Check for expired token
  const currentTime = Date.now() / 1000; // to get in milliseconds
```

```
    if (decoded.exp < currentTime) {
      // Logout user
      store.dispatch(logoutUser());

      // Redirect to login
      window.location.href = "./login";
    }
  }

class App extends Component {
  render() {
    return (
      <Provider store={store}>
        <Router>
          <div className="App">
            <Navbar />
            <Route exact path="/" component={Landing} />
            <Route exact path="/register" component={Register} />
            <Route exact path="/login" component={Login} />
            <Switch>
              <PrivateRoute exact path="/dashboard" component=
{Dashboard} />
            </Switch>
          </div>
        </Router>
      </Provider>
    );
  }
}

export default App;
```

One last step!

It wouldn't make sense for logged in users to be able to access the `/login` and `/register` pages. If a logged in user navigates to either of these, we should immediately redirect them to the dashboard.

To achieve this, add the following lifecycle method below the constructor in `Register.js`.

```
componentDidMount() {
    // If logged in and user navigates to Register page, should
redirect them to dashboard
    if (this.props.auth.isAuthenticated) {
      this.props.history.push("/dashboard");
      }
    }
```

And add the same lifecycle method below the constructor in `Login.js`.

```
  componentDidMount() {
      // If logged in and user navigates to Login page, should redirect
  them to dashboard
      if (this.props.auth.isAuthenticated) {
        this.props.history.push("/dashboard");
      }
    }
```

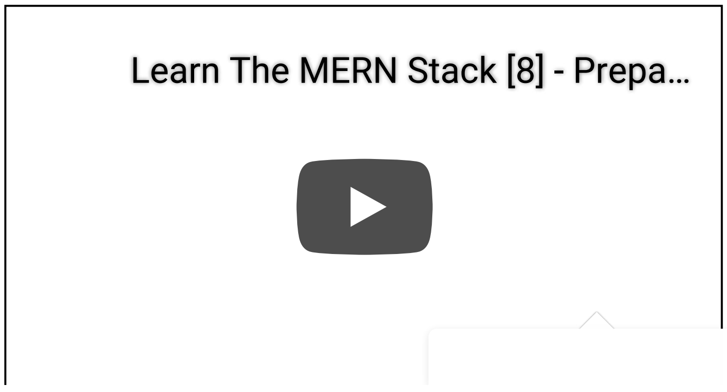
. . .

That's a wrap.

We now have an app that allows users to

- Register
- Log in
- Access protected pages only accessible to logged in users
- Stay logged in when they close the app or refresh the page
- Log out

If you want to deploy your app to `Heroku`, please refer to the following video.



Full codebase for the project can be viewed here:

rishipr/mern-auth
Minimal full-stack MERN app with authentication using passport and JWTs — rishipr/mern-auth
github.com

The purpose of this was to create a strong foundation to build off for a more functional `MERN` app and to get you comfortable working with the `MERN` stack. If you are looking for a convenient auth solution, you may want to explore Google's `Firebase`. I haven't used it yet, but have heard great things and have it on my "to explore" list.

Acknowledgements

Shoutout to Brad Traversy over at Traversy Media for his wonderful MERN course on Udemy (how I learned this material). He also provides fantastic educational videos and resources on his YouTube channel — would highly recommend!

Traversy Media
Traversy Media features the best online web development and programming tutorials for all of the latest web...
www.youtube.com

. . .

Learn more

5 Tools for Faster Development in React

5 tools to speed the development of your React application, focusing on components.

blog.bitsrc.io

11 React UI Component Libraries you Should Know in 2018

11 React component libraries with great components for building your next app's UI interface in 2018.

blog.bitsrc.io

How to Share React UI Components between Projects and Apps

How to easily share and sync your React UI components between all your team's projects and applications with Bit.

blog.bitsrc.io

React Nodejs Mongodb JavaScript Full Stack