Steps to add react router in the project:-

Step One ➖
Define Routes:-

1) Import createBrowserRouter, RouterProvider from react-router-dom.

```
import { createBrowserRouter, RouterProvider } from 'react-router-dom';

import HomePage from './pages/Home';

const router = createBrowserRouter([
  { path: '/', element: <HomePage /> },
]);

function App() {
  return <RouterProvider router={router} />;
}

export default App;
```

2)
3) We can add more routes as follows:-

```
import { createBrowserRouter, RouterProvider } from 'react-router-dom';

import HomePage from './pages/Home';
import ProductsPage from './pages/Products';

const router = createBrowserRouter([
  { path: '/', element: <HomePage /> },
  { path: '/products', element: <ProductsPage /> }
]);

function App() {
  return <RouterProvider router={router} />;
}

export default App;
```

4)
5) How to add links to new react router?

6)

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

function HomePage() {
  return (
    <>
      <h1>My Home Page</h1>
      <p>
        Go to <Link to="/products">the list of products</Link>.
      </p>
    </>
  );
}

export default HomePage;
```

7) Adding navigation bar ➖We need to add root layout and then add all the children routes inside it.Create a new file called Root.js and add a <Outlet/> tag to render all the components there as shown below.

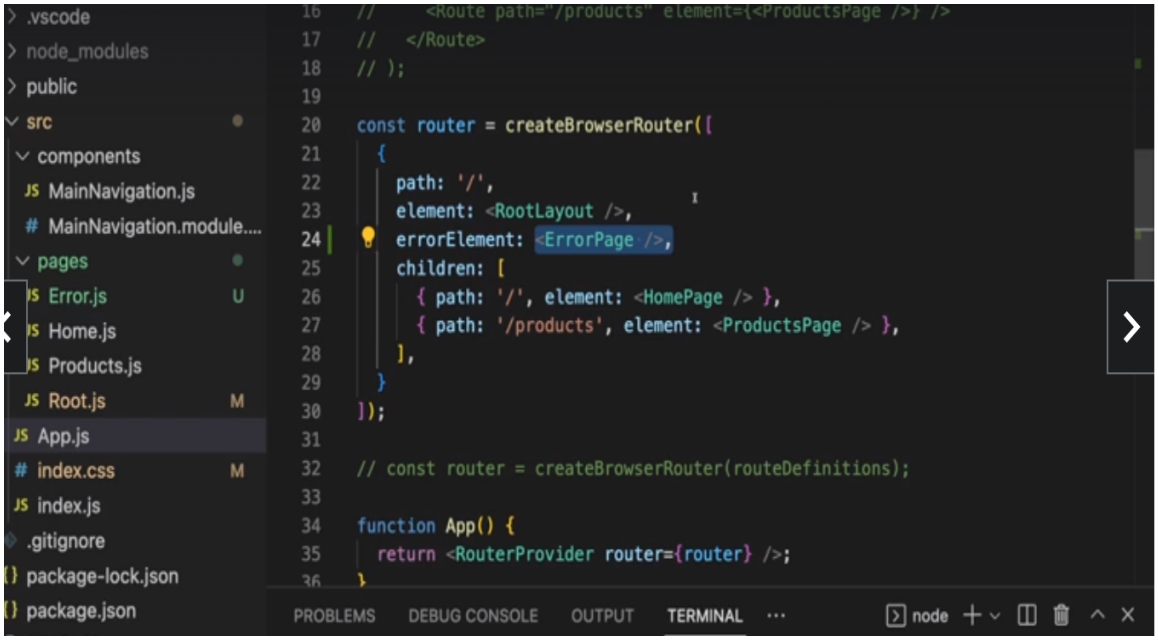
8)

```jsx
import { Outlet } from 'react-router-dom';

function RootLayout() {
  return (
    <>
      <h1>Root Layout</h1>
      <Outlet />
    </>
  );
}

export default RootLayout;
```

9)

