

# React-router

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

# URL

`http://www.example.com:80/path/to/myfile.html  
?key1=value1&key2=value2#SomewhereInTheDocument`

# URL

`http://www.example.com:80/path/to/myfile.html  
?key1=value1&key2=value2#SomewhereInTheDocument`

**Протокол.** Он отображает, какой протокол браузер должен использовать. Обычно это HTTP-протокол или его безопасная версия - HTTPS. Могут быть другие (`mailto:`, чтобы открыть почтовый клиент, или `ftp:` для запуска передачи файлов).

# URL

http://**www.example.com**:80/path/to/myfile.html  
?key1=value1&key2=value2#SomewhereInTheDocument

**Домен.** Какой веб-сервер должен быть запрошен.

# URL

`http://www.example.com:80/path/to/myfile.html  
?key1=value1&key2=value2#SomewhereInTheDocument`

**Порт.** технический параметр, используемый для доступа к ресурсам на веб-сервере.

# URL

`http://www.example.com:80/path/to/myfile.html  
?key1=value1&key2=value2#SomewhereInTheDocument`

**Путь.** адрес ресурса на веб-сервере. В прошлом, адрес отображал местоположение реального файла в реальной директории на веб-сервере. В наши дни это чаще всего абстракция, позволяющая обрабатывать адреса и отображать тот или иной контент из баз данных.

# URL

http://www.example.com:80/path/to/myfile.html  
**?key1=value1&key2=value2**#SomewhereInTheDocument

**Параметры запроса.** Дополнительные параметры, которые браузер сообщает веб-серверу. Эти параметры - список пар ключ/значение, которые разделены символом **&**.

# URL

`http://www.example.com:80/path/to/myfile.html  
?key1=value1&key2=value2#SomewhereInTheDocument`

**Якорь.** Это «ссылка» на другую часть того же самого ресурса. Якорь представляет собой вид "закладки" внутри ресурса, которая переадресовывает браузер на «заложенную» часть ресурса.



# Основные компоненты и пропсы



```
// Components  
<BrowserRouter />  
<Route />  
<Switch />  
<Link />  
<Redirect />  
  
// HOC  
withRouter(YourComponent)  
  
// props  
history  
location  
match
```

# BrowserRouter



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

const App = () => (
  <BrowserRouter>
    // anything here can be wrapped in withRouter
  </BrowserRouter>
);
```

# Link



```
const YourComponent = props => {  
  // some logic  
  return (  
    ...  
    <Link to="/path/to/somewhere/">  
      <AnyComponentYouWant />  
        
      or some text  
    </Link>  
  )  
}
```

# Route



```
const App = () => (  
  <BrowserRouter>  
    <Route path="/">  
      <Home />  
    </Route>  
  
    <Route path="/" component={Header} />  
  
    <Route path={{ pathname: "location/like/object/", search: '?  
key=value' }}>  
      <SomeComponent />  
    </Route>  
  
  </BrowserRouter>  
)
```

# Route



```
const Home = ({ history, location, match }) => {  
  // ...  
}
```

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

Любой компонент, который был отрисован компонентом Route, получает **history, location, match** в качестве пропсов

# Route



```
const App = () => {  
  return (  
    <BrowserRouter>  
      <Route path="/users/:userId">  
        <Users />  
      </Route>  
    </BrowserRouter>  
  )  
}  
  
const Users = ({ match }) => {  
  const { userId } = match.params;  
  
  return <div>Current user: {userId}</div>  
}
```

Dynamic paths  
↓

# Route



```
const App = ( ) => {  
  return (  
    <BrowserRouter>  
      <Route path="/">  
        <Users />  
      </Route>  
  
      <Route path="/tasks/">  
        <Tasks />  
      </Route>  
  
      <Route path="/tasks/:taskId/">  
        <SpecificTask />  
      </Route>  
    </BrowserRouter>  
  )  
}
```

pathname == '/tasks/123/'

Какие компоненты будут нарисованы?

# Route



```
const App = ( ) => {  
  return (  
    <BrowserRouter>  
      <Route path="/">  
        <Users />  
      </Route>  
  
      <Route path="/tasks/">  
        <Tasks />  
      </Route>  
  
      <Route path="/tasks/:taskId/">  
        <SpecificTask />  
      </Route>  
    </BrowserRouter>  
  )  
}
```

pathname == '/tasks/123/'

Какие компоненты будут нарисованы?

**Все 3: Users, Tasks, SpecificTask**



# Route



