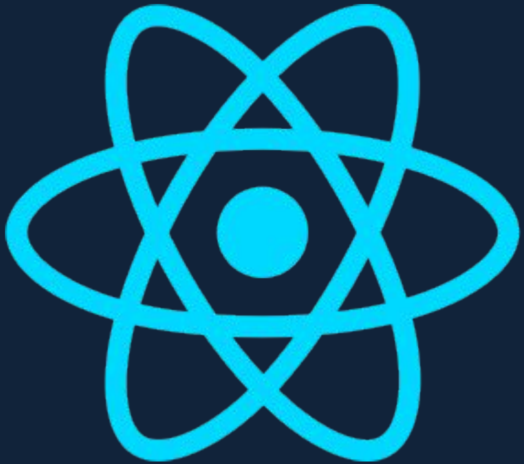
Workshop React Router



React Router

A SPA handles routing in most cases by itself.

A routing library helps with managing the routing logic/state.

Frameworks like Next.js or Blitz.js handle routing for us – it’s build into the framework.

Why / What you'll learn

- How to define Routes in a declarative way
- Using `react-router-dom` to build a more complex application

The router maps URLs to components / screens

`/books` → `<BookList />`

React Router

React Router is a routing library for React.

It’s “just” React – everything is a component and follows the principles we’ve learned so far.

The React Router Module

- **Declarative** routing for React
- Uses React elements to define routes
- Components of React Router in the `react-router-dom` module:

  -
    ```
    npm install react-router-dom@5
    npm install --save-dev @types/react-router-dom@5
    ```

  -
    ```
    import {Router} from 'react-router-dom'
    ```

`<Router />`

Primary component of React Router. It keeps your UI and the URL in sync.

Router Implementations

- `<BrowserRouter />` for HTML5-History Routing
- `<HashRouter />` for Hash-Routing (older browsers)
- `<MemoryRouter />` for ReactNative and Tests
- `<StaticRouter />` for ServerSideRendering

```
import {
  BrowserRouter as Router,
} from 'react-router-dom'
```

Router Element In Action

Just wrap your App inside of your Router-Component

```
import {BrowserRouter as Router} from 'react-router-dom';

ReactDOM.render(
    <Router>
        <App />
    </Router>,
  document.getElementById('root')
);
```

Router Element In Action

<code>

Just wrap your App inside of your Router-Component

```
import {BrowserRouter as Router} from 'react-router-dom';

ReactDOM.render(
    <Router>
        <App />
    </Router>,
  document.getElementById('root')
);
```

Needs to be at the top of the component tree, as it provides the routing context.

Route

A `<Route />` is used to declaratively map routes to your application's component hierarchy.

The Route Component

- Render some UI when a location matches the route's path
- Route Properties
  - path
  - exact
  - strict
  - component

```
import {
   Route,
} from 'react-router-dom'
```

The Route Component

With <Route> you’re able to insert components on path-match

```
<main>
  <Route exact path="/"><Home /></Route>
  <Route path="/about"><About /></Route>
</main>
```

Route Property Path

```
<Route path="/users/" component={User}/>
```

path	location.pathname	matches?
/users/	/users/1	yes
/users/	/users/max	yes
/users/	/users/max/profile/tip	yes

Route Property Strict

```
<Route strict path="/one/" component={About}/>
```

path	location.pathname	matches?
/one/	/one	no
/one/	/one/	yes
/one/	/one/two	yes

Route Property Exact

```
<Route exact path="/one" component={About}/>
```

path	location.pathname	exact	matches?
/one	/one/two	true	no
/one	/one/two	false	yes

Route Properties Strict + Exact

```
<Route exact strict path="/one" component={About}/>
```

path	location.pathname	matches?
/one	/one	yes
/one	/one/	no
/one	/one/two	no

Route Passing Props

Use “render” for more flexibility like passing props:

```
<Route path="/home" render={() => <Home myProp={someVar}/>}/>
```

Use children:

```
<Route path="/home"><Home myProp={someVar}/></Route>
```

Switch

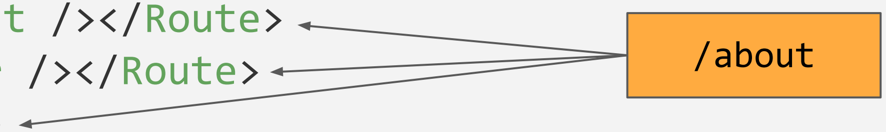
render only the first matching route (one at a time)

Normally, React Router would render all `<Route />`s which match the given path.

