Code.Hub

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
```

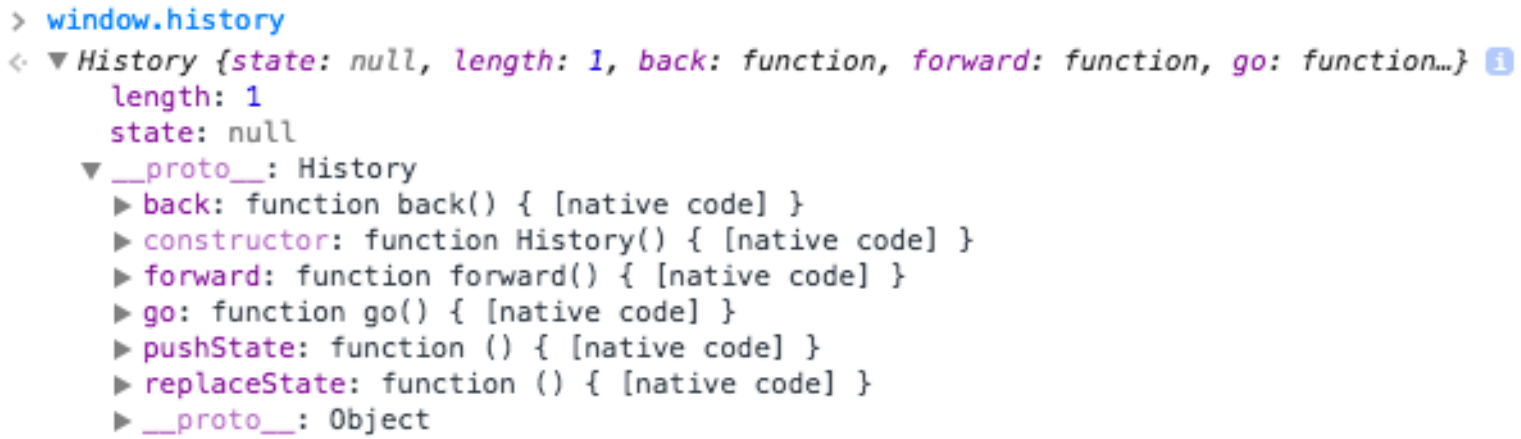
Listen

- 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



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

Code.Hub

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
```

