



The first Hub for Developers

Ztoupis Konstantinos

React Router

Code.Learn Program:
React

History

- a JavaScript library that lets you easily manage session history anywhere JavaScript runs
- provides a minimal API that lets you manage the history stack, navigate, confirm navigation, and persist state between sessions

Location

- an object that reflects “where” your application currently is
- contains a number of properties that are derived from a URL: pathname, search, and hash
- has a unique key property associated with it: can be used to identify and store data specific to a location
- can have state associated with it: provides a means of attaching data to a location that is not present in the URL

Location

```
{  
  pathname: '/home',  
  search: '?key=value',  
  hash: '#extra-info',  
  state: { isLoggedIn: true },  
  key: 'abc123'  
}
```

History object

- keeps track of an array of locations
- maintains an index value, which refers to the position of current location in the array

Navigation

Navigation methods allow you to change the current location:

- Push
- Replace
- goBack, goForward, go
- Listen

Push

- getting to a new location
- add a new location to the array after the current location
- clicking on a <Link> from react router, it will use history.push to navigate

```
history.push({ pathname: '/new-page' });
```

Replace

- is similar to push
- not adding a new location, replace the location at the current index
- it is used by react router's <Redirect> component

```
history.replace({ pathname: '/go-here' });
```


goBack, goForward, go

- goBack goes back one page. This essentially decrements the index in the locations array by one

```
history.goBack(); // -1
```

- goForward is the opposite of goBack, going forward one page. It will only work when there are “future” locations to go to, which happens when the user has clicked the back button

```
history.goForward(); // +1
```

goBack, goForward, go

- go is a more powerful combination of goBack and goForward. Negative numbers passed to the method will be used to go backwards in the array, and positive numbers will be used to go forward

```
history.go(-3); // -3
```

Listen

- method, which takes a function as its argument
- any time the location changes the history object will call all of its listener functions

```
history.listen(function(location) {  
  const text = location.pathname  
});
```

HTML5 History API

The HTML5 History API gives developers the ability to modify a website's URL without a full page refresh

```
> window.history
< ▼ History {state: null, length: 1, back: function, forward: function, go: function...} ⓘ
  length: 1
  state: null
  ▼ __proto__: History
    ► back: function back() { [native code] }
    ► constructor: function History() { [native code] }
    ► forward: function forward() { [native code] }
    ► go: function go() { [native code] }
    ► pushState: function () { [native code] }
    ► replaceState: function () { [native code] }
    ► __proto__: Object
```

The DOM window object provides access to the browser's history through the history object

Traveling through history

Moving backward:

```
window.history.back();
```

Moving forward:

```
window.history.forward();
```

Moving to a specific point in history: `window.history.go(-1);`

Adding - modifying history entries

`pushState()` method

- state object
- title
- URL

`replaceState()` method

- state object
- title
- URL

`popstate` event:
dispatched to the window every
time the active history entry
changes

`history.state`: reading the
current state

Routing

- the process of keeping the browser URL in sync with what's being rendered on the page
- two types of routing:
 - Static Routing
 - Dynamic routing

Static routing

routes are declared as part of your app's initialization before any rendering takes place

```
app.get('/', handleIndex)
app.get('/invoices', handleInvoices)
app.get('/invoices/:id', handleInvoice)
app.get('/invoices/:id/edit', handleInvoiceEdit)

app.listen()
```



```
Router.map(function() {
  this.route('about');
  this.route('contact');
  this.route('rentals', function() {
    this.route('show', { path: '/:rental_id' });
  });
});
```

export default Router



```
const appRoutes: Routes = [
  { path: 'crisis-center',
    component: CrisisListComponent
  },
  { path: 'hero/:id',
    component: HeroDetailComponent
  },
  { path: 'heroes',
    component: HeroListComponent,
    data: { title: 'Heroes List' }
  },
  { path: '',
    redirectTo: '/heroes',
    pathMatch: 'full'
  },
  { path: '**',
    component: PageNotFoundComponent
  }
];

@NgModule({
  imports: [
    RouterModule.forRoot(appRoutes)
  ]
})

export class AppModule { }
```



