

El objeto de esta práctica es afianzar, reforzar y consolidar los conocimientos teóricos presentados en la lección.

Al finalizarla, el estudiante:

- Habrá trabajado con componentes `<Link>`.
- Habrá configurado las propiedades que estilizan los enlaces activos.
- Habrá actualizado el elemento `<title>` de la aplicación.
- Habrá cambiado explícitamente de una ruta a otra.

Objetivos

El objetivo de la práctica es mostrar cómo trabajar más afondo con el encaminador y las vistas. Simularemos la práctica anterior, pero añadiremos algunas cosas más que asienten los conceptos teóricos presentados en esta lección.

Preparación del entorno

En la práctica anterior, utilizamos la historia *hash*, aquella que añade una almohadilla automáticamente a los URLs solicitados por los usuarios. Ahora, vamos a utilizar la historia de navegador, para lo que es necesario que nuestra aplicación sea servida por un servidor web. Por esta razón, vamos a dividir nuestra aplicación en dos proyectos: uno que contiene la aplicación servidora e implementaremos mediante **Node** y **Express**; y el otro, la aplicación cliente **React**.

Proyecto Express

Vamos a crear el servidor web:

1. Abrir una consola.
2. Instalar el generador de **Justo** para **Express**:

```
> npm install -g justo-generator-express
```
3. Crear el directorio de la práctica e ir a él.
4. Dentro del directorio de la práctica, crear un nuevo directorio `server` e ir a él.
5. Invocar el generador de **Justo** para **Express**:

```
> justo -g express
```

A la hora de responder, seleccionar el valor predeterminado salvo en las siguientes preguntas:

 - Cuando pregunte si usar **Handlebars**, indicar **N**.
 - Cuando pregunte **Static content max-age**, responder **1**.
 - Cuando pregunte si usar el *middleware* **cookie-parser**, indicar **N**.
 - Cuando pregunte qué tipo de sesión de estado usar, seleccionar `<none>`.
 - Cuando pregunte qué hacer cuando el usuario solicite un recurso no encontrado, seleccionar que envíe `public/index.html`.
6. Instalar dependencias:

```
> npm install
```
7. Generar la aplicación **Express**:

```
> justo build
```

La aplicación ejecutable se genera en el directorio `dist/es5/nodejs/server`.

8. Ir al directorio `dist/es5/nodejs/server`.
9. Instalar dependencias:
`> npm install`
10. Arrancar la aplicación servidora en modo desarrollo:
`> npm run start-dev`
11. Abrir el navegador e ir a `localhost:8080`.

Debe aparecer Hello World!

En esta consola, mantendremos el servidor web de nuestra aplicación **React** de prácticas. Manténgala abierta con el servidor en ejecución durante toda la práctica.

Proyecto React

Ahora, vamos a crear un proyecto para nuestra aplicación **React** mediante el generador de **Justo**:

1. Abrir una nueva consola.
2. Ir al directorio de la práctica.
3. Crear un nuevo directorio, `client` e ir a él.
Tanto `server` como `client` deben colgar directamente del directorio de la práctica.
4. Invocar el generador de **Justo** para **React**:

```
> justo -g react
```

A la hora de responder:

- Seleccionar **Y** cuando solicite si deseamos usar **React Router**.
- Seleccionar **browserHistory** como historia.
- Seleccionar el uso de **Font Awesome**.

5. Instalar las dependencias del proyecto:

```
> npm install
```

6. Editar el archivo **Justo.js**.
7. Ir a la tarea **deploy** e indicar que la aplicación se debe copiar en el directorio del servidor web `../server/dist/es5/nodejs/server/app/public`:

```
catalog.workflow(  
  {name: "deploy", desc: "Deploy dist to a location."},  
  function(params) {  
    const dst = params[0] || "../server/dist/es5/nodejs/server/app/public";  
  
    if (dst) {  
      copy(`Copy dist to ${dst}`, {  
        src: "dist/",  
        dst: dst  
      });  
    }  
  }  
);
```

8. Guardar cambios.
9. Invocar la tarea **build** del catálogo del proyecto para construir la aplicación:
`> justo build`
10. Invocar la tarea **deploy** para copiar la aplicación en el servidor web:
`> justo deploy`
11. Abrir el navegador e ir a `localhost:8080`.