Without Switch

Without `<Switch />` every component of every matching route is rendered.

```
import { Route, Switch } from "react-router-dom";

return (
  <div>
    <Route exact path="/"><Home /></Route>
    <Route path="/about"><About /></Route>
    <Route path="/:user"><User /></Route>
    <Route><NoMatch /></Route>
  </div>
);
```



Using Switch

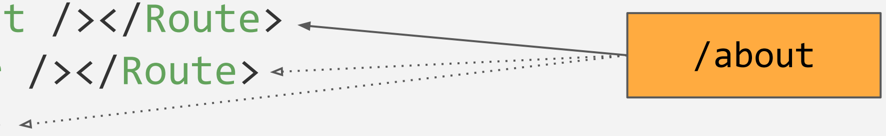
`<Switch />` can help us here

```
import { Route, Switch } from "react-router-dom";

return (
  <Switch>
    <Route exact path="/"><Home /></Route>
    <Route path="/about"><About /></Route>
    <Route path="/:user"><User /></Route>
    <Route><NoMatch /></Route>
  </Switch>
);
```

Using Switch

With `<Switch />` only route is rendered at a time

```
import { Route, Switch } from "react-router-dom";

return (
  <Switch>
    <Route exact path="/"><Home /></Route>
    <Route path="/about"><About /></Route>
    <Route path="/:user"><User /></Route>
    <Route><NoMatch /></Route>
  </Switch>
);
```



Link

The primary way to allow users to navigate around your application.

The Link Component

- `<Link />` will render a fully accessible anchor tag with the proper href.
  - Property: `to="/my/route"`
- `<NavLink />` adds properties to highlight the current route
  - `activeClassName`
  - `activeStyle`

```
import {
   Link, NavLink
} from 'react-router-dom'
```

Using Link

With <Link> you’re able to create links to routes

```
<Router>
  <div>
    <ul>
      <li><Link to="/home">Home</Link></li>
      <li><Link to="/about">About</Link></li>
    </ul>
    <Switch>
      <Route path="/home" ><Home /></Route>
      <Route path="/about" ><About /></Route>
      <Redirect to=”/home” />
    </Switch>
  </div>
</Router>
```

Task

Install and use React-Router v5

Route match and Params

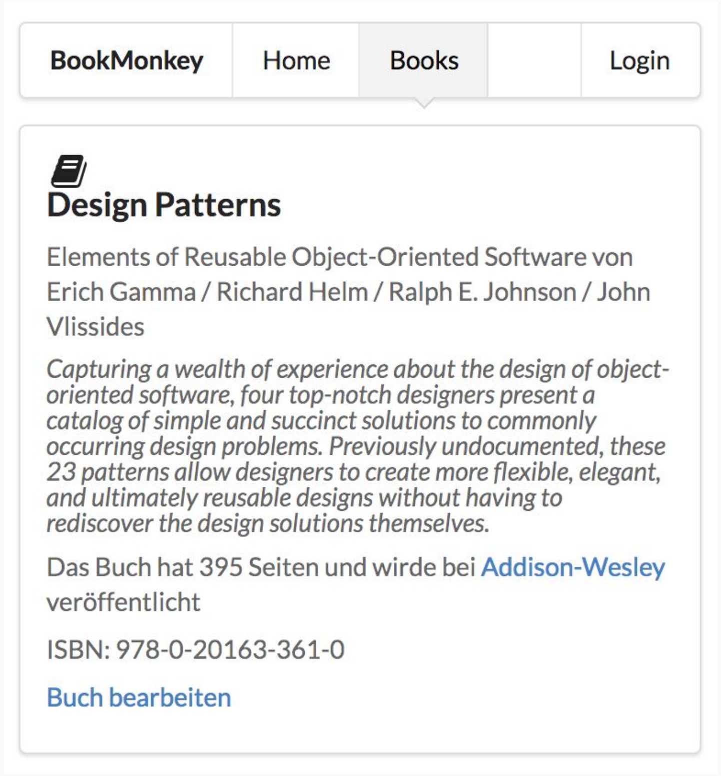
A match object contains information about how a <Route path> matched the URL.

Detail of a books

A detailed View of a book including the Abstract, Number of Pages, Publisher and ISBN.

Book details should be available under

`/books/:isbn`



Design Patterns

Elements of Reusable Object-Oriented Software von Erich Gamma / Richard Helm / Ralph E. Johnson / John Vlissides

Capturing a wealth of experience about the design of object-oriented software, four top-notch designers present a catalog of simple and succinct solutions to commonly occurring design problems. Previously undocumented, these 23 patterns allow designers to create more flexible, elegant, and ultimately reusable designs without having to rediscover the design solutions themselves.

Das Buch hat 395 Seiten und wirde bei Addison-Wesley veröffentlicht

ISBN: 978-0-20163-361-0

Buch bearbeiten

Read Params In A Component

<code>

Read the params via the `useParams` hook

```
import { useParams } from "react-router-dom";

const BookDetails: React.FC = () => {
  const { isbn } = useParams<{ isbn: string }>();
  // use the isbn to load your data
  return <p>ISBN: {isbn}</p>;
};
```

Type for expected params, following the URL (`/books/:isbn`).

match objects

- `<Route exact path="/books/:isbn" component={BookDetail}/>`
- Access the match object inside a component via `useRouteMatch()`
- Match properties
  - `params` - (object) Key/value pairs
  - `isExact` - (bool) true if the entire URL was matched (no trailing characters)
  - `path` - (string) The path pattern used to match. Useful for building nested <Route>s
    - e.g. `books/:isbn`
  - `url` - (string) The matched portion of the URL. Useful for building nested <Link>s
    - e.g. `books/572394732832`

Read match information in a component

<code>

Read the match information and params via the `useRouteMatch` hook

```
import { useRouteMatch } from "react-router-dom";

const BookDetails: React.FC = () => {
  const {
    params: { isbn },
    // other match properties
  } = useRouteMatch<{ isbn: string }>();
  // use the isbn to load your data
  return <p>ISBN: {isbn}</p>;
};
```

Other information like `path`, `url`, `isExact`.