Dynamic routing

- routing that takes place **as your app is rendering**
- not in a configuration or convention outside of a running app

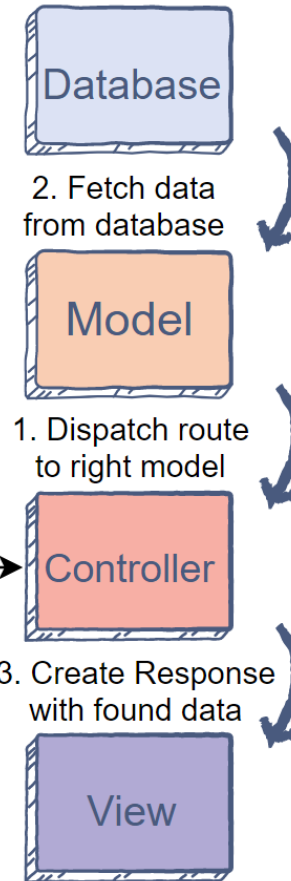
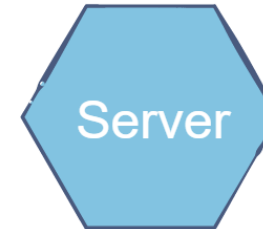
Static vs Dynamic

Old Architecture

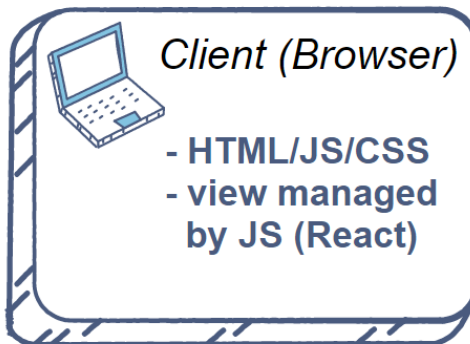
Future requests send/receive all views/assets



GET http://wikipedia.org



Process
Request



GET http://educative.io

Future requests are just for data

Modern Architecture

Client vs Server Side

- client side is the browser. Its processing happens on the local machine - like rendering a user interface in React.
- server side is where the information is processed and then sent through to a browser. Server-side means that the action takes place on a web server.

Why use React router?

- allows to build single page web applications (SPA) with navigation
- uses component structure to call components, which display the appropriate information
- allows the user to utilize browser functionality like the back button, and the refresh page, all while maintaining the correct view of the application

What is React Router?

- is a library that allows you to handle routes in a web app, using dynamic routing
- implements a component-based approach to routing
- provides different routing components according to the needs of the application and platform

React router

- a routing library built on top of the react which is used to create the routing in react apps
- react router lets you handle outing **declaratively**
- react router before v4: static
- react router after v4: dynamic

React router

- is the most popular 3rd party library in the React ecosystem.
- in fact, during the last 6 months, it has been included in 44% of all React projects

React router

- created in 2014
- is a declarative
- component based
- client and server-side routing library for React
- as React gives you a declarative and composable API for adding to and updating application state, React Router gives you a declarative and composable API for adding to and updating the user's navigation history.

