SECTION 6:
ROUTER

ROUTER

- Router is obviously very useful helper and library which provides routing utilities might be very convenient.
- React.js itself doesn’t provide any routing utilities, but there’re several free, open-sources libs that works perfectly well.
- We will discuss react-router which has been developed by React.js team: https://github.com/reactjs/react-router
- React Router keeps your UI in sync with the URL. It has a simple API with powerful features like lazy code loading, dynamic route matching, and location transition handling built right in.

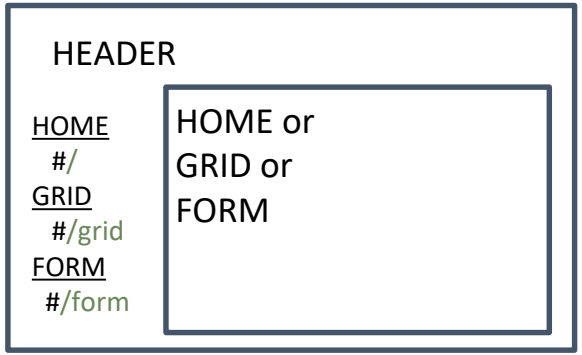
ROUTER PATH

- A route path is a string pattern that is used to match a URL (or a portion of one)
- Route paths are interpreted literally, except for the following special symbols:
  - :paramName – matches a URL segment up to the next /, ?, or #. The matched string is called a param
  - () – Wraps a portion of the URL that is optional
  - * – Matches all characters (non-greedy) up to the next character in the pattern, or to the end of the URL if there is none, and creates a splat param
  - ** - Matches all characters (greedy) until the next /, ?, or # and creates a splat param

ROUTER PATH EXAMPLE

```
<Route path=”#/hello/michael">          // matches /hello/michael
<Route path=”#/hello/:name">       // matches /hello/michael and /hello/ryan
<Route path=”#/hello(/:name)">      // matches /hello, /hello/michael, and /hello/ryan
<Route path=”#/files/*.*">        // matches /files/hello.jpg and /files/hello.html
<Route path=”#/**/*.jpg">         // matches /files/hello.jpg and /files/path/to/file.jpg
```

ROUTER



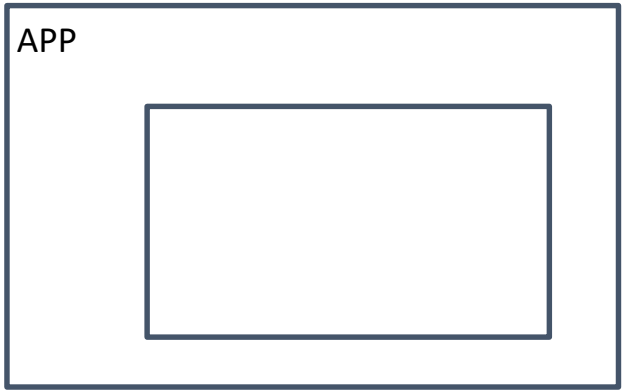
- At its heart, React Router is a component
- Lets create new screens (Home, Grid, Form) and define routes for them:
- In this example based on url appropriate component will be rendered
- hashHistory--it manages the routing history with the hash portion of the url

```
render(<Router/>, document.getElementById('app'))
```

```
render((
  <Router history={hashHistory}>
    <Route path="/" component={Home}/>
    {/* add the routes here */}
    <Route path="/grid" component={Grid}/>
    <Route path="/form" component={Form}/>
  </Router>
), document.getElementById('app'))
```

```
<ul role="nav">
  <li><Link to="/grid">Grid</Link></li>
  <li><Link to="/form">Form</Link></li>
</ul>
```

ROUTER

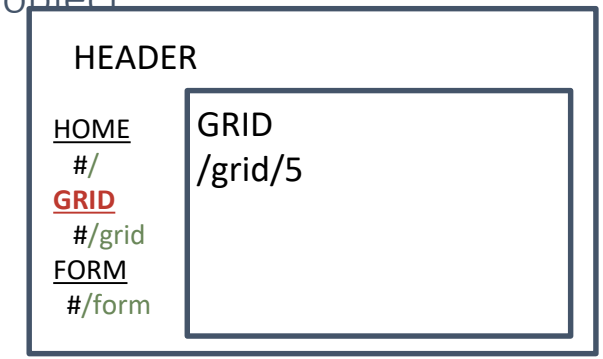


- Prev. example force you to have this routes render at every screen, which is not good. Lets fix it
- We can consider app as Route of Routes: / —> /grid —> grid/1 —> grid/2/columns—>…
- Base on that we can create nested routes and call define them only once:
- The best way to build large things is to stitch small things together.

```
render((
  <Router history={hashHistory}>
    <Route path="/" component={App}>
      {/* make them children of `App` */}
      <Route path="/grid" component={Repos}/>
      <Route path="/form" component={About}/>
    </Route>
  </Router>
), document.getElementById('app'))
```

