When building React applications, one thing developers don’t like to utilize is routing in React - usually because of the assumed learning curve involved. In this article, we are going to debunk that myth and show you how easy it is to implement routing and serving responsive routes in your React applications.

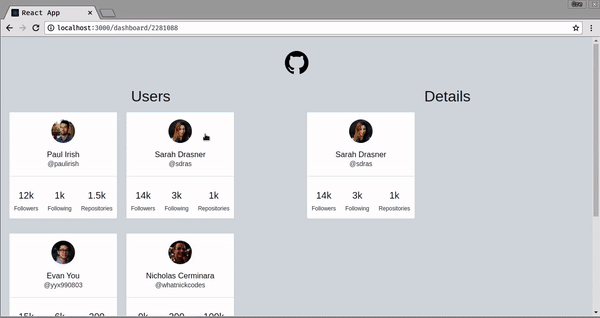
In layman’s terms, responsive routing is pretty much serving different routes to users based on the viewport of their device. CSS media queries are usually used to achieve this. The only issue with this is the fact that with media queries, you’re restricted to either showing or not showing different elements by using the CSS props. With responsive routes, you can now serve entire separate views our React applications to different users based directly on their screen sizes.

[**What we’ll build**](https://scotch.io/tutorials/conditional-routing-with-react-router-v4#toc-what-we-ll-build)

**Table of Contents**

* [**What we’ll build**](https://scotch.io/tutorials/conditional-routing-with-react-router-v4#toc-what-we-ll-build)
* [**Prerequisites**](https://scotch.io/tutorials/conditional-routing-with-react-router-v4#toc-prerequisites)
* [**Getting Started**](https://scotch.io/tutorials/conditional-routing-with-react-router-v4#toc-getting-started)
* [**Creating Navigation Component**](https://scotch.io/tutorials/conditional-routing-with-react-router-v4#toc-creating-navigation-component)
* [**Creating the User Cards**](https://scotch.io/tutorials/conditional-routing-with-react-router-v4#toc-creating-the-user-cards)
* [**Listing all Users**](https://scotch.io/tutorials/conditional-routing-with-react-router-v4#toc-listing-all-users)
* [**Creating the User Details View**](https://scotch.io/tutorials/conditional-routing-with-react-router-v4#toc-creating-the-user-details-view)
* [**Creating the Dashboard Component**](https://scotch.io/tutorials/conditional-routing-with-react-router-v4#toc-creating-the-dashboard-component)
* [**Responsive Routing**](https://scotch.io/tutorials/conditional-routing-with-react-router-v4#toc-responsive-routing)
* [**Conclusion**](https://scotch.io/tutorials/conditional-routing-with-react-router-v4#toc-conclusion)

In this tutorial, we will see how to build a simple user dashboard application that serves different routes to users based on the size of their device screens.



[**Prerequisites**](https://scotch.io/tutorials/conditional-routing-with-react-router-v4#toc-prerequisites)

To follow through adequately, you’ll need the following:

* NodeJS installed on your machine
* NPM installed on your machine

To confirm your installations, run the following command:

node --version

npm --version

If you get their version numbers as results, then you’re good to go

[**Getting Started**](https://scotch.io/tutorials/conditional-routing-with-react-router-v4#toc-getting-started)

**Installing React**

This article is based on React you need to have it installed on your machine. To install React, run the following command:

npm install -g create-react-app

Once this is done, your have successfully installed React on your machine. And we can go ahead to create our new each application by running the command:

create-react-app responsive-routing

cd responsive-routing

Next thing to do is to install the necessary modules we would need to successfully build this demo. These modules are the react-router-dom and react-media. We install this by running the command:

npm install react-router-dom react-media

Now, we can start the application by running the command:

npm start

[**Creating Navigation Component**](https://scotch.io/tutorials/conditional-routing-with-react-router-v4#toc-creating-navigation-component)

The Github logo in the center of the page serves as the navigation part of our application. Let's see how to make that. In your src/ folder, create a new folder called Nav and the following files as follows:

cd src

mkdir Nav

cd Nav && touch index.js Nav.css

You’ll need to add the Github logo and save it as logo.svg you can download it from [here](NULL)

Now, update the src/Nav/index.js file to look like this:

// src/Nav/index.js

import React from 'react';

import './Nav.css';

import logo from './logo.svg';

const Nav = () => (

<nav>

<img src={logo} alt="" />

</nav>

);

export default Nav;

The navigation component has the following styling:

**Free Node eBook**

Build your first Node apps and learn server-side JavaScript.

