ROUTING & API

ROUTING & API

- **Que aprenderemos en este apartado?**

1. **Front-end routing**
2. **React-Router**
3. **Promesas**
4. **Obteniendo datos de una API**

REACT-ROUTER

Para crear aplicaciones en React que estén compuestas de varias pantallas necesitamos un enrutador.

Un enrutador nos permite relacionar URL's con componentes.

React no incluye un enrutador pero podemos utilizar una librería llamada react-router.

El primer paso para utilizar react-router es instalar la librería.

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

REACT-ROUTER

Con esta librería podemos lograr lo siguiente:

- Establecer rutas en nuestra aplicación (Ejem. Home, Inicio de sesión, etc.)
- Crear páginas públicas y privadas.
- Realizar redirecciones a otras páginas según ciertas condiciones
- Acceder al historial de navegación
- Crear rutas para páginas 404 (no encontradas)

REACT-ROUTER

Los componentes de navegación que utiliza esta librería son:

- **BrowserRouter (Router)**: Este inyecta propiedades a nuestro componente para poder acceder al historial de navegación, realizar redirecciones, etc.
- **Route**: Es el componente que utilizamos para crear nuestras rutas a otros páginas (componentes)
- **Link**: Este es utilizado para los enlaces (igual como etiquetas <a>)

REACT-ROUTER

Configuración de rutas:

```
<Switch>
    <Route exact path="/" component={Home}/>
    <Route exact path="/info" component={Info}/>
    <Route exact path="/contacto" component={Contacto}/>
    <Route component={PageNotFound}/>
</Switch>
```

REACT-ROUTER

Crear los links

```
<li className="nav-container--item s-mr-2">
<Link to="/">Inicio</Link></li>

<li className="nav-container--item s-mr-2">
<Link to="/info">Info</Link></li>

<li className="nav-container--item">
<Link to="/contacto">Contactanos</Link></li>
```

PROMESAS

- Una “promesa” es un objeto que actúa como proxy en los casos en los que no se puede utilizar el verdadero valor porque aún no se conoce (no se ha generado, llegado, …) pero se debe continuar sin esperar a que este disponible (no se puede bloquear la función esperando a su obtención).
- Una “promesa” puede tener los siguientes estados:
  - Pendiente: Aún no se sabe si se podrá o no obtener el resultado.
  - Resuelta: Se ha podido obtener el resultado (Promise.resolve())
  - Rechazada: Ha habido algún tipo de error y no se ha podido obtener el resultado (Promise.reject())
- Los métodos del objeto promesa devuelven al propio objeto para permitir apilar llamadas sucesivas.
- Como objeto, la promesa se puede almacenar en una variable, pasar como parámetro o devolver desde una función, lo que permite aplicar los métodos en distintos puntos del código.

PROMESAS

- El objeto Promise gestiona la creación de la promesa y los cambios de estados de la misma.

```
list() {
  return new Promise((resolve, reject) => {
   this.http.get(this.baseUrl).subscribe(
       data => resolve(data),
       err => reject(err)
       )
  });
 }
```

- Para crear promesas ya concluidas:
  - Promise.reject: Crea una promesa nueva como rechazada cuyo resultado es igual que el argumento pasado.
  - Promise.resolve: Crea una promesa nueva como resuelta cuyo resultado es igual que su argumento.

PROMESAS

- El objeto Promise creado expone los métodos:
  - then(fnResuelta, fnRechazada): Recibe como parámetro la función a ejecutar cuando termine la anterior y, opcionalmente, la función a ejecutar en caso de que falle la anterior.
  - catch(fnError): Recibe como parámetro la función a ejecutar en caso de que falle.

    list().then(calcular, ponError).then(guardar)
- Otras formas de crear e invocar promesas son:
  - Promise.all: Combina dos o más promesas y realiza la devolución solo cuando todas las promesas especificadas se completan o alguna se rechaza.
  - Promise.race: Crea una nueva promesa que resolverá o rechazará con el mismo valor de resultado que la primera promesa que se va resolver o rechazar entre los argumentos pasados.

SERVICIOS REST

- Un **estilo de arquitectura** para desarrollar aplicaciones web distribuidas que se basa en el uso del protocolo HTTP e Hypermedia.
- Definido en el 2000 por Roy Fielding, para no reinventar la rueda, se basa en aprovechar lo que ya estaba definido en el HTTP pero que no se utilizaba.
- El HTTP ya define 8 métodos (algunas veces referidos como "verbos") que indica la acción que desea que se efectúe sobre el recurso identificado:
  - HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT
- El HTTP permite en el encabezado transmitir la información de comportamiento:
  - Accept, Content-type, Response (códigos de estado), Authorization, Cache-control, …

SERVICIOS REST

- **Request**: Método /uri?parámetros
  - GET: Recupera el recurso
    - Todos: Sin parámetros
    - Uno: Con parámetros
  - POST: Crea un nuevo recurso
  - PUT: Edita el recurso
  - DELETE: Elimina el recurso
- **Accept**: Indica al servidor el formato o posibles formatos esperados, utilizando MIME.
- **Content-type**: Indica en que formato está codificado el cuerpo, utilizando MIME
- **Response**: Código de estado con el que el servidor informa del resultado de la petición.

SERVICIOS REST

Request: GET /users
    Response: 200
    content-type:application/json
Request: GET /users/11
    Response: 200
    content-type:application/json
Request: POST /users
    Response: 201
    content-type:application/json
    body
Request: PUT /users/11
    Response: 200
    content-type:application/json
    body
Request: DELETE /users/11
    Response: 204 no content

AXIOS

- Cliente HTTP basado en promesas para el navegador y node.js
- Características:
  - Hacer XMLHttpRequests desde el navegador
  - Hacer solicitudes http desde node.js
  - Transformar los datos de solicitud y respuesta
  - Transformaciones automáticas para datos JSON
  - Soporte de API Promise
  - Interceptores de solicitudes y respuestas
  - Cancelar solicitudes
  - Soporte de cliente para proteger contra XSRF

AXIOS

https://github.com/axios/axios

- Instalación:
  - $ npm install axios
- Desde un CDN:
  - <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
- Importación:
  - import axios from 'axios';
- Peticiones:

```
axios.get('/user')
  .then(function (response) { console.log(response); })
  .catch(function (error) { console.log(error); });
axios.post('/user', { firstName: 'Fred', lastName: 'Flintstone' })
  .then(function (response) { console.log(response); })
  .catch(function (error) { console.log(error); });
```

AXIOS

Métodos

- axios(config)
- axios.request(config)
- axios.get(url[, config])
- axios.delete(url[, config])
- axios.head(url[, config])
- axios.options(url[, config])
- axios.post(url[, data[, config]])
- axios.put(url[, data[, config]])
- axios.patch(url[, data[, config]])

Response Schema

```
{
  data: {},
  status: 200,
  statusText: 'OK',
  headers: {},
  config: {},
  request: {}
}
```