React router

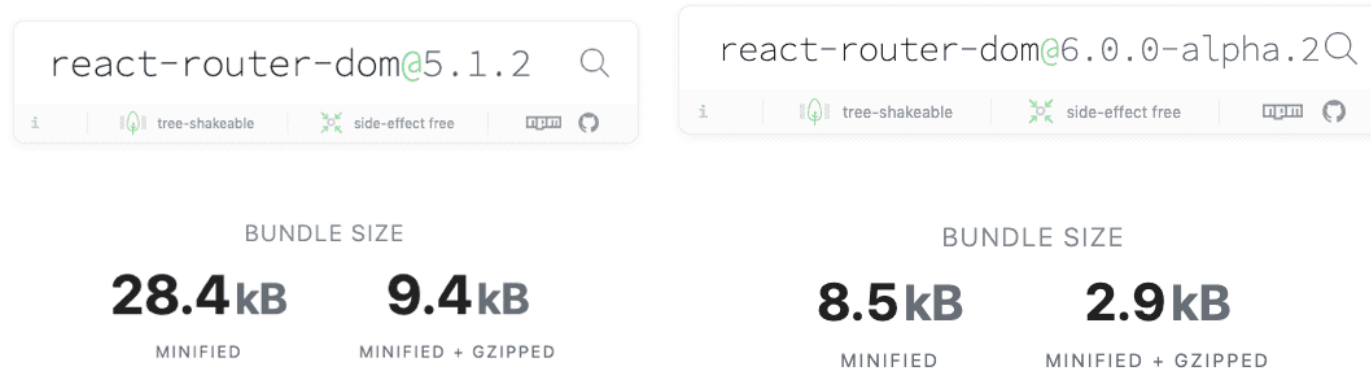
three packages:

- react-router: the core package for the router
- react-router-dom: on a website
- react-router-native: on a mobile app development environment using React Native

installation: `npm install --save react-router-dom`

Smaller Bundles

- Router v6 is a lot smaller than its predecessor
- react-router-dom@5.1.2 vs. react-router-dom@6.0.0-alpha.2 reveals the total bundle size decreased by 70%
- smaller bundles means your app loads more quickly, especially over slow/poor network connections.



Basic components

three types of components:

- router components
- route matching components
- navigation components

```
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';
```

BrowserRouter

- Naturally, in order to do its thing, React Router needs to be both aware and in control of your app's location. The way it does this is with its BrowserRouter component.
- Under the hood, BrowserRouter uses both the history library as well as React Context. The history library helps React Router keep track of the browsing history of the application using the browser's built-in history stack, and React Context helps make history available wherever React Router needs it.
- There's not much to BrowserRouter, you just need to make sure that if you're using React Router on the web, you wrap your app inside of the BrowserRouter component.

BrowserRouter

```
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter } from 'react-router-dom';
import App from './App';

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

BrowserRouter

a router that uses the HTML5 history API (pushState, replaceState and the popstate event) to keep your UI in sync with the URL

Props:

- basename: string
- getUserConfirmation: func
- forceRefresh: bool
- keyLength: number
- children: node

```
<BrowserRouter  
  basename={optionalString}  
  getUserConfirmation={optionalFunc}  
  forceRefresh={optionalBool}  
  keyLength={optionalNumber}  
>  
  <App/>  
</BrowserRouter>
```

The other <Routers />

- `MemoryRouter` keeps track of the history of the application in memory, rather than in the URL. Use this instead of `BrowserRouter` if you're developing a React Native application.
- `StaticRouter`, as the name implies, is useful in environments where the app's location never actually changes, like when rendering a single route to static HTML on a server.

Route

Route allows us to map the app's location to different React components.

```
<Route path='/dashboard' element={<Dashboard />} />
```

They can be rendered many routes as you'd like.

```
<Route path="/" element={<Home />} />  
<Route path="/about" element={<About />} />  
<Route path="/settings" element={<Settings />} />
```

With Route elements in this configuration, it's possible for multiple routes to match on a single URL. It might be needed to do that sometimes, but most often it is needed React Router to only render the route that matches best. This can be easily done with Routes.

Routes

Routes are the metaphorical conductor of the routes. Whenever you have one or more Routes, you'll most likely want to wrap them in a Routes.

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

function App () {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/about" element={<About />} />
      <Route path="/settings" element={<Settings />} />
      <Route path="*" element={<NotFound />} />
    </Routes>
  );
}
```