Top of Form

📧

GET THE EBOOK

Bottom of Form

https://track.mailerlite.com/webforms/o/1112154/c7r3j6?vaa4d608450783acdc64d5338ff94f6d5

/\*\* src/Nav/Nav.css \*\*/

nav {

display: flex;

justify-content: center;

height: 50px;

margin-bottom: 10px;

}

nav > img {

display: block;

width: 50px;

height: auto;

}

Now, let render the Nav component. To do this, let’s edit the default src/App.js file to look like this:

// src/App.js

import React, { Component } from 'react';

import Nav from './Nav';

= App extends Component {

render() {

return (

<div>

<Nav />

</div>

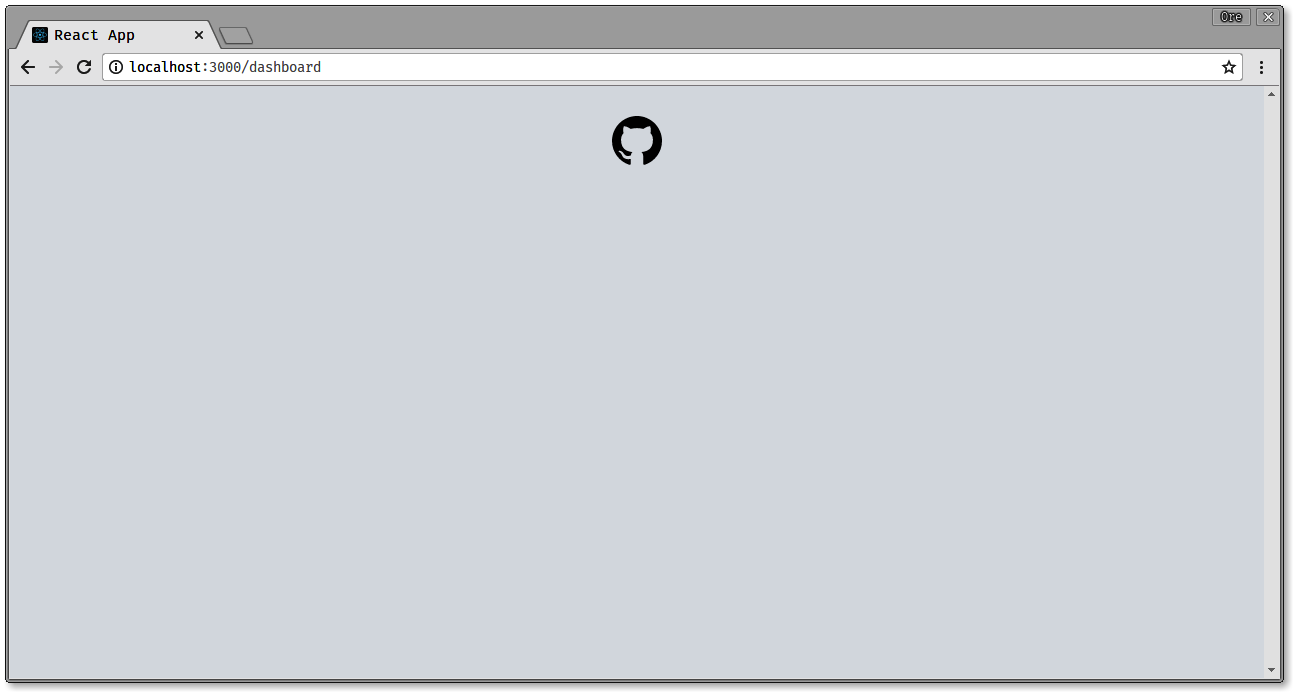
);

}

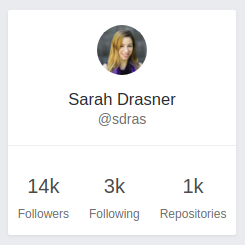
}

export default App;

Now, when we run our application using npm start and head over to the browser, we get the following:



[**Creating the User Cards**](https://scotch.io/tutorials/conditional-routing-with-react-router-v4#toc-creating-the-user-cards)



The user cards are responsible for displaying details of the user. Now, let’s see how to create this. In the src/ directory of your app, create a new Users folder and create the following files:

mkdir Users

cd Users && touch UsersCard.js UsersCard.css

Edit the UsersCard.js file to look like this:

// src/Users/UsersCard.js

import React from 'react';

import {Link} from 'react-router-dom';

import './UsersCard.css'

const UsersCard = ({ user, match }) => <Link to={`${match.url}/${user.id}`} className="column card">