```jsx
const router = createBrowserRouter([
  {
    path: '/',
    element: <RootLayout />,
    children: [
      { path: '/', element: <HomePage /> },
      { path: '/products', element: <ProductsPage /> },
    ],
  },
]);
```

10) We can now add navigation to the root layout . We can replace <h1>Root Layout</h1> with the navigation element.
11) How to add error elements? (like 404 page)
12) Create a new error page .

13) 

14) For any unsupported route we can show an error page.
15) Working with navlinks. (Highlighting active link).
16) NavLink can be used as a replacement to Link component for hovering effect.
17) 

18) Since every route starts with “/” so it highlights all the routes starting with “/”. In order to avoid this we need to add end property to the <NavLink/> as shown below.
19) We can show an index route or default route if the parent path is active that can be done as follows.

20)

```
<nav>
  <ul className={classes.list}>
    <li>
      <NavLink
        to="/"
        className={({ isActive }) =>
          isActive ? classes.active : undefined
        }
        style={({isActive}) => ({textAlign: isActive ? '
        end
      >
        Home
      </NavLink>
    </li>
    <li>
      <NavLink
        to="/products"
        className={({ isActive }) =>
          isActive ? classes.active : undefined
```

21) How to navigate programmatically?

22) We can do that by using a special hook called useNavigate();

23)

```
import { Link, useNavigate } from 'react-router-dom';

function HomePage() {
  const navigate = useNavigate();

  function navigateHandler() {
    navigate('/products');
  }

  return (
    <>
      <h1>My Home Page</h1>
      <p>
        Go to <Link to="/products">the list of products</Link>.
      </p>
      <p>
        <button onClick={navigateHandler}>Navigate</button>
      </p>
    </>
  );
```

24) Using Dynamic Routes. (using params):-

25)
```
    { path: '/', element: <HomePage /> },
    { path: '/products', element: <ProductsPage /> },
    { path: '/products/:productId', element: <ProductDetailPage /> }
  ],
}
;
```

26) Same component would be loaded everytime but param would be different.

27) We use useParams hook for fetching the params. As shown below.

28)
```
import { useParams } from 'react-router-dom';

function ProductDetailPage() {
  const params = useParams();

  return (
    <>
      <h1>Product Details!</h1>
      <p>{params.productId}</p>
    </>
  );
}