The reason for this is because it's Routes job is to understand all of its children Route elements, and intelligently choose which ones are the

Navigation

`<Link>` component:

- creates links in your application
- wherever you render a `<Link>`, an anchor (`<a>`) will be rendered in your application's HTML

```
<Link to="/">Home</Link>
```

Navigation

`<NavLink>` component:

- a special type of `<Link>` that can style itself as 'active'
- when its `to` prop matches the current location

```
<NavLink to="/" activeClassName='yeah'>Home</NavLink>
```

Link

```
<nav>
  <Link to="/">Home</Link>
  <Link to="/about">About</Link>
  <Link
to="/settings">Settings</Link>
</nav>
```

Pass an object as Link props for more control. Doing so allows you to add a query string via the search property or pass along any data to the new route via state.

To tell Link what path to take the user to when clicked, you pass it a to prop.

```
<nav>
  <Link to="/">Home</Link>
  <Link to="/about">About</Link>
  <Link to={{
    pathname: '/settings',
    search: '?sort=date',
    state: { fromHome: true },
  }}>Settings</Link>
</nav>
```

Pass props to Link

To pass data through a Link component to a new route, use Link's state prop.

```
<Link to="/onboarding/profile" state={{ from: "occupation "}}>  
  Next Step  
</Link>
```

Data which is passed via the state prop, that data will be available on the location's state property, which you can get access to by using the custom useLocation Hook that comes with React Router.

Pass props to Link

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

function Profile () {
  const location = useLocation();
  const { from } = location.state;

  return (
    ...
  );
}
```

URL Parameters

Like function parameters allow to declare placeholders when a function is defined, URL Parameters allow to declare placeholders for portions of a URL.

The way you tell React Router that a certain portion of the URL is a placeholder (or URL Parameter), is by using a `:` in the Route's path prop.

```
<Route path='/article/:topicId' element={<Article />} />
```

URL Parameters

Now whenever anyone visits a URL that matches the `/wiki/:topicId` pattern (`/article/javascript`, `/article/react`, `/article/anything`), the Article component is rendered.

React Router comes with a `useParams` Hook that returns an object with a mapping between the URL parameter(s) and its value.

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

function Article () {

  const { topicId } = useParams();
  ...
}
```


Nested Routes

Nested Routes allow the parent Route to act as a wrapper and control the rendering of a child Route.

```
function App () {  
  return (  
    <Routes>  
      <Route path="/" element={<Home />} />  
      <Route path="/messages/*" element={<Messages />} />  
      <Route path="/settings" element={<Settings />} />  
    </Routes>  
  );  
}
```

By appending a `/*` to the end of our `/messages` path, we're essentially telling React Router that Messages has a nested Routes component and our parent path should match for `/messages` as well as any other location that matches the `/messages/*` pattern.

Nested Routes

```
function Messages () {  
  return (  
    <Container>  
      <Conversations />  
  
      <Routes>  
        <Route path=':id' element={<Chat />} />  
      </Routes>  
    </Container>  
  );  
}
```

React router

How App component can contain all the information it needed to create nested routes rather than having to do it inside of Messages?

```
function App () {  
  return (  
    <Routes>  
      <Route path="/" element={<Home />} />  
      <Route path="/messages" element={<Messages />}>  
        <Route path=:id element={<Chats />} />  
      </Route>  
      <Route path="/settings" element={<Settings />} />  
    </Routes>  
  );  
}
```

React router

The child Route is now relative to the parent, so the parent (/messages) path doesn't need to include it.

Now, the last thing you need to do is tell React Router where in the parent Route (Messages) should it render the child Route (Chats). For this React Router's Outlet component is used.

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

function Messages () {
  return (
    <Container>
      <Conversations />
      <Outlet />
    </Container>
  )
}
```

Pass props to Router Components

A prop is passed to the component as you normally would.