<img src={user.avatar} alt=""/>

<p className="users-card\_\_name">{user.name}</p>

<p className="users-card\_\_username">@{user.username}</p>

<div className="users-card\_\_divider"></div>

<div className="users-card\_\_stats">

<div>

<p>{user.followers}</p>

<span>Followers</span>

</div>

<div>

<p>{user.following}</p>

<span>Following</span>

</div>

<div>

<p>{user.repos}</p>

<span>Repositories</span>

</div>

</div>

</Link>;

export default UsersCard;

If you pay attention to the code snippet above, we used the Link component from the react-router-dom to allow the user navigate to view details of a single user when the card is clicked. So, for a given user card with an id of 10009 and the Link component will generate a URL like this:

http://your-app/current-page/10009

* http://your-app/current-page represents existing URL
* 10009 represents the user id.

All this information will be passed when the component is rendered. The component has the following styling:

/\*\* src/Nav/UsersCard.css \*\*/

.card {

border-radius: 2px;

background-color: #ffffff;

box-shadow: 0 1.5px 3px 0 rgba(0, 0, 0, 0.05);

max-width: 228px;

margin: 10px;

display: flex;

flex-direction: column;

align-items: center;

padding: 0;

}

.card img {

width: 50px;

height: auto;

border-radius: 50%;

display: block;

padding: 15px 0;

}

.users-card\_\_name {

font-weight: 400;

font-size: 16.5px;

line-height: 1.19;

letter-spacing: normal;

text-align: left;

color: #25292e;

}

.users-card\_\_username {

font-size: 14px;

color: #707070;

}

.users-card\_\_divider {

border: solid 0.5px #efefef;

width: 100%;

margin: 15px 0;

}

.users-card\_\_stats {

display: flex;

}

.users-card\_\_stats p {

font-size: 20px;

}

.users-card\_\_stats div {

margin: 10px;

text-align: center;

}

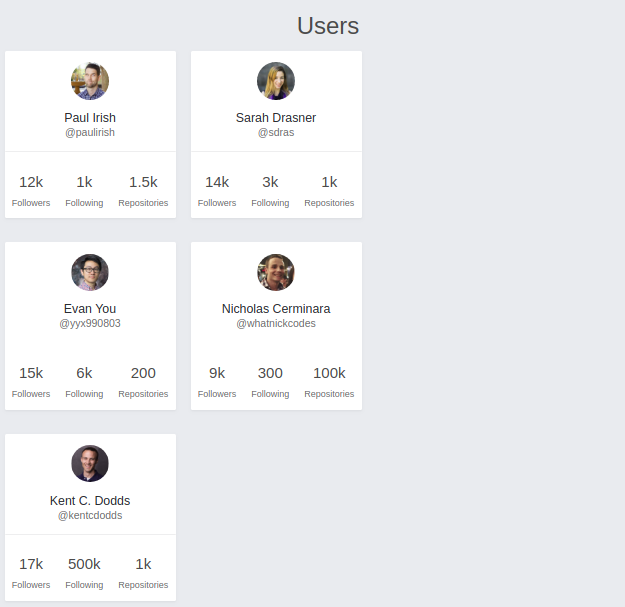
.users-card\_\_stats span {

color: #707070;

font-size: 12px;

}

[**Listing all Users**](https://scotch.io/tutorials/conditional-routing-with-react-router-v4#toc-listing-all-users)



What we see above is a listing of users. To get our application to look like this, we need to first create a UsersList component. In the src/Users directory, create the following files:

touch UsersList.js UsersList.css

To display the UserCards in the above format, we need to do some heavy lifting - don’t worry, i’ll be your hypeman.

Let’s edit the UsersList.js as follows. First, we make the necessary imports:

// src/Users/UsersList.js

import React from 'react';

import UsersCard from './UsersCard';

import './UsersList.css';

const listOfUsersPerRow = (users, row, itemsPerRow, match) =>

users

.slice((row - 1) \* itemsPerRow, row \* itemsPerRow)

.map(user => <UsersCard user={user} key={user.id} match={match} />);

const listOfRows = (users, itemsPerRow, match) => {

const numberOfUsers = users.length;

const rows = Math.ceil(numberOfUsers / itemsPerRow);

return Array(rows)

.fill()

.map((val, rowIndex) => (

<div className="columns">

{listOfUsersPerRow(users, rowIndex + 1, itemsPerRow, match)}

</div>

));

};

//...