export default ProductDetailPage;
```

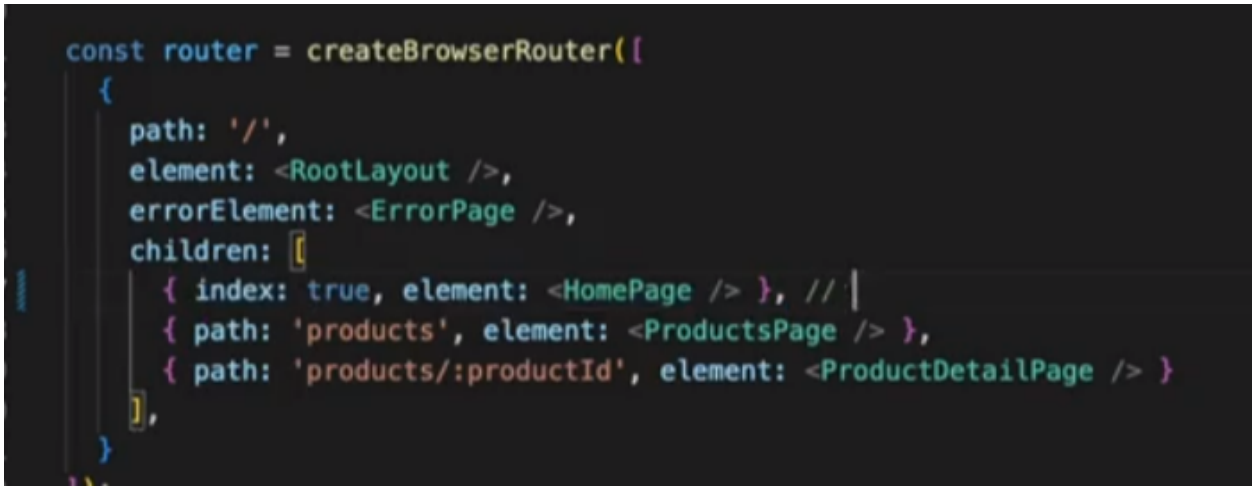
29) How to add dynamic link items?

30)
```
<h1>The Products Page</h1>
<ul>
  {PRODUCTS.map((prod) => (
    <li key={prod.id}>
      <Link to={`/products/${prod.id}`}>{prod.title}</Link>
    </li>
  ))}
</ul>
</>
```

31) What are relative and absolute paths?

32) What are index routes.? Index routes stands for default routes which should be shown when a particular route is considered to be shown as default route.

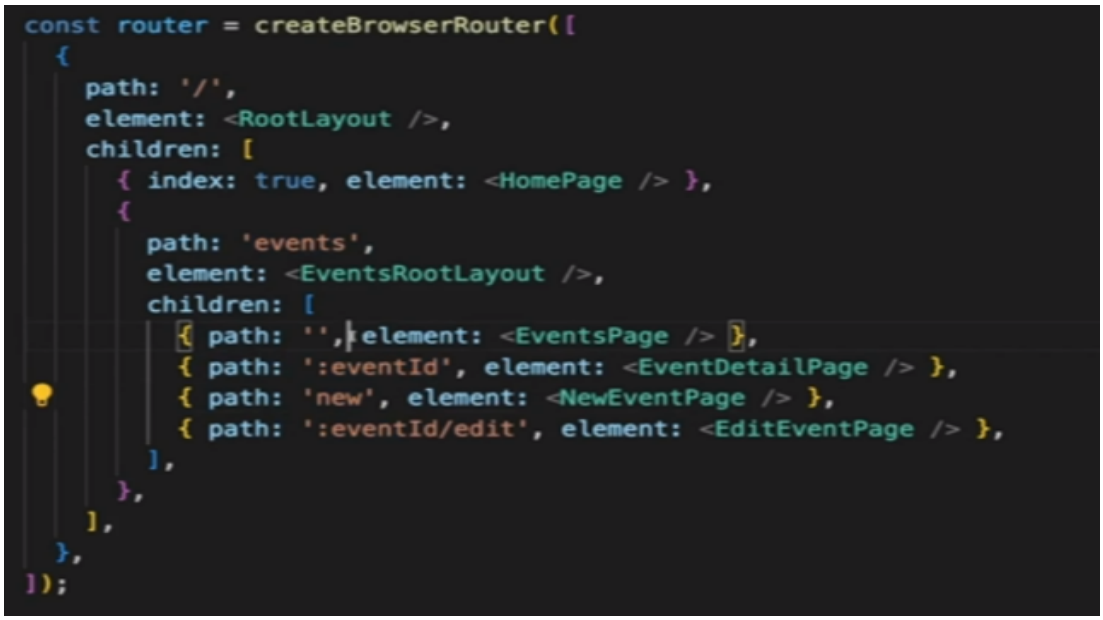
33)



```
const router = createBrowserRouter([
  {
    path: '/',
    element: <RootLayout />,
    errorElement: <ErrorPage />,
    children: [
      { index: true, element: <HomePage /> }, //
      { path: 'products', element: <ProductsPage /> },
      { path: 'products/:productId', element: <ProductDetailPage /> }
    ],
  }
```

Some Other Extremely Important concepts in React.

1)