```
<Route  
  path='/dashboard'  
  element={<Dashboard authenticated={true}/>}  
>
```

Programmatically Navigate

React Router offers two different ways to programmatically navigate, depending on your preference.

- the imperative navigate method
- the declarative Navigate component

To get access to the imperative navigate method, React Router's `useNavigate` Hook is used. From there, you can pass navigate the new path you'd like the user to be taken to when navigate is invoked.

Programmatically Navigate

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

function Register () {
  const navigate = useNavigate();

  return (
    <div>
      <h1>Register</h1>
      <Form afterSubmit={() => navigate('/dashboard')} />
    </div>
  );
}
```

Programmatically Navigate

A more declarative approach is available, using React Router's `Navigate` component.

`Navigate` works just like any other React component, however, instead of rendering some UI, it navigates the user to a new location.

```
import { Navigate } from 'react-router-dom';  
  
....  
<Navigate to='/dashboard'/>
```


Query Strings

React Router comes with a custom `useSearchParams` Hook which is a small wrapper over `URLSearchParams`.

`useSearchParams` returns an array with the first element being an instance of `URLSearchParams` and the second element being a way to update the query string.

```
/search?type=image&src=facebook-site
```

Query Strings

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

const Results = () => {
  const [searchParams, setSearchParams] = useSearchParams();

  const type = searchParams.get('type');
  const src = searchParams.get('src');

  return (
    ...
  );
};
```

404 Page

Route with a path of `*` is needed, and React Router will make sure to only render the element if none of the other Routes match.

```
<Routes>
  <Route path="/" element={<Home /> } />
  <Route path="/about" element={<About /> } />
  <Route path="/settings" element={<Settings /> } />
  <Route path="*" element={<NotFound /> } />
</Routes>
```

Protected Routes

For every route being private, instead of giving our Routes element prop the component we want it to render directly, a RequireAuth component is a wrapper for this components.

```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/pricing" element={<Pricing />} />
  <Route path="/dashboard"
    element={
      <RequireAuth>
        <Dashboard />
      </RequireAuth>
    }
  />
</Routes>
```

React router


The RequireAuth is the following:

```
function RequireAuth({ children }) {  
  const { authed } = useAuth();  
  const location = useLocation();  
  
  return authed === true  
    ? children  
    : <Navigate  
      to="/login"  
      replace  
      state={{ path: location.pathname }}  
    />;  
}
```

Route Config

React Router v6 comes with a `useRoutes` Hook that makes collocating your routes into a central route config not only possible, but also simple with a first class API.

```
/
/invoices
  :id
  pending
  complete
/users
  :id
  settings
```



```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/invoices" element={<Invoices />}>
    <Route path=":id" element={<Invoice />} />
    <Route path="pending" element={<Pending />} />
    <Route path="complete" element={<Complete />}
  />
</Route>
<Route path="/users/*" element={<Users />} />
</Routes>
```

useRoutes

With `useRoutes`, instead of declaring routes using React elements (JSX), it can be done using JavaScript objects.

`useRoutes` takes in an array of JavaScript objects which represent the routes in your application. Similar to the React element API with `<Route>`, each route has a `path`, `element`, and an optional `children` property.

useRoutes

```
const Routes = () => useRoutes([
  { path: '/', element: <Home /> },
  {
    path: '/invoices',
    element: <Invoices />,
    children: [
      { path: ':id', element: <Invoice /> },
      { path: '/pending', element: <Pending /> },
      { path: '/complete', element: <Complete /> }
    ]
  },
  ...
])
```


useRoutes

```
....  
{  
  path: '/users',  
  element: <Users />,  
  children: [  
    { path: ':id', element: <Profile /> },  
    { path: '/settings', element: <Settings /> }  
  ]  
}  
])
```

Benefits of React Router

- add routing to different views/components on Single Page Applications
- composable
- easily add links after designing the webpage
- react router conditionally renders certain components depending on the route from the URL