The listOfUsersPerRow and listOfRows functions work hand in hand to ensure that we have not more than the specified number cards on each row. Next thing to do is then use the functions to create the listing of the users as follows

//src/Users/UsersList.js

//...

const UsersList = ({ users, itemsPerRow = 2, match }) => (

<div className="cards">

<h3 className="is-size-3 has-text-centered">Users</h3>

{listOfRows(users, itemsPerRow, match)}

</div>

);

export default UsersList;

The UsersList.css will look like this:

/\*\* src/Users/UsersList.css \*\*/

.cards {

margin-left: 20px;

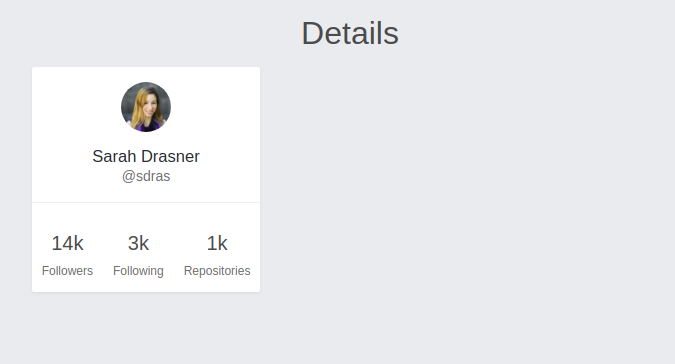
}

.columns {

margin-top: 0;

}

[**Creating the User Details View**](https://scotch.io/tutorials/conditional-routing-with-react-router-v4#toc-creating-the-user-details-view)



When a single user card is clicked from the listing of users, the single user card is displayed under a details section. Let’s see how to make this component.

Create the following files in the src/Users directory:

touch UsersDetails.js UsersDetails.css

Now, let’s add the following to the UsersDetails.js file:

// src/Users/UsersDetails.js

import React from 'react';

import UsersCard from './UsersCard';

const UsersDetails = ({ user, match }) => <div>

<h3 className="is-size-3 has-text-centered">Details</h3>

<UsersCard user={user} match={match} />

</div>;

export default UsersDetails;

[**Creating the Dashboard Component**](https://scotch.io/tutorials/conditional-routing-with-react-router-v4#toc-creating-the-dashboard-component)

The dashboard component is simple. We display the UserList and when a card is clicked, display the details on the side of the screen without having to reload the page. Let’s see how to make it work. Create a UsersDashboard.js file in the Users directory:

touch UserDashboard.js

Edit the UserDashboard.js to look as follows:

// src/Users/UsersDashboard.js

import React from 'react';

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

import UsersList from './UsersList';

import UsersDetails from './UsersDetails';

const UsersDashboard = ({ users, user, match }) => (

<div className="columns">

<div className="column">

<UsersList users={users} match={match} />

</div>

<div className="column">

<Route

path={match.url + '/:id'}

render={props => (

<UsersDetails

user={

users.filter(

user => user.id === parseInt(props.match.params.id, 10)

)[0]

}

match={match}

/>

)}

/>

</div>

</div>

);

export default UsersDashboard;

In the above, we used the Route component provided by react-router-dom as a component to display the specific user detail when the card is clicked.

Now, lets put this all together. Update the src/App.js file to look as follows:

// src/App.js

import React, { Component } from 'react';

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

import Nav from './Nav';

import UsersList from './Users/UsersList';

import UsersDetails from './Users/UsersDetails';

import UsersDashboard from './Users/UsersDashboard';

import './App.css';

class App extends Component {

state = {

users: [

{

id: 39191,

avatar: 'https://avatars0.githubusercontent.com/u/39191?v=4',

name: 'Paul Irish',

username: 'paulirish',

followers: '12k',

following: '1k',

repos: '1.5k'

},

//... other user data

]

};

render() {

return (

<div className="App">

<Nav />

<Route

path="/dashboard"