ROUTER CONFIGURATION

- Would you like to make link active when it's clicked? You can use activeStyle and activeClassName props for it:
- Lets say you need to pass "id" in url params:
- Pick it id from url? Look into params object:




```
<Link to="/grid" activeStyle={{ color: 'red' }}>Grid</Link>
```

```
<Link to="/grid" activeClassName="activeLink">Grid</Link>
```

```
render((
  <Router history={hashHistory}>
    <Route path="/" component={App}>
      <Route path="/grid" component={Repos}/>
      <Route path="/grid/:id" component={Repos}/>
  </Route>
</Router>
), document.getElementById('app'))
```

```
//Grid component
render() {
  return (
    <h2>{this.props.params.id}</h2>
  )
}
```

ROUTER CONFIGURATION

- What if you want predefine some route component? Use IndexRoute:
- It's the same as your server gives index.html when you are at "/"

```
render((
  <Router history={hashHistory}>
    <Route path="/" component={App}>
      <IndexRoute component={Home}/>
      <Route path="/grid" component={Grid}/>
      <Route path="/grid/:id" component={Grid}/>
  </Route>
</Router>
), document.getElementById('app'))
```

- Pick it id from url? Look into params object:

```
render() {
   return (
      <h2>{this.props.params.id}</h2>
   )
}
```

ROUTER CONFIGURATION

- If you want to get rid of hash, use browserHistory instead of hashHistory:
- One noticable catch here: Your server needs to be configured appropriately to handle such routes.
- Every time server gets request he needs to return the same page. React Router will handle everything else.
- Configuration with Node.js and express can looks like that:

```
render((
    <Router history={browserHistory}>
      {/* ... */}
    </Router>
), document.getElementById('app'))
```

```
// handle every other route with index.html, which will contain
// a script tag to your application's JavaScript file(s).
app.get('*', function (request, response){
  response.sendFile(path.resolve(__dirname, 'public',
'index.html'))
})
```

http://localhost:8080/#/grid/5
http://localhost:8080/grid/5 -> index.html
http://localhost:8080/style.css -> style.css

ROUTER HOOKS

- React Router allows you to add several hooks:
- RouterWillLeave Hook lets you do things you want to do before route changes

```
componentDidMount() {
  this.props.router.setRouteLeaveHook(this.props.route,
this.routerWillLeave)
},
routerWillLeave(nextLocation) {
  // return false to prevent a transition w/o prompting the user,
  // or return a string to allow the user to decide:
  if (!this.state.isSaved)
    return 'Your work is not saved! Are you sure you want to
leave?'
}
```

ROUTER HOOKS

- React Router allows you to add several hooks:
- RouterWillLeave hook lets you control router transition.
- Return False – to prevent
- Return string – to prompt
- Return true – to allow

```
componentDidMount() {
   this.props.router.setRouteLeaveHook(this.props.route,
this.routerWillLeave)
},
routerWillLeave(nextLocation) {
   if (!this.state.isSaved)
      return 'Your work is not saved! Are you sure you want to
leave?'
}
```

DYNAMIC ROUTING

- React Router does all of its path matching and component fetching asynchronously, which allows you to not only load up the components lazily, but also lazily load the route configuration
- Dynamic Routes can define next methods:
- getChildRoutes – Provides list of matched child route element to be rendered but asynchronous and receives the location
- getComponents – Provides list of components to be rendered by route
- getIndexRoute – Provides default component to render if no children matches the route, but asynchronous and receives the location

DYNAMIC ROUTING

- No imports required!
- First load is very fast!
- Pair it with webpack chunks

  and your project will be as fast as fast can be!

```
const rootRoute = {
 component: 'div',
 childRoutes: [ {
  path: '/',
  component: require('./components/App'),
  childRoutes: [
   require('./routes/Calendar'),
   require('./routes/Course'),
   require('./routes/Grades'),
   require('./routes/Messages'),
   require('./routes/Profile')
  ]
 } ]
}

render(
<Router history={browserHistory} routes={rootRoute} />,
 document.getElementById('example')
)
```