```
const router = createBrowserRouter([
  {
    path: '/',
    element: <RootLayout />,
    children: [
      { index: true, element: <HomePage /> },
      {
        path: 'events',
        element: <EventsRootLayout />,
        children: [
          { path: '', element: <EventsPage /> },
          { path: ':eventId', element: <EventDetailPage /> },
          { path: 'new', element: <NewEventPage /> },
          { path: ':eventId/edit', element: <EditEventPage /> },
        ],
      },
    ],
  },
]);
```

2) In the above example we are having some nested routes. We can have multiple such nested routes with individual roolLayouts with relative paths.
3) **For NavLink if the previous navLink is still showing highlighted we need to add end property to it.**
4) **How to fetch Data ?**
5) React router helps us in fetching the data first and them loading the component.
6) Step One Add an extra loader property to the route.

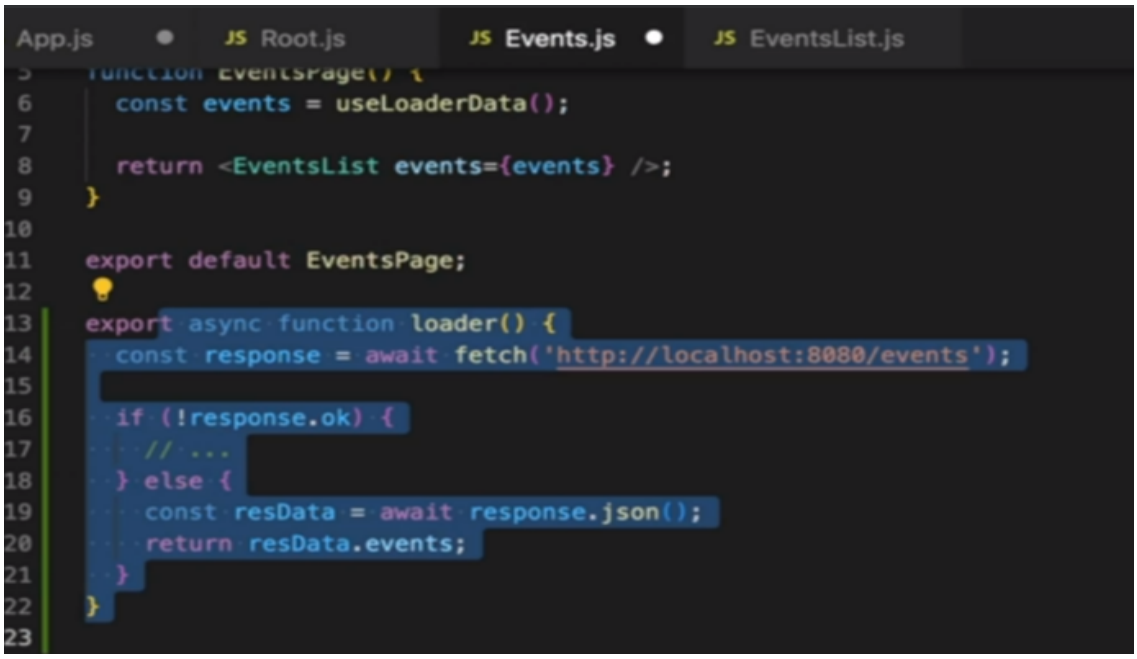
7)
```
element: <EventsRootLayout />,
children: [
  {
    index: true,
    element: <EventsPage />,
    loader: async () => {
      const response = await fetch('http://localhost:8080/events');

      if (!response.ok) {
        // ...
      } else {
        const resData = await response.json();
        return resData.events;
      }
    },
  },
```

8) How to send the data of the loader function to the component.?

9)
```
import { useLoaderData } from 'react-router-dom';

import EventsList from '../components/EventsList';

function EventsPage() {
  const events = useLoaderData();

  return <EventsList events={events} />;
}

export default EventsPage;
```

10) We can directly access loader function values in the component itself as follows:-

11)
```
import { useLoaderData } from 'react-router-dom';

import classes from './EventsList.module.css';

function EventsList() {
  const events = useLoaderData();