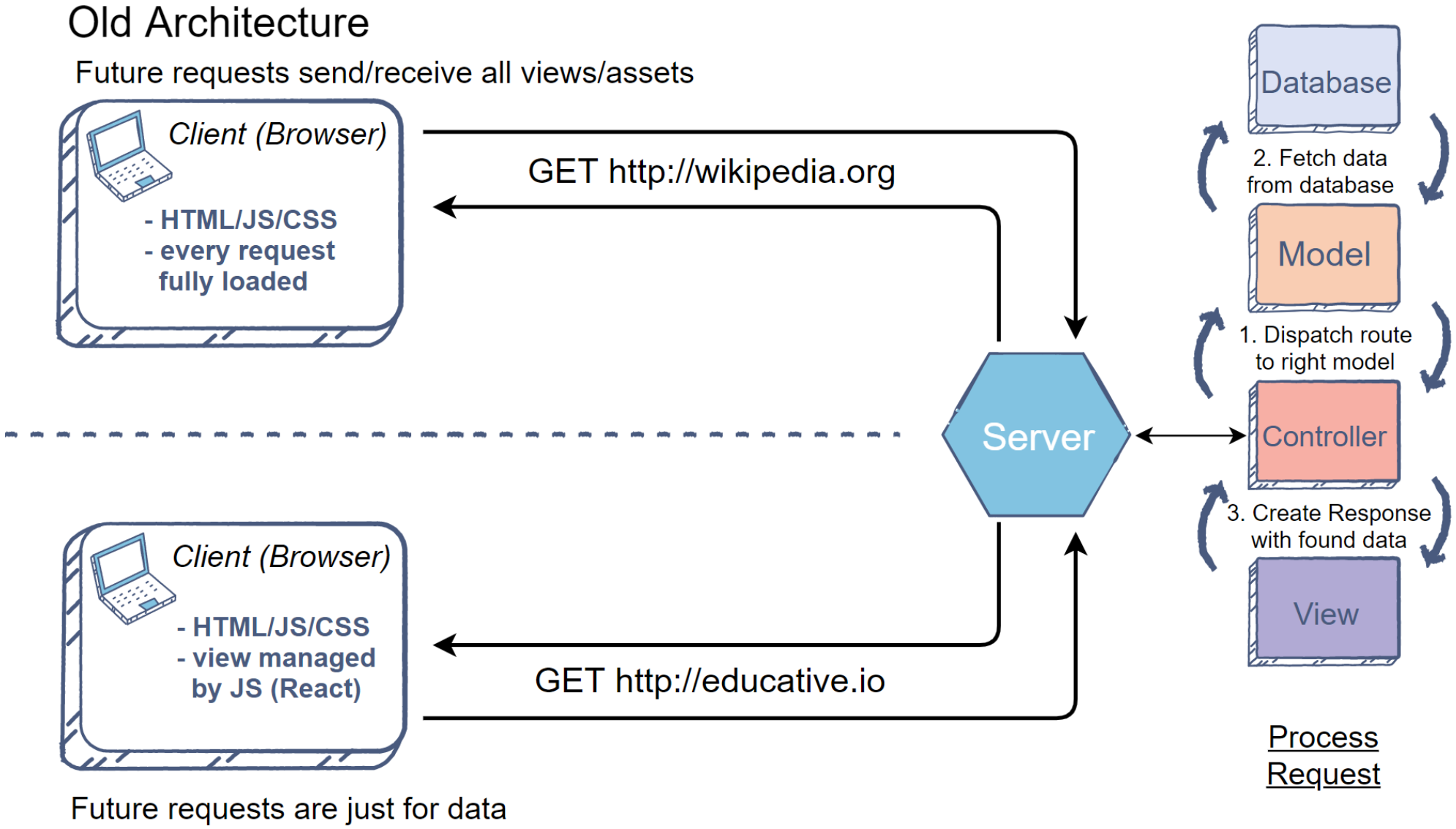
ember

```
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



Code.Hub

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?

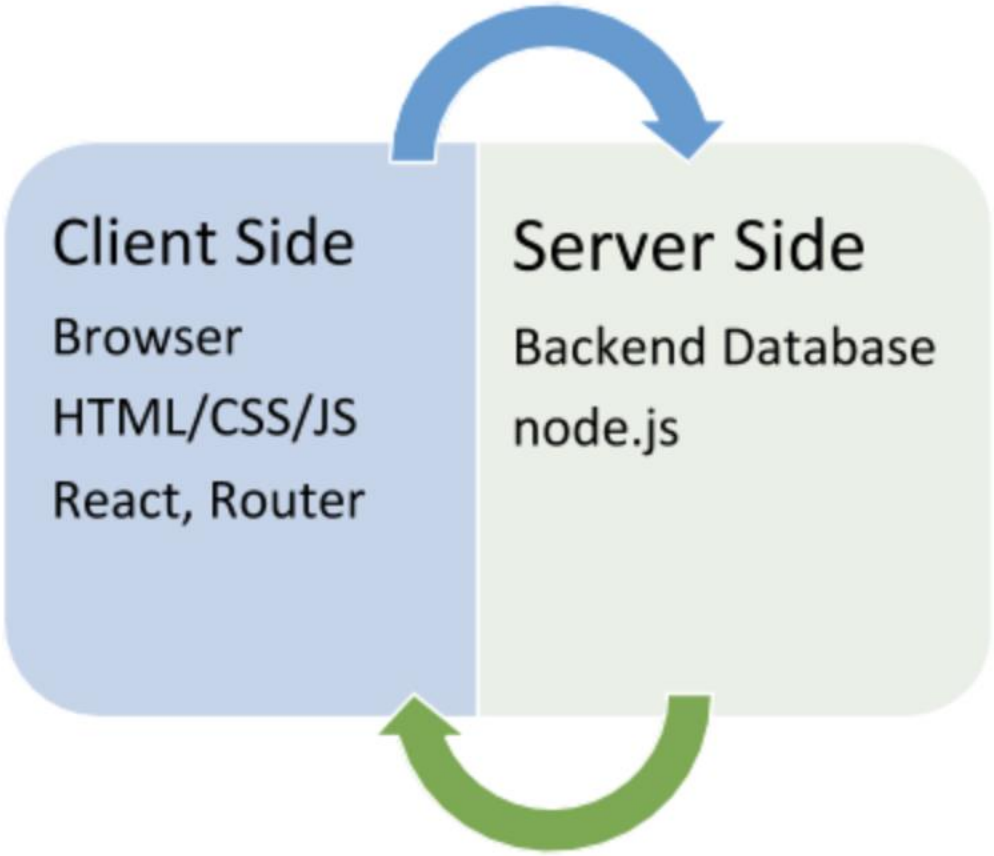
- is a library that allows you to handle routes in a web app, using dynamic routing
- implements a component-based approach to routing
- provides different routing components according to the needs of the application and platform

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.

Client vs Server Side



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

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

Basic components

three types of components:

- router components
- route matching components
- navigation components

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

Routers

- at the core of every react router application should be a router component

- there are two types of router
  - <BrowserRouter>: having a server that responds to requests
  - <HashRouter>: using a static file server

- every router creates specialized history object to keep track of the current location of the page

Routers difference

the primary difference between them is evident in the URLs that they create:

