React Router

`<Route>` Functions like an *if-statement*, conditionally renders the given component based on the URL path

`<Link>` Functions like an *a tag* link, "tricks" user that they are going to a new page while changing URL

`<Switch>` Let only 1 route get matched

Examples

Router and Redux need `src/index.js` modifications (imports omitted).

```
let store = createStore(reducer);
ReactDOM.render(

    <Provider store={store}>
      <BrowserRouter>
        <App />
      </BrowserRouter>
    </Provider>,

  document.getElementById("root"));
```

Put top-level routes in `App.js`:

```
<Switch>
  <Route path="/about/"
    component={About} />
  <Route path="/post/:id/"
    component={BlogPost} />
</Switch>
```

Link examples:

```
<Link to="/about/">About</Link>
<Link to={"/post/"+postId+"/"}>
  Read More...</Link>
```

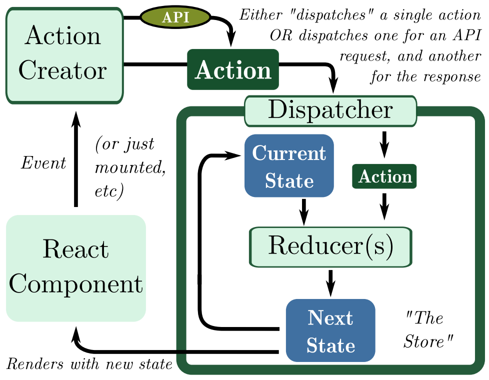
MERN Stack

MongoDB NoSQL database that stores JSON *documents*, with no built-in schema-enforcement

Express.js Most popular backend web framework for Node.js

React + Redux State-management library, popular with large React projects where state gets too huge for the App component

React Redux



Store The Redux "ORM", used most for data fetched from back-end

Action Represents an "event" that occurs, e.g. an action is *dispatched* when data is `fetch`ed from the back-end, and another when the response comes back

Reducer Triggered when an action is dispatched, a reducer modifies (a duplicate of) the state based on the action that happened

React Redux Code

Action Creators (found in `actions/`)

```
const doIncrement = () => {
  return {type: INCREMENT};}
const addTodo = (item) => {
  return {type: ADD, text: item};}
```

Dispatching (found in `components/`)

```
let action =
  addTodo(this.state.text);
this.props.dispatch(action);
```

Reducers (found in `reducers/`)

```
const initialState = {
  count: 0,
  todoList: [],
};
const todo = (state, action) => {
switch (action.type) {
  case INCREMENT:
  return Object.assign({},state,{
    count: state.count + 1,
  });
  case ADD:
  return Object.assign({},state,{
    todoList: todoList.concat([
    { text: action.text } ]),
  });
```

MongoDB

noSQL database A database that doesn't use SQL and traditional table / row / column organization

document *row* in SQL — an item of data, in BSON (a JSON variant)

collection *table* in SQL — group of documents (items of data)

ObjectID Long random string serving as unique ID for each document

MongoDB CRUD

```
db.userprofiles.find(
  {name: "janeqhacker"})
db.userprofiles.insertOne({
  name: "janeqhacker",
  mood: "happy",
  posts: [] })
db.userprofiles.update(
  { name: "janeqhacker" }, {
    $push: { posts: "Agreed!" },
    $set: { mood: "happy" } })
db.userprofiles.deleteOne(
  {name: "janeqhacker"})
```

Express.js + Mongo

```
const app = express();
app.get("/", (req, res) => {
  res.send("Hello World!"); });
app.post("/all", (req, res) => {
  db.collection("userprofiles")
    .find({})
    .toArray((err, data) => {
      if (err) throw err;
      res.json(data);
    });
});
app.get("/u/:name", (req,res)=>{
  const uName = req.params.name;
  db.collection("userprofiles")
    .find({username: uName})
    .toArray((err, data) => {
      if (err) throw err;
      res.json(data);
    });
});
app.post("/create", (req,res) => {
  const data = {name: "jqhacker"};
  db.collection("userprofiles")
    .insertOne(data, (err,data)=>{
    if (err) throw err;
    res.json({success: true});
});
app.listen(3000, () => {
  console.log("ready @ :3000"); });
```