```
const App = ( ) => {  
  return (  
    <BrowserRouter>  
      <Route path="/">  
        <Users />  
      </Route>  
  
      <Route path="/tasks/">  
        <Tasks />  
      </Route>  
  
      <Route path="/tasks/:taskId/">  
        <SpecificTask />  
      </Route>  
    </BrowserRouter>  
  )  
}
```

pathname == '/tasks/123/'

**Какие компоненты будут нарисованы?**

**Все 3: Users, Tasks, SpecificTask**

**Рисуются все роуты сверху вниз  
без остановки.**

# Route exact



```
const App = () => {  
  return (  
    <BrowserRouter>  
      <Route exact path="/">  
        <Users />  
      </Route>  
  
      <Route exact path="/tasks/">  
        <Tasks />  
      </Route>  
  
      <Route exact path="/tasks/:taskId/">  
        <SpecificTask />  
      </Route>  
    </BrowserRouter>  
  )  
}
```

pathname == '/tasks/'

Какие компоненты будут нарисованы?

# Route



```
const App = () => {  
  return (  
    <BrowserRouter>  
      <Route exact path="/">  
        <Users />  
      </Route>  
  
      <Route exact path="/tasks/">  
        <Tasks />  
      </Route>  
  
      <Route exact path="/tasks/:taskId/">  
        <SpecificTask />  
      </Route>  
    </BrowserRouter>  
  )  
}
```

pathname == '/tasks/'

Какие компоненты будут нарисованы?

Только Tasks.

Однако

<Route exact path="/tasks/:taskId" />  
будет проверен, но т.к. path не подходит,  
SpecificTask не будет нарисован.

# Switch



```
const App = () => (  
  <BrowserRouter>  
    <Switch>  
      <Route exact path="/" component={Home} />  
      <Route exact path="/tasks/new" component={NewTask} />  
      <Route path="/tasks/new/preview" component={TaskPreview} />  
      <Route path="/tasks/:taskId" component={Task} />  
    </Switch>  
  </BrowserRouter>  
)
```

**Если роуты находятся в  
<Switch>, то будет  
нарисован первый  
подходящий, остальные  
не будут проверяться на  
совпадение path**

# Switch



Можно комбинировать!

```
const App = ( ) => (  
  <BrowserRouter>  
    <Route path="/tasks/" component={TasksHeader} />  
    <Switch>  
      <Route exact path="/" component={Home} />  
      <Route exact path="/tasks/new" component={NewTask} />  
      <Route path="/tasks/new/preview" component={TaskPreview} />  
      <Route path="/tasks/:taskId" component={Task} />  
    </Switch>  
  </BrowserRouter>  
) ;
```

# Redirect



```
<Redirect to="/some/path/" />
```

# Redirect



```
<Redirect from="/old/path" to="/new/path/" />  
<Route exact path="/new/path" component={Home} />
```

# Switch + Redirect



```
const App = () => {  
  return (  
    <BrowserRouter>  
      <Switch>  
        <Route exact path="/" component={Home} />  
        <Route exact path="/tasks/" component={Tasks} />  
        <Route path="/tasks/:taskId" component={SpecificTask} />  
  
        <Redirect to="/" />  
      </Switch>  
    </BrowserRouter>  
  )  
}
```



# location




```
location = {  
  pathname: '/path/to/somewhere/',  
  search: '?param=true&search=string',  
  hash: '#anchor',  
  state: {  
    [key: string]: any  
  }  
}
```

# match



```
match = {  
  params: { [key: string]: string }, // Key/value pairs parsed from the URL  
    corresponding to the dynamic segments of the path  
  path: string, // The path pattern used to match. Useful for building nested  
    <Route>s  
  url: string, // The matched portion of the URL. Useful for building nested  
    <Link>s  
  isExact: boolean, // true if the entire URL was matched (no trailing  
    characters)  
}
```

# history



```
history = {  
  push: (location, [state]) => Pushes a new entry onto the history  
stack,  
  replace: (path, [state]) => Replaces the current entry on the history  
stack,  
  ...  
}
```

# withRouter



```
const MySweetComponent = ({ history, location, match }) => {  
  // do cool stuff  
}  
  
const WrappedSweetComponent = withRouter(MySweetComponent);  
  
const App = () => (  
  <BrowserRouter>  
    <WrappedSweetComponent />  
  </BrowserRouter>  
);
```