render={props => (

<UsersDashboard users={this.state.users} {...props} />

)}

/>

<Redirect from="/" to="/dashboard"/>

<Redirect from="/users" to="/dashboard"/>

</div>

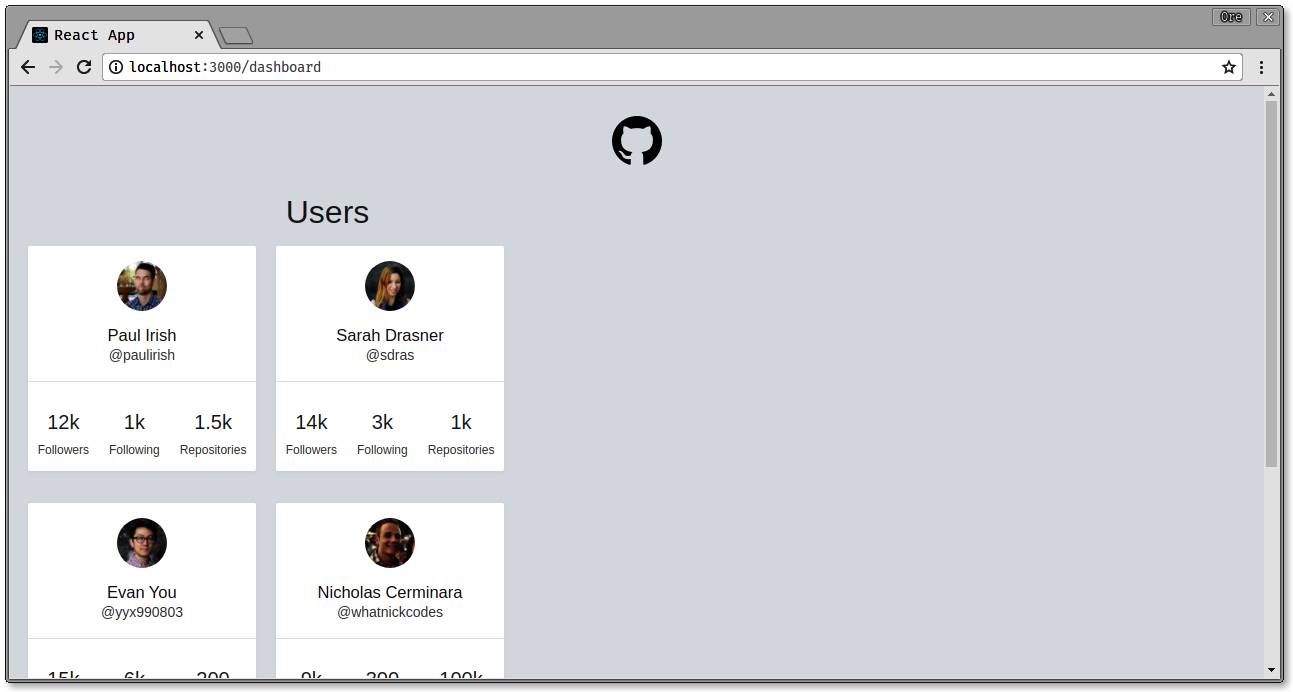
);