  return (
    <div className={classes.events}>
      <h1>All Events</h1>
      <ul className={classes.list}>
        {events.map((event) => (
          <li key={event.id} className={classes.item}>
            <a href="...">
              <img src={event.image} alt={event.title} />
              <div className={classes.content}>
                <h2>{event.title}</h2>
                <time>{event.date}</time>
              </div>
            </a>
```

12) We cannot load data on the higher level in the route like rootlayout. We can access it in the same level or lower level.
13) It not the best practice to add loader function in the route. It is better to add it in a separate function as show below.
14) 

15) Then we can use that loader function just change the loader function to that actual function name by importing that function.
16) When are loader functions executed?
17) Data fetching is initiated as soon as we route to the component. However it waits for some time to finish the data fetching and then load the data.
18) We can use useNavigation() hook to show loader but it would be shown on a different page while the component is getting loaded.
19) We can return any kind of data in a loader. React router package automatically extract data from the loader. So received response is supported by a react special code called new Response();
20) React hooks cannot be used inside loader functions();
21) How can we handle errors in loader functions?
22) We can create an errorPage just like we did in 404 not found page and add it to errorElement in the routes as shown below:-
23) Error elements do bubble up and the closest error element would be fetched to the UI. Else error element present in the root would be shown,

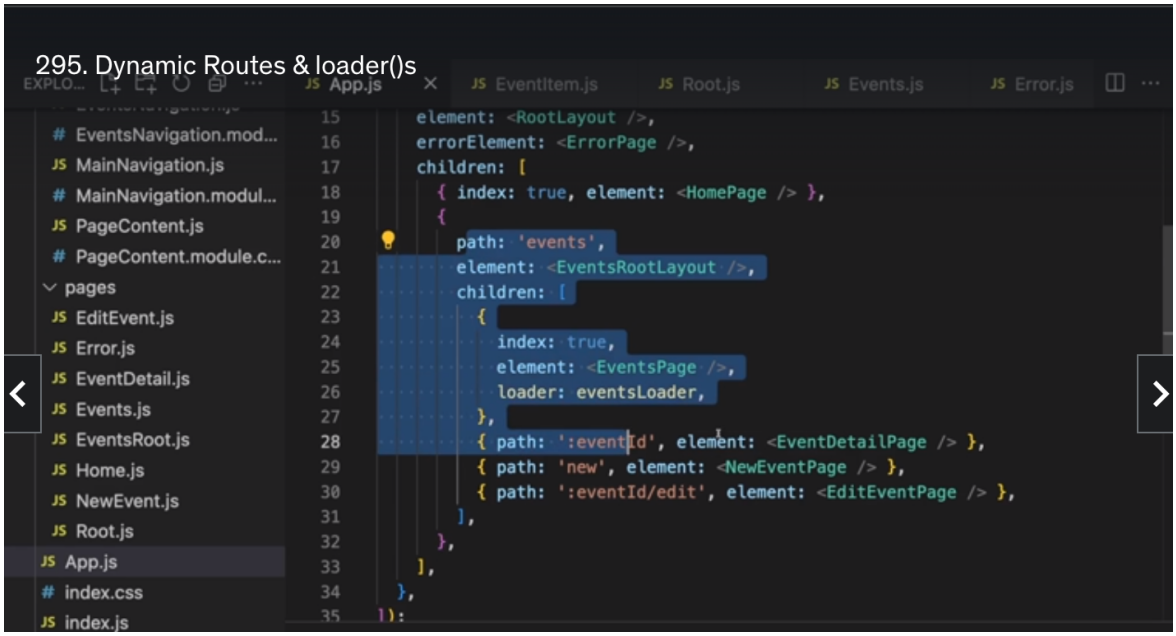
24)
```
{
  path: '/',
  element: <RootLayout />,
  errorElement: <ErrorPage />,
  children: [
    { index: true, element: <HomePage /> },
    {
      path: 'events',
      element: <EventsRootLayout />,
      children: [
        {
          index: true,
          element: <EventsPage />,
          loader: eventsLoader,
        },
        { path: ':eventId', element: <EventDetailPage /> },
        { path: 'new', element: <NewEventPage /> },
        { path: ':eventId/edit', element: <EditEventPage /> },
      ],
    },
```

25) We can throw error in the events page as shown below incase of error.We can modify the loader function in the eventsComponet as follows.
26) We can use json utility function for avoiding issues as shown below:-
27)
```
export async function loader() {
  const response = await fetch('http://localhost:8080/events');