Type for expected params, following the URL (`/books/:isbn`).

Task

Create a route for a basic BookDetails component

Task

Show data in BookDetails

Nested routes

Use `<Route>` inside a component that is mounted via `<Route>`

Nested routes can help with…

- **Code splitting**: routes which might never be hit are not in the main bundle
- **Code organization**: different teams can work on different parts of an application without interfering

Example: Nested route for editing a book

<code>

Inside our screen to display book details, we can declare sub-routes e.g. for editing a book.

```
const BookDetails: React.FC = () => {
  const { path, url, params } = useRouteMatch<{ isbn: string }>();
  const book = useBook(params.isbn);
  return (
    <>
      <Route exact path={path}>
        <Book book={book} />
        <Link to={`${url}/edit`}>Edit</Link>
      </Route>
      <Route exact path={`${path}/edit`}>
        <EditBook book={book} />
      </Route>
    </>
  );
};
```

Screen gets displayed when going to `/books/:isbn`.

Example: Nested route for editing a book

<code>

Inside our screen to display book details, we can declare sub-routes e.g. for editing a book.

```
const BookDetails: React.FC = () => {
  const { path, url, params } = useRouteMatch<{ isbn: string }>();
  const book = useBook(params.isbn);
  return (
    <>
      <Route exact path={path}>
        <Book book={book} />
        <Link to={`${url}/edit`}>Edit</Link>
      </Route>
      <Route exact path={`${path}/edit`}>
        <EditBook book={book} />
      </Route>
    </>
  );
};
```

Get match object with `params`, `path` and `url`.

Example: Nested route for editing a book

<code>

Inside our screen to display book details, we can declare sub-routes e.g. for editing a book.

```
const BookDetails: React.FC = () => {
  const { path, url, params } = useRouteMatch<{ isbn: string }>();
  const book = useBook(params.isbn);
  return (
    <>
      <Route exact path={path}>
        <Book book={book} />
        <Link to={`${url}/edit`}>Edit</Link>
      </Route>
      <Route exact path={`${path}/edit`}>
        <EditBook book={book} />
      </Route>
    </>
  );
};
```

Use path to declare nested routes (`path === "/books/:isbn"`).

Example: Nested route for editing a book

<code>

Inside our screen to display book details, we can declare sub-routes e.g. for editing a book.

```
const BookDetails: React.FC = () => {
  const { path, url, params } = useRouteMatch<{ isbn: string }>();
  const book = useBook(params.isbn);
  return (
    <>
      <Route exact path={path}>
        <Book book={book} />
        <Link to={`${url}/edit`}>Edit</Link>
      </Route>
      <Route exact path={`${path}/edit`}>
        <EditBook book={book} />
      </Route>
    </>
  );
};
```

Use url to declare correct link (`url === "/books/123"`).

Example: Nested route for editing a book

Inside our screen to display book details, we can declare sub-routes e.g. for editing a book.

```
const BookDetails: React.FC = () => {
  const { path, url, params } = useRouteMatch<{ isbn: string }>();
  const book = useBook(params.isbn);
  return (
    <>
      <Route exact path={path}>
        <Book book={book} />
        <Link to={`${url}/edit`}>Edit</Link>
      </Route>
      <Route exact path={`${path}/edit`}>
        <EditBook book={book} />
      </Route>
    </>
  );
};
```

Use `exact` to enforce an exact match, otherwise the order matters.

Example: Nested route for editing a book

<code>

Update the main route declaration in `App.tsx` to not be an exact match for `/books/:isbn` anymore.

```
<Switch>
  <Route exact path="/books">
    <BooksScreen />
  </Route>
  <Route ~~exact~~ path="/books/:isbn">
    <BookScreen />
  </Route>
</Switch>
```

Can’t be `exact` anymore, as nested routes wouldn’t match otherwise.

Example: Nested route for editing a book

<code>

Update the main route declaration in `App.tsx` to not be an exact match for `/books/:isbn` anymore.

```
<Switch>
  <Route exact path="/books">
    <BooksScreen />
  </Route>
  <Route path="/books/:isbn">
    <BookScreen />
  </Route>
</Switch>
```

Handles multiple routes now:

- `/books/:isbn`
- `/books/:isbn/edit`
- `/books/:isbn/… ?`

Task

Create a nested route to edit a book

⚠️

Be careful...

- Nested routes can help with code splitting and code organisation or let teams work independently from each other.
- You lose the overview over all routes in one place: Some routes might be declared somewhere deep in your application.