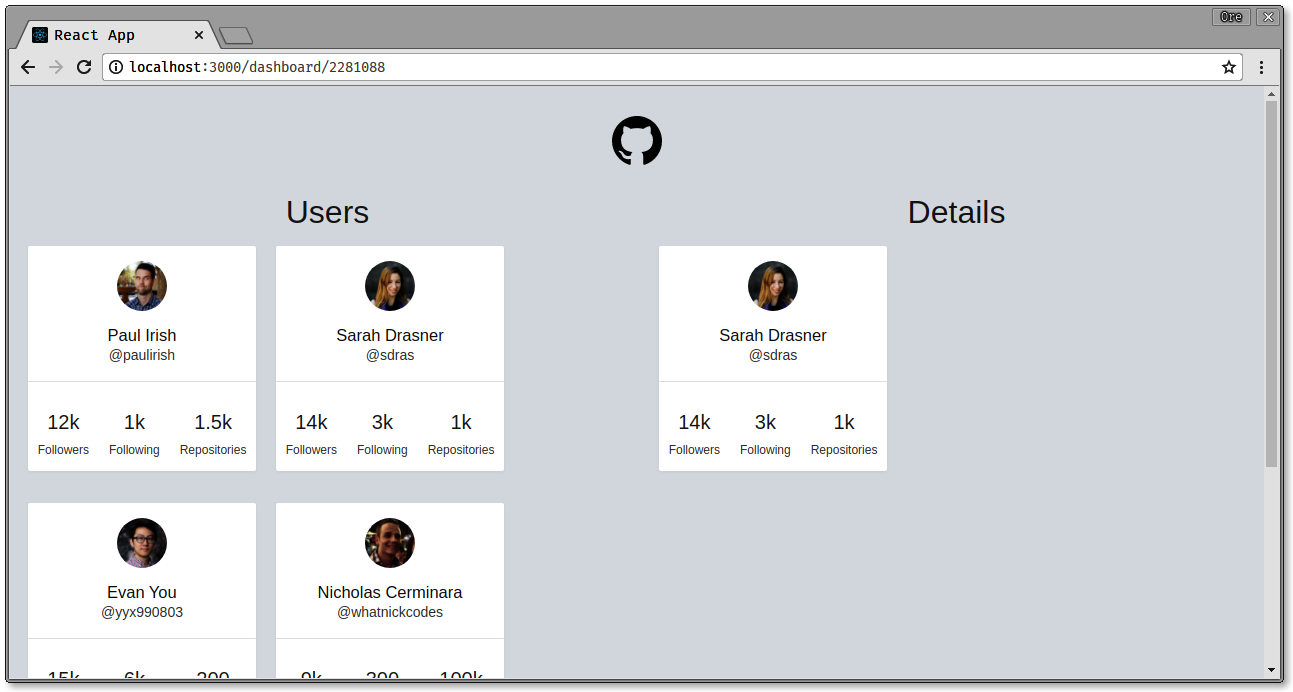
}

}

export default App;

When we go back to the browser, we get the following:





Note the difference in the URL when the user details are displayed

[**Responsive Routing**](https://scotch.io/tutorials/conditional-routing-with-react-router-v4#toc-responsive-routing)

Here’s where it all gets interesting, when users visit this application, no matter the screen size, they get this same view and functionality. In full-blown applications, it’s good to give the users experiences they can enjoy properly and one way to do that is to serve them views that match their exact device sizes. We are now going to take a look at how to do this in our application.

When visiting the application on a wide screen, the user is redirected to the /dashboardroute of the application and when viewing on a smaller screen, the user should be directed to the /users route of the application. Let’s see how to do this.

Update the src/App.js file to look like this:

// src/App.js

import React, { Component } from 'react';

import { Route, Switch, Redirect } from 'react-router-dom';

import Media from 'react-media';

import Nav from './Nav';

import UsersList from './Users/UsersList';

import UsersDetails from './Users/UsersDetails';

import UsersDashboard from './Users/UsersDashboard';

import './App.css';

class App extends Component {

//set application state

[...]

render() {

return (

<div className="App">

<Nav />

<Media query="(max-width: 599px)">

{matches =>

matches ? (

<Switch>

<Route

exact

path="/users"

render={props => (

<UsersList users={this.state.users} {...props} />

)}

/>

<Route

path="/users/:id"

render={props => (

<UsersDetails

user={

this.state.users.filter(

user =>

user.id === parseInt(props.match.params.id, 10)

)[0]

}

{...props}

/>

)}

/>

<Redirect from="/" to="/users"/>

<Redirect from="/dashboard" to="/users"/>

</Switch>

)

[...]

In the snippet above, we use the Media component to check the size of the screen. If the screen width is less than 599px, we set the set what we want to be displayed for different routes and also redirect the / and /dashboard routes to the /users route.

Now, if the screen size is greater than 599px we go ahead to display the full user dashboard as we did earlier

// src/App.js

[...]

: (

<Switch>

<Route

path="/dashboard"

render={props => (

<UsersDashboard users={this.state.users} {...props} />

)}

/>

<Redirect from="/" to="/dashboard"/>

<Redirect from="/users" to="/dashboard"/>

</Switch>

)

}

</Media>

</div>

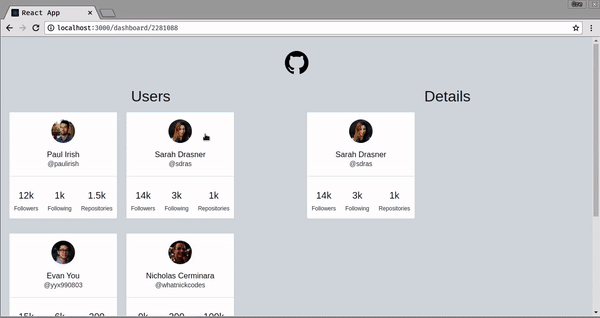
);

}

}

export default App;

Now, when we visit our application, our app works like this:



At this point, we can see that is a lot better to serve different routes based on screen sizes than using just media queries because you can now serve specially designed components to users based on their device sizes.

[**Conclusion**](https://scotch.io/tutorials/conditional-routing-with-react-router-v4#toc-conclusion)

In this article, we saw an introduction to component-based routing with React and how to implement conditional rendering in your React applications. Here’s a link to the full [Github](https://github.com/christiannwamba/responsive-routing)repository. Feel free to leave a comment or suggestion below