  if (!response.ok) {
    // return { isError: true, message: 'Could not fetch events.' };
    // throw new Response(JSON.stringify({ message: 'Could not fetch events.'
    //   status: 500,
    // });
    throw json(
      { message: 'Could not fetch events.' },
      {
        status: 500,
      }
    );
  } else {
    return response;
  }
}
```

28) With JSON utility we dont type more code also the parsing is automatically.
29) How to work with dynamic routes using loader?
30) Whenever using relative paths we dont have to specify complete path. We can directly specify the routes as mentioned below.

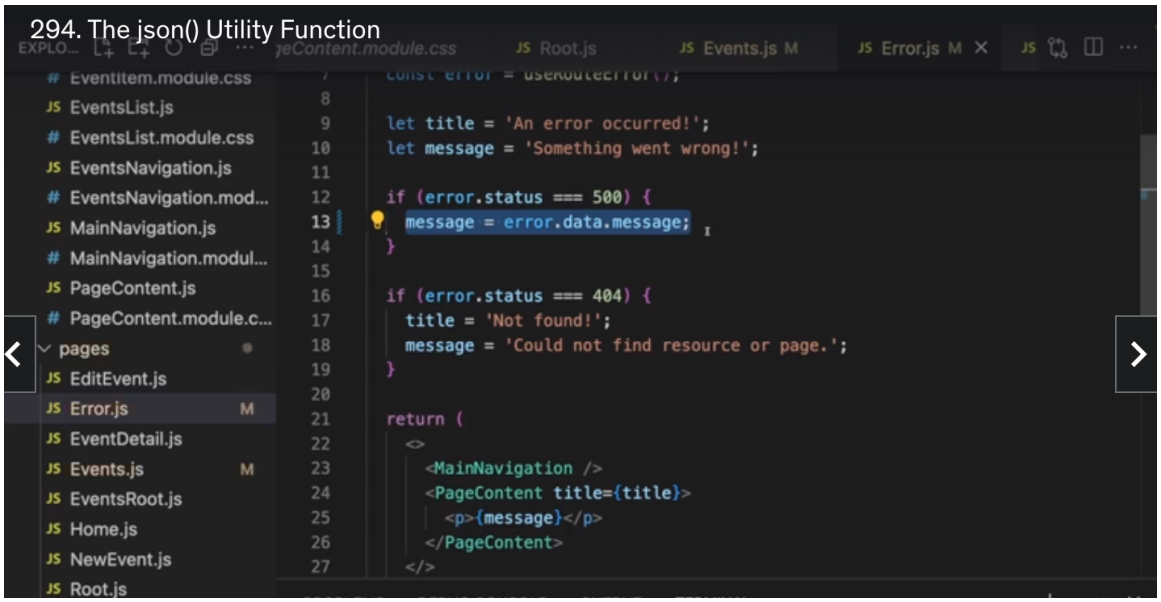
295. Dynamic Routes & loader()s

31)



294. The json() Utility Function

32)



33) After using relative routing , we can directly redirect it using to property.

loader()s

```
Error.js   EventsRoot.js   events.js   EventsList.js

function EventsList({events}) {
  // const events = useLoaderData();