```
https://www.url.com/home //
<BrowserRouter>
```

<BrowserRouter>: uses the HTML5 History API to keep track of your router history

<HashRouter>: uses the hash portion of the URL (window.location.hash) to remember things

<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>
```

<HashRouter>

a router that uses the hash portion of the URL (i.e. window.location.hash) to keep your UI in sync with the URL

Props:

- basename: string
- getUserConfirmation: func
- hashType: string
- children: node

```
<HashRouter
  basename={optionalString}
  getUserConfirmation={optionalFunc}
  hashType={optionalString}>
  <App/>
</HashRouter>
```

Rendering a Router

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

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

Routes

- the most important component in react router
- renders some UI if the current location matches the route's path
- should have a prop named path, and if the pathname is matched with the current location, it gets rendered
- when the path does not match, the route will not render anything

Route matching

- <Route>
- <Switch>

In the Route component, we need to pass the two props:

- path: it means we need to specify the path

- component: which component user needs to see when they will navigate to that path

```
<Route exact={true} path="/menu" component={Home}/>
```

Path

```
<Route path='/menu' />
```

- when the pathname is '/', the path does not match
- when the pathname is '/menu ' or '/menu/1', the path matches

match only '/menu', use the "exact" prop. The following will match '/menu', but not '/menu/1':

```
<Route exact path='/menu' />
```

Path

about matching routes, react router only cares about the pathname of a location:

```
http://www.url.com/my-home/one?call=false
```

the only part that react router attempts to match is /my-home/one

Matching paths

- uses the path-to-regexp package to determine if a route element's path prop matches the current location
- compiles the path string into a regular expression, which will be matched against the location's pathname

Matching paths

when the route’s path matches, a match object with the following properties will be created:

- url: the matched part of the current location’s pathname
- path: the route's path
- isExact: path === pathname
- params: an object containing values from the pathname that were captured by path-to-regexp

Props from <Route>

the element rendered by the <Route> will be passed a number of props:

- the match object
- the current location
- the history object

Creating routes

/: homepage
/menu: the menu
/menu/:number: a profile for the menu
/about: content about website

```
<Route exact path='/' component={Home}/>
 {/* both /menu and /menu/:number begin with /menu */}
 <Route path='/menu' component={Menu}/>
 <Route path='/about' component={About}/>
```

<Route>

- is perhaps the most important component in react router
- its most basic responsibility is to render some UI when a location matches the route's path

Props:

- Component
- render: func
- children: func
- exact: bool
- strict: bool
- sensitive: bool
- path: string
- location: object

Route Rendering Props

- three props for rendering:
  - component
  - render
  - children
- only one should be provided to a <Route> element

Route Rendering Props

- *component*: a React component. When a route with a component prop matches, the route will return a new element whose type is the provided React component
- *render*: a function that returns a React element. This is similar to component, but is useful for inline rendering and passing extra props to the element
- *children*: a function that returns a React element. Unlike the prior two props, this will always be rendered, regardless of whether the route's path matches the current location