Ahora, debe aparecer la aplicación **React**:

Creación de rutas

Vamos a crear las rutas:

1. Ir a la consola de la aplicación **React**.
2. Crear el archivo de rutas bands mediante el generador:

```
> justo -g react route file
```

Indicar:

- Nombre de archivo: bands.
- Punto de montaje o *path*: bands.
- Tipo de vista envoltorio (*layout view type*): **Immutable**.
- Tipo de vista índice (*index view type*): **Immutable**.

3. Añadir la vista Info a la carpeta /app/views/bands mediante el generador:

```
> justo -g react view
```

Esta vista la utilizaremos para mostrar la información de una determinada banda. Responda teniendo en cuenta lo siguiente:

- Seleccione bands como carpeta donde crearla.
- Indique Info como nombre de vista.
- Seleccione **Immutable** como tipo de componente.

4. Añadir la ruta /bands/:id que se asocie a la vista Info mediante el generador:

```
> justo -g react route
```

Responda como sigue:

- Archivo de rutas donde añadir la ruta: bands.
- *Path*: :id.
- Vista asociada: Info.

Definición de datos

Vamos a crear el archivo de datos de la aplicación:

1. Crear el archivo bands.js en la carpeta app/data:

```
export default {  
  "la-dama-se-esconde": {  
    name: "La Dama se Esconde",  
    year: 1985,  
    origin: "San Sebastián"  
  },  
  "la-habitacion-roja": {  
    name: "La Habitación Roja",  
    year: 1994,  
    origin: "Valencia"  
  },  
  "lori-meyers": {  
    name: "Lori Meyers",  
    year: 1998,  
    origin: "Granada"  
  },  
  "love-of-lesbian": {  
    name: "Love of Lesbian",  
    year: 1997,  
    origin: "Barcelona"  
  },  
  "neuman": {
```

```

    name: "Neuman",
    year: 1999,
    origin: "Murcia"
  }
};

```

Creación de componente tabla

Ahora, vamos a crear el componente tabla que muestra los datos:

1. Ir a la consola de la aplicación cliente **React**.
2. Crear el componente tabla mediante el generador:

```

> justo -g react component
Respuestas:

```

- Cuando pregunte en qué subcarpeta añadir el componente, seleccionar **<other>** y a continuación escribir **bands**.
- Nombre del componente: **BandTable**.
- Tipo de componente: **Immutable**.
- Tipo de estructura: **Simple**.
- Tipo de implementación: **Function**.

3. Editar el archivo `app/components/bands/BandTable.jsx`.
4. Importar el archivo de datos:

```
import bands from "../../data/bands";
```

5. Importar el componente **Link** de **React Router**:

```
import {Link} from "react-router";
```

6. Modificar la función de reproducción del componente:

```

export default function BandTable(props) {
  return (
    <table>
      <thead>
        <tr>
          <th>Mostrar</th>
          <th>Banda</th>
          <th>Año</th>
        </tr>
      </thead>

      <tbody>
        {
          Object.keys(bands).map(function(id) {
            var band = bands[id];

            return (
              <tr key={id}>
                <td>
                  <Link to={`/${bands}/${id}`}>
                    <span className="fa fa-edit" />
                  </Link>
                </td>
                <td>{band.name}</td>
                <td>{band.year}</td>
              </tr>
            );
          })
        }
      </tbody>
    </table>
  );
}

```

Observe el enlace **<Link>**.

7. Guardar cambios.

Definición de vistas

Las vistas ya las hemos creado mediante el generador, pero no las hemos modificado como corresponde para completar la práctica. Ha llegado el momento de hacerlo.

1. Editar la vista `app/views/bands/Layout.jsx`.
2. Ir al método `render()` y añadir el enlace para la vista New, que todavía no hemos creado:

```
render() {
  return (
    <div>
      <Link to="/bands/__new__"><span className="fa fa-plus" /> Nueva banda</Link>
      {this.props.children}
    </div>
  );
}
```

3. Guardar cambios.
4. Editar la vista `app/views/bands/Index.jsx`.
5. Importar el componente `BandTable`:

```
import BandTable from "../../components/bands/BandTable";
```

6. Ir al método `render()` y definirlo para que muestre la tabla de bandas:

```
render() {
  return (
    <BandTable />
  );
}
```

7. Ir al método `componentDidMount()` y fijar qué título debe mostrar el navegador cuando se presente la vista al usuario:

```
componentDidMount() {
  document.title = "Lista de bandas";
}
```

8. Guardar cambios.
9. Editar la vista `app/views/bands/Info.jsx`.
10. Importar los datos:

```
import bands from "../../data/bands";
```