  return (
    <div className={classes.events}>
      <h1>All Events</h1>
      <ul className={classes.list}>
        {events.map((event) => (
          <li key={event.id} className={classes.item}>
            <Link to={event.id}>
              <img src={event.image} alt={event.title} />
              <div className={classes.content}>
                <h2>{event.title}</h2>
                <time>{event.date}</time>
              </div>
            </Link>
          </li>
        ))}
      </ul>
    </div>
```

34)

35) Now whenever user clicks on event list page , user is redirected to event detail page. Event detail page has an EventItem component where we pass event prop as show below. Now I can use loader function to pass paramId but we cannot access react hooks inside loader functions. React loader function accepts two params. Request and params which allow us to access route params as show below.

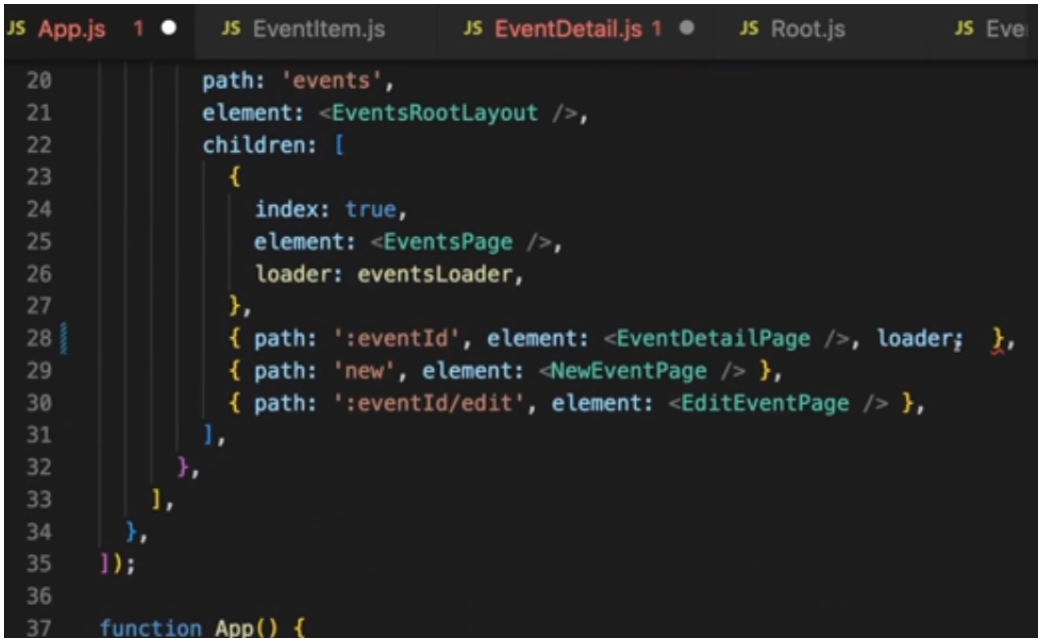
& loader()s

```
App.js   EventItem.js   EventDetail.js 1   Root.js   Eve

    return (
      <EventItem event={} />
    );
  }

  export default EventDetailPage;

  export async function loader({request, params}) {
    const id = params.eventId;

    const response = await fetch('http://localhost:8080/events/' + id);

    if (!response.ok) {
      throw json({message: 'Could not fetch details for selected event.'}, {
        status: 500
      })
    } else {
      return response;
    }
  }
```

36)

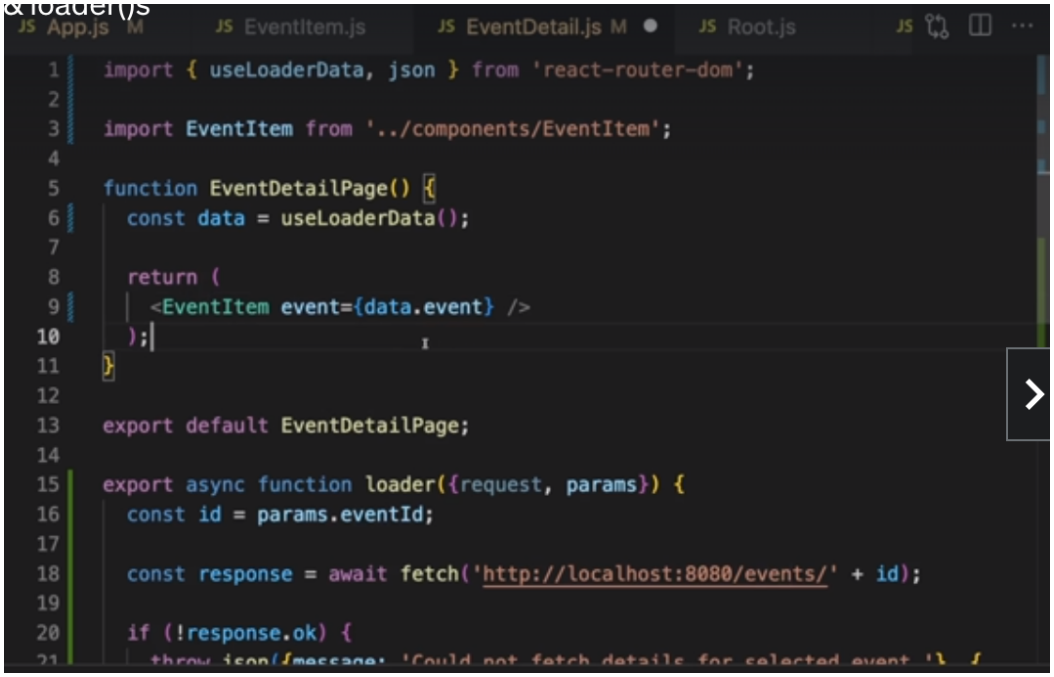
37) Now we can register the loader function in the route as shown below.

38) We can write import loader as eventDetailLoader from the file location and associate it to the route as shown below.

39) 

```
JS App.js 1 ●   JS EventItem.js   JS EventDetail.js 1 ●   JS Root.js   JS Eve
20            path: 'events',
21            element: <EventsRootLayout />,
22            children: [
23              {
24                index: true,
25                element: <EventsPage />,
26                loader: eventsLoader,
27              },
28              { path: ':eventId', element: <EventDetailPage />, loader;  },
29              { path: 'new', element: <NewEventPage /> },
30              { path: ':eventId/edit', element: <EditEventPage /> },
31            ],
32          },
33        ],
34      },
35    ]);
36
37    function App() {
```

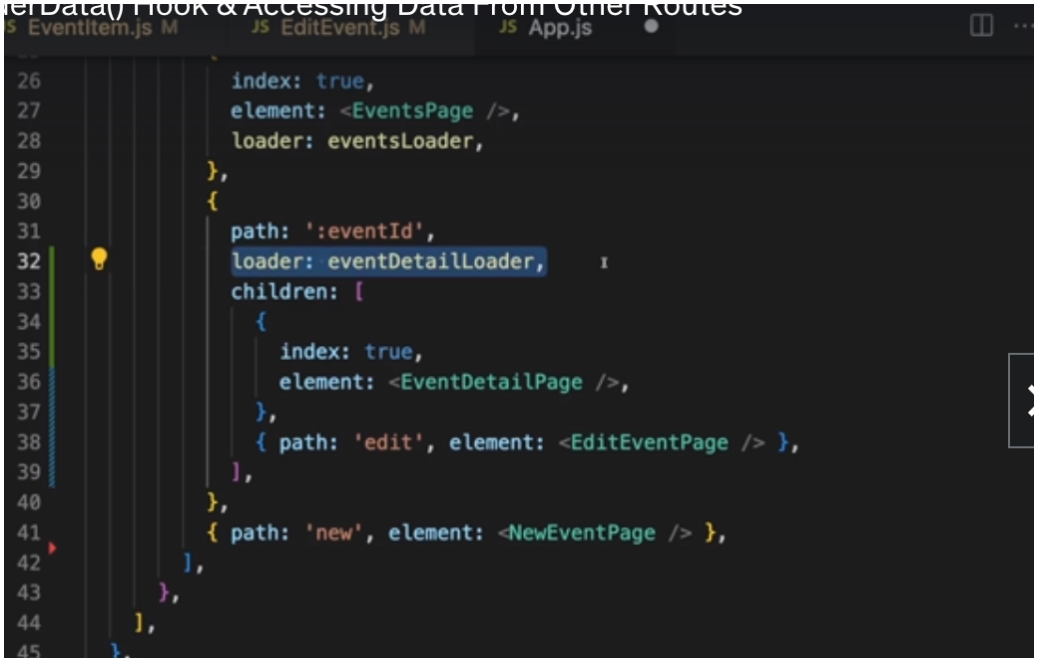
40) Now we can use useLoaderData hook in eventDetails page to get hold of that data. & loader()s

41) 

```
JS App.js M   JS EventItem.js   JS EventDetail.js M ●   JS Root.js   JS
1   import { useLoaderData, json } from 'react-router-dom';
2
3   import EventItem from '../components/EventItem';
4
5   function EventDetailPage() {
6     const data = useLoaderData();
7
8     return (
9       <EventItem event={data.event} />
10    );
11  }
12
13  export default EventDetailPage;
14
15  export async function loader({request, params}) {
16    const id = params.eventId;
17
18    const response = await fetch('http://localhost:8080/events/' + id);
19
20    if (!response.ok) {
21      throw json({message: 'Could not fetch details for selected event'}, {
```