Rendering a Route

```
<Route path='/page' component={Page} />

const extraProps = { color: 'black' }
<Route path='/page' render={(props) => (
  <Page {...props} data={extraProps}/>
)}/>

<Route path='/page' children={(props) => (
    props.match ?
      <Page {...props}/> :
      <EmptyPage {...props}/>
)}/>
```

exact / strict / sensitive

exact: bool

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

strict: bool

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

sensitive:

path	location.pathname	sensitive	matches?
/one	/one	true	yes
/One	/one	true	no
/One	/one	false	yes

Code.Hub

404 page

- called not found page
- when a user navigates to the wrong path that doesn’t present in the website we need to show the not found page

<Switch>

- grouping <Route>s together
- is not required for grouping <Route>s
- will iterate over all of its children <Route> elements and only render the first one that matches the current location

Props:

- location: object
- children: node

Rendering Switch

```
<Switch>
  <Route exact path='/' component={Home}/>

  <Route path='/about' component={About}/>

  <Route component={NoMatch}/>
</Switch>
```

Nested routes

```
function Menu() {
  return (
    <Switch>
     <Route exact path='/menu' component={GenMenu}/>

     <Route path='/menu/:number' component={IndMenu}/>

    </Switch>
  );
}
```

Path Params

- :number part of the path /menu/:number: the part of the pathname that comes after /menu/ will be captured and stored as match.params.number

- the path name /menu/1 will generate a params object: { number: '1' }

Nested routes

Within the <Menu> component we will render routes for two paths:

- /menu: should only be matched when the pathname is exactly /menu, so we should also give that route element the exact prop
- /menu/:number: route uses a path param to capture the part of the pathname that comes after /menu

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>
```

Navigation

<Redirect> component:

- to force navigation

- when a <Redirect> renders, it will navigate using its to prop

```
<Redirect to='/' />
```

<Link>

- links to a new location
- navigate between pages
- create links using anchor elements, clicking on them would cause the whole page to reload. React Router provides a <Link> component to prevent that from happening

Rendering Link

```
import { Link } from 'react-router-dom';
..//..
<nav>
  <ul>
    <li><Link to='/'>Home</Link></li>
    <li><Link to='/menu'>Menu</Link></li>
    <li><Link to='/about'>About</Link></li>
  </ul>
</nav>
..//..
```

<Link>

Provides declarative, accessible navigation around your application

Props:

- to: string/object (containing a combination of pathname, search, hash, and state properties)
- replace: bool
- innerRef: function
- others: pass props to be on the <a> such as a title, id, className, etc.

<Link>

```
<Link to='/menu?sort=id' />

<Link to={{
  pathname='/menu',
  search='?sort=id',
  hash='#number1',
  state={isLoggedIn: true}
}}/>

<Link to='/menu' replace />

const refCallback = () => {};
<Link to='/' innerRef={refCallback} />
```

<NavLink>

- special version of the <Link>
- add styling attributes to the rendered element when it matches the current URL

Props:

- activeClassName: string
- activeStyle: object
- exact: bool
- strict: bool
- isActive: func
- location: object
- ariaCurrent: string

<NavLink>

```
<NavLink to='/menu' >Menu</NavLink>

<NavLink to='/menu‘ activeStyle={{
color='white'}}>
  Menu
</NavLink>

<NavLink exact to='/menu'>Menu</NavLink>
```

<Redirect>

- navigate to a new location
- the new location will override the current location in the history stack

Props:

- to: string/object
- push: bool
- from: string
- exact: bool
- strict: bool

```
<Redirect to='/menu' />

<Redirect to={{
  pathname='/menu', search='?sort=id',
  state={isLoggedIn: true}
}}>
  Menu
</Redirect>
```

<Redirect>

```
<Redirect push to='/menu' />

<Switch>
  <Redirect from='/menu1' to='/menu2' />
  <Route path='/menu2' component={Menu} />
<Switch>

<Switch>
  <Redirect from='/menu/:id' to='new/menu/:id' />
  <Route path='new/menu/:id' component={Menu} />
<Switch>
```

match

- contains information about how a <Route path> matched the URL
- contain the following properties:
  - params
  - isExact
  - path
  - url

history

- refers to the history package
- provides several different implementations for managing session history in JavaScript in various environments

history

contain the following properties:

- length
- action
- location: May have the following properties:
  - pathname
  - search
  - hash
  - state

- push(path, [state])
- replace(path, [state])
- go(n)
- goBack()
- goForward()
- block(prompt)

React Router v5

Before React Router v5

- When you wanted to render the route and get router props for component:

- Or when you wanted to pass extra props

