# Npm I eslint vite-plugin-eslint eslint-config-react-app –save-dev

* Command to install EsLint in Vite Applications
* Create a new file in root folder
  + .eslintrc.json
    - Add this line
    - {“extends”:”react-app”}
* In vite.config.js
  + Import eslint from ‘vite-plugin-eslint’
  + In Plugins object, add
    - Eslint()

# A new way of React Routing

* React Router v6.4 introduced a whole new way of defining routes
* W can use the newly defined mechanisms for fetching data inside React Router
* Old Routing still works, but we cannot use the fetch capabilities in that old Way
  + **Only with the new createBrowserRouter**
  + And the new definition of the routes using the Array of Objects

## Steps

* Install React Router
  + Npm I react-router-dom !!!
* Check the createBrowserRouter documentation

# Building the App Layout

* We will build a Global Application Layout and implement it using the **React Router**
  + The NavBar (header)
  + The main
  + The cart overview
  + The Footer
  + These 4 Will be always visible, so are part of the **Application Layout**
* All differences will be in the main section
  + AppLayout is the **main** page in the Router
  + All other pages are **children**  of the main
  + After that, we use **Outlet**  to display based on the URL

A screen shot of a computer program

Description automatically generated

# Fetching Data with React Router – loaders

* A loader is a **function** that fetches data from an API
* We provide that **loader function** to the route we want the data to be used by
* The data will be provided to the **component itself** using a **CustomHook**

## Steps

1. Create the Loader
   1. The convention is to place the **Loader function**  inside the Component where we need the data
   2. Check **Menu Component**

A screenshot of a computer program

Description automatically generated

1. We use this into the **React Router** loader property

A screen shot of a computer

Description automatically generated

1. We get the data into the component using **useLoaderData()** hook, provided by react-router-dom

A screen shot of a computer program

Description automatically generated

# Adding a loading Spinner using react router

* React Router provides a **hook** that tells us in what state the app is right now
* There are **three states**
  + **Idle**
    - No navigation pending
  + **Submitting**
    - A route action is being called due to a form submission
    - POST, PUT, PATCH, DELETE
  + **Loading**
    - The loaders for the next routes are being called to render next page
* The hook is **useNavigation()**
* We use it on the **top-level** component and use **navigation.state** to check the ‘loading’ state

A screen shot of a computer program

Description automatically generated

# Error Handling with Error Elements

* In createBrowserRouter, whenever there is an Error in the router, we can throw an error element instead of one of the elements (\* 🡪 normal router)

## Steps

1. Specify the error element in the **top-level route**
   1. errorElement=<ErrorElement/> component A screenshot of a computer program

      Description automatically generated
2. We get the error message using the **hook 🡪 useRouteError()**  inside the Error component

A computer screen shot of a program code

Description automatically generated

* There are 2 types of errors
  + Error.data 🡺 URL error (page not found)
  + Error.message 🡺 Loader error (API not working)

# Fetching order data from API

## Steps

1. Create a new loader to **fetch** data from the API based on the orderID
   1. The loader receives **by default** some parameters that we can use to get the **orderId** from the URL instead of the useParams hook

A screen shot of a computer code

Description automatically generated

1. Pass the loader to the **path that will need the data from it**

A screen shot of a computer program

Description automatically generated

1. Use the same **useLoaderData**  hook to get the data into the component

A screen shot of a computer code

Description automatically generated

# Sending data to the API (POST) – react-router Actions

* The **Actions** are the writes to the **loader** reads
* With Actions, you can perform data mutations
* We will use **actions** to **create a new order (POST to API**)

## Steps

1. Import and use the **Form** compoment from react-router-dom
   1. Instead of the simple <form> HTML element
2. Specify the **method** this Form will use
   1. We specify **POST**
3. Specify the **action**
   1. The path to where the form will navigate after submit
   2. Not Necessary since React Router will default to the **nearest path**
4. Create the **action**
   1. Similar to the **loader**
   2. The **action** function will get access to the **request**, so we can access some params
   3. To access the **formData:**
      1. await request.request.formData()
      2. Object.fromEntries(formData)
      3. A screen shot of a computer code

         Description automatically generated
   4. To add the **cart**, we are using a hidden input and pass the cart array as a **stringify JSON**
      1. A screen shot of a computer

         Description automatically generated
   5. Back in the **action**, we have to model the data that is sent to the server a bit
      1. Cart 🡪 back to Object type
      2. Priority 🡪 always present
   6. We call the **createOrder()**  from the apiRestaurant
      1. This returns the **newly created order** 🡪 newOrder = await createOrder(order)
   7. We want to immediately **redirect** to the /order/newId to show the user the order details
      1. We programmatically navigate to **/order/newId** using **redirect (a new function)**

A screen shot of a computer code

Description automatically generated

# Form Error Handling

* We can use the **action** to check is the form inputs are correct or not
* We create an **errors Object** where we check and store for all the Errors that might appear in the Form

A screen shot of a computer

Description automatically generated

* If the **Errors Object** has some keys (length > 0), then we actually **return that error object** 
  + Like this, we Stop the creation of a new order if there are errors
* We can access that **Error Object**  inside the **component** using the **useActionData() hook;**

A screen shot of a computer program

Description automatically generated

* We use this to **display the error**  in the Form