11. Ir al método `render()` y hacer que muestre el contenido de la banda indicada en el parámetro `:id` de la ruta a la que está asociada la vista:

```
render() {
  var band;

  //(1) get data
  band = bands[this.props.params.id];

  //(2) set title
  document.title = `Banda: ${band.name}`;

  //(3) render
  return (
    <div>
      Artista: {band.name}<br />
      Año: {band.year}<br />
      Origen: {band.origin}
    </div>
  );
}
```

¿Por qué fijamos el título del documento aquí y no en el método `componentDidMount()`? Porque los datos son estáticos, proceden de un archivo `JavaScript`, al cual el componente accede directamente sin dejarlo en ninguna propiedad particular del componente. Por lo que el método

`render()` no parece un mal sitio donde hacerlo.

Si el título procede de una propiedad o estado del componente, se recomienda el método `componentDidMount()`.

12. Guardar cambios.

13. Editar el archivo `app/views/App.jsx`.

14. Añadir un enlace para poder acceder a la vista `/bands`:

```
render() {
  return (
    <div>
      <Link to="/bands"><span className="fa fa-list" /> Bandas</Link>
      {this.props.children}
    </div>
  );
}
```

15. Guardar cambios.

16. Ir a la consola.

17. Generar la aplicación y desplegarla en el servidor web:

```
> just build deploy
```

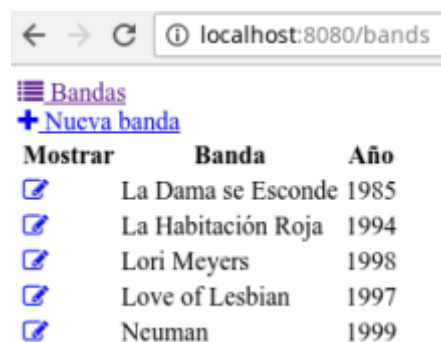
18. Abrir el navegador e ir a `localhost:8080`:

 [Bandas](#)
View:AppIndex

Observe el título del documento, es el fijado en el archivo `index.html`:

React app - Google Chrome

19. Hacer clic en el enlace `Bandas`:



Mostrar	Banda	Año
<input checked="" type="checkbox"/>	La Dama se Esconde	1985
<input checked="" type="checkbox"/>	La Habitación Roja	1994
<input checked="" type="checkbox"/>	Lori Meyers	1998
<input checked="" type="checkbox"/>	Love of Lesbian	1997
<input checked="" type="checkbox"/>	Neuman	1999

Observe la URL: `localhost:8080/bands`.

Observe que el título ha cambiado:

Lista de bandas - Google Chrome

20. Hacer clic en el enlace de `La Dama se Esconde`:



Mostrar	Banda	Año
<input checked="" type="checkbox"/>	La Dama se Esconde	1985
<input checked="" type="checkbox"/>	La Habitación Roja	1994
<input checked="" type="checkbox"/>	Lori Meyers	1998
<input checked="" type="checkbox"/>	Love of Lesbian	1997
<input checked="" type="checkbox"/>	Neuman	1999

Artista: La Dama se Esconde
Año: 1985
Origen: San Sebastián

Observe la URL: `localhost:8080/bands/la-dama-se-esconde`.

Observe el título:

Banda: La Dama se Esconde - Google Chrome

Observe también que el enlace Nueva banda aparece tanto en la vista de bandas, la correspondiente a la ruta `/bands`, como en la vista que muestra la información de la banda seleccionada. Esto se debe a que lo hemos definido en la vista contenedor (*layout*) de la ruta compuesta `/bands`.

Definición de vista para insertar nuevos datos

Ahora, vamos a añadir una vista con la que añadir datos al objeto de bandas y con el que mostrar cómo redireccionar explícitamente de una vista a otra mediante la propiedad `router` del contexto:

1. Ir a la consola.
2. Generar un componente formulario para añadir datos de nuevas bandas:

```
> justo -g react form
```

Respuestas:

- Carpeta donde añadir el componente: `bands`.
- Nombre del componente: `NewBandForm`.
- Añadir `defaultProps`: `N`.
- Añadir `propTypes`: `N`.
- Definir contexto: `N`.
- Usar contexto: `Y`.
- Cancelar la acción predeterminada del formulario: `Y`.
- Controlar el evento `Change`: `Y`.
- Añadir atributo `autoComplete`: `N`.
- Añadir atributo `noValidate`: `N`.

3. Editar el componente `app/components/bands/NewBandForm.jsx`.
4. Importar el objeto de datos:

```
import bands from "../../data/bands";
```

5. Indicar el estado inicial en el constructor del componente:

```
constructor(props) {  
  super(props);  
  
  //initial state  
  this.state = {  
    id: "",  
    name: "",  
    year: "",  
    origin: ""  
  };  
}
```

6. Ir a la consola.
7. Solicitar el código del componente de entrada a través del cual recibir del usuario el `slug` de la nueva banda:

```
> justo -g react snippet input
```

Respuestas:

- Tipo de `<input>`: `text`.
- Nombre de elemento: `id`.
- Tipo de componente: `Controlled`.
- Tipo de componente controlado: `Smart`.
- Valor: `{this.state.id}`.
- Añadir `onChange`: `N`.

- Añadir `autoFocus`: Y.
- Añadir `required`: Y.
- `Placeholder`: slug.

8. Pegar el código en el archivo del componente formulario.

9. Obtener el código para el cuadro de texto name:

```
> justo -g react snippet input
```

Respuestas:

- Tipo de `<input>`: `text`.
- Nombre de elemento: `name`.
- Tipo de componente: `Controlled`.
- Tipo de componente controlado: `Smart`.
- Valor: `{this.state.name}`.
- Añadir `onChange`: N.
- Añadir `autoFocus`: N.
- Añadir `required`: Y.
- `Placeholder`: nombre de banda.

10. Copiar en el formulario debajo del componente id.

11. Repetir el proceso anterior para el componente year:

```
> justo -g react snippet input
```

Respuestas:

- Tipo de `<input>`: `number`.
- Nombre de elemento: `year`.
- Tipo de componente: `Controlled`.
- Tipo de componente controlado: `Smart`.
- Valor: `{this.state.year}`.
- Añadir `onChange`: N.
- Añadir `autoFocus`: N.
- Añadir `required`: N.
- `Placeholder`: año.

12. Copiar el componente en el formulario, debajo de name.

13. Obtener el código para el cuadro de texto origin:

```
> justo -g react snippet input
```

Respuestas:

- Tipo de `<input>`: `text`.
- Nombre de elemento: `origin`.
- Tipo de componente: `Controlled`.
- Tipo de componente controlado: `Smart`.
- Valor: `{this.state.origin}`.
- Añadir `onChange`: N.
- Añadir `autoFocus`: N.
- Añadir `required`: N.

- *Placeholder*: origen.

14. Pegar debajo del componente year.

15. Crear el botón de envío de datos:

> justo -g react snippet input

Respuestas:

- Tipo de `<input>`: submit.
- Texto a mostrar: Guardar.

16. Copiar debajo del componente origen.

Debemos tener algo como:

```
render() {
  return (
    <form onSubmit={this.handleSubmit}
      onChange={this.handleChange}>
      <input type="text"
        name="id"
        value={this.state.id}
        placeholder="slug"
        required
        autoFocus />
      <input type="text"
        name="name"
        value={this.state.name}
        placeholder="nombre de banda"
        required />
      <input type="number"
        name="year"
        value={this.state.year}
        placeholder="año" />
      <input type="text"
        name="origin"
        value={this.state.origin}
        placeholder="origen" />
      <input type="submit" value="Guardar" />
    </form>
  );
}
```

17. Ir al método `handleSubmit()`.

18. Solicitar confirmación de inserción, añadir los datos al objeto bands y redirigir a la ruta `/bands`:

```
handleSubmit(evt) {
  //(1) prevent default action
  evt.preventDefault();

  //(2) save
  if (confirm("¿Está seguro?")) {
    bands[this.state.id] = {
      name: this.state.name,
      year: this.state.year,
      origin: this.state.origin
    };

    this.context.router.push("/bands");
  }
}
```

19. Ir a la propiedad estática `contextTypes`.

20. Indicar que el componente hace uso de la propiedad `router` del contexto:

```
static get contextTypes() {
  return {
    router: React.PropTypes.any
  };
}
```

21. Guardar cambios.

22. Añadir la vista NewBand a bands:

```
> justo -g react view
```

Respuestas:

- Carpeta donde añadirla: bands.
- Nombre de vista: NewBand.
- Tipo de vista: **Immutable**.

23. Añadir la ruta `__new__` a bands:

```
> justo -g react route
```

Respuestas:

- Archivo de rutas: bands.
- Path: `__new__`.
- Vista asociada: NewBand.

24. Editar el archivo `app/routes/bands.jsx`.

25. Subir la ruta `__new__` encima de la ruta `:id`.

El orden en el que aparecen las rutas es importante.

26. Guardar cambios.

27. Ir al archivo `app/views/bands/NewBands.jsx`.

28. Importar el componente NewBandForm:

```
import NewBandForm from "../../components/bands/NewBandForm";
```

29. Ir al método `componentDidMount()` y fijar como título Nueva banda:

```
componentDidMount() {  
  document.title = "Nueva banda";  
}
```

30. Ir al método `render()` y reproducir el componente NewBandForm:

```
render() {  
  return (  
    <NewBandForm />  
  );  
}
```

31. Guardar cambios.

32. Reconstruir la aplicación y desplegarla en el servidor web:

```
> justo build deploy
```

33. Abrir el navegador e ir a `localhost:8080`.

34. Hacer clic en Bandas.

35. Hacer clic en Nueva banda:











36. Rellenar los datos de una nueva banda:



37. Hacer clic en Guardar y aceptar la confirmación.

Tras la inserción de datos, el controlador del evento `Submit` remitirá explícitamente a `/bands`:

	Bandas	
	Nueva banda	
Mostrar	Banda	Año
	La Dama se Esconde	1985
	La Habitación Roja	1994
	Lori Meyers	1998
	Love of Lesbian	1997
	Neuman	1999
	Broken Bells	2009

Enlace activo

Ahora, vamos a configurar el estilo de los enlaces activos, recordemos, aquellos que están relacionados con la vista actual:

1. Editar el archivo `app/views/App.jsx`.
2. Ir al método `render()` y configurar el enlace al listado de bandas para que aparezca en naranja cuando sea el activo:

```
render() {
  return (
    <div>
      <Link to="/bands" activeStyle={{color: "orange"}}>
        <span className="fa fa-list" /> Bandas
      </Link>
      {this.props.children}
    </div>
  );
}
```

3. Guardar cambios.
4. Editar el archivo `app/views/bands/Layout.jsx`.
5. Ir al método `render()` y configurar el enlace para que aparezca en naranja cuando sea el activo:

```
render() {
  return (
    <div>
      <Link to="/bands/__new__" activeStyle={{color: "orange"}}>
        <span className="fa fa-plus" /> Nueva banda
      </Link>
      {this.props.children}
    </div>
  );
}
```

6. Guardar cambios.
7. Ir a la consola.
8. Reconstruir la aplicación y desplegarla en el servidor web:


```
> just build deploy
```
9. Abrir el navegador e ir a `localhost::8080`.

Observe el color del enlace Bandas. Ahora, no corresponde a la vista activa.
10. Hacer clic en Bandas.

Bandas		
+ Nueva banda		
Mostrar	Banda	Año
	La Dama se Esconde	1985
	La Habitación Roja	1994
	Lori Meyers	1998
	Love of Lesbian	1997
	Neuman	1999

Observe que ha cambiado el color del enlace Bandas.

- Hacer clic en Nueva banda:

Bandas	+ Nueva banda			
<input type="text" value="slug"/>	<input type="text" value="nombre de banda"/>	<input type="text" value="año"/>	<input type="text" value="origen"/>	<input type="button" value="Guardar"/>

Observe que ha cambiado el color del enlace Nueva banda. Pero ¿por qué aparece también Bandas como activo? **React Router** considera como activo un enlace de una ruta compuesta cuando nos encontramos en la propia ruta o en una de sus subrutas.

- Hacer clic de nuevo en Bandas y observar que el enlace Nueva banda vuelve a cambiar de color, deja de estar activo.
- Editar el archivo `app/views/App.jsx`.
- Ir al método `render()` y configurar el enlace activo para que sólo se muestre como tal cuando estemos en la propia ruta, no así cuando estemos dentro de una de sus subrutas:

```
render() {
  return (
    <div>
      <Link to="/bands" activeStyle={{color: "orange"}} onlyActiveOnIndex>
        <span className="fa fa-list" /> Bandas
      </Link>
      {this.props.children}
    </div>
  );
}
```

Recordemos que para conseguir nuestro objetivo, las rutas compuestas deben indicar la propiedad `onlyActiveOnIndex`.

- Guardar cambios.
- Ir a la consola.
- Reconstruir la aplicación y desplegarla en el servidor web:


```
> just build deploy
```
- Abrir el navegador e ir a `localhost::8080`.
- Hacer clic en el enlace Bandas.

Observe que cambia de color.
- Hacer clic en Nueva banda.

Observe que ahora Bandas ha dejado de ser activa.