```
<Route path="/" component={Home}
```

```
<Route path="/" render={({ match }) =>
  <Profile match={match} mine={true} />}
/>
```

React Router v5

After React Router v5

```
<Route path="/">
  <Home />
</Route>
```

- no implicit passing of any props to Home component
- add any extra props to the Home component
- no re-mounting the component on every render

useHistory

- provides access to the history prop in React Router
- refers to the history package dependency that the router uses
- a primary use case would be for programmatic routing with functions, like push, replace, etc.

useHistory

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

function Home() {
  const history = useHistory();
  return <button onClick={() => history.push('/profile')}>Profile</button>;
}
```

useLocation

- provides access to the location prop in React Router
- it is similar to window.location in the browser itself, but this is accessible everywhere as it represents the Router state and location
- a primary use case for this would be to access the query params or the complete route string

useLocation

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

function Profile() {
  const location = useLocation();
  const currentPath = location.pathname;
  const searchParams = new URLSearchParams(location.search);
  return <p>Profile</p>;
}
```

useParams

- provides access to search parameters in the URL
- this was possible earlier only using match.params

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

function Profile() {
  const { name } = useParams();
  return <p>{name}'s Profile</p>;
}
```

useParams

```
function Dashboard() {
  return (
    <Fragment>
      <main>
        <Route path="/profile/:name"><Profile /></Route>
      </main>
    </Fragment>
  );
}
```

useRouteMatch

- provides access to the match object
- if it is provided with no arguments, it returns the closest match in the component or its parents
- a primary use case would be to construct nested paths

useRouteMatch

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

function Auth() {
  const match = useRouteMatch();
  return (
    <Fragment>
      <Route path={`${match.url}/home`}><Home/></Route>
      <Route path={`${match.url}/login`}><Login /></Route>
    </Fragment>
  );
}
```

Hooks makes them all easier

- own profile to be rendered at /profile
- somebody else's profile if the URL contains the name of the person /profile/tom or /profile/gal.

Without hooks, a Switch must be used, list both routes and customize them with props

Hooks makes them all easier

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

export default function App() {
  return (
    <Router>
      <Route path="/profile"><Profile /></Route>
    </Router>
  );}
```

Hooks makes them all easier

```
function Profile() {
  const match = useRouteMatch('/profile/:name');
  return match ?
        <p>{match.params.name}'s Profile</p> :
        <p>My own profile</p>;
}
```

Hooks makes them all easier

- avoid a lot of confusion and intricacies that came with the earlier patterns
- fewer unforced errors
- router code more maintainable
- easy way to upgrade to new React Router versions

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

Recapitulation

install **react-router-dom**: is the package used to implement the routing process in your web application

Meaningful parts

import the meaningful parts from package:

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

- in index.js

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

- in navigation component

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

- in display component

Wrap the main component

wrap your App component inside a BrowserRouter

```
<BrowserRouter>
 <App />
</BrowserRouter>
```

react router uses this component to make your application aware of the navigation (history, current path, etc)

Navigation with NavLink items

fill navigation component using NavLink items

```
<NavLink exact to="/url">Menu</NavLink>
```

- NavLink is different from Link because it handles the active class-naming
- exact: activation of the link only if the path is identical as the one given in the to property
- to: is where the user will be redirected when clicking this link

Display with Route items

fill display component with route items

```
<Route exact path="/url" component={Component} />
```

- Route is just a conditional rendering component
- exact: it will render the Component given in the component property, only if the current path is identical to the path property
- exact on each Route lets you render only one component at a time

Annotations

- locations are objects with properties to describe the different parts of a URL:
  // a basic location object:
  { pathname: '/', search: '', hash: '', key: 'abc123'
  state: {} }

- you can render a pathless <Route>, which will match every location

Annotations

- if you use the children prop, the route will render even when its path does not match the current location
- <Route> and <Switch> components can both take a location prop: allows them to be matched using a location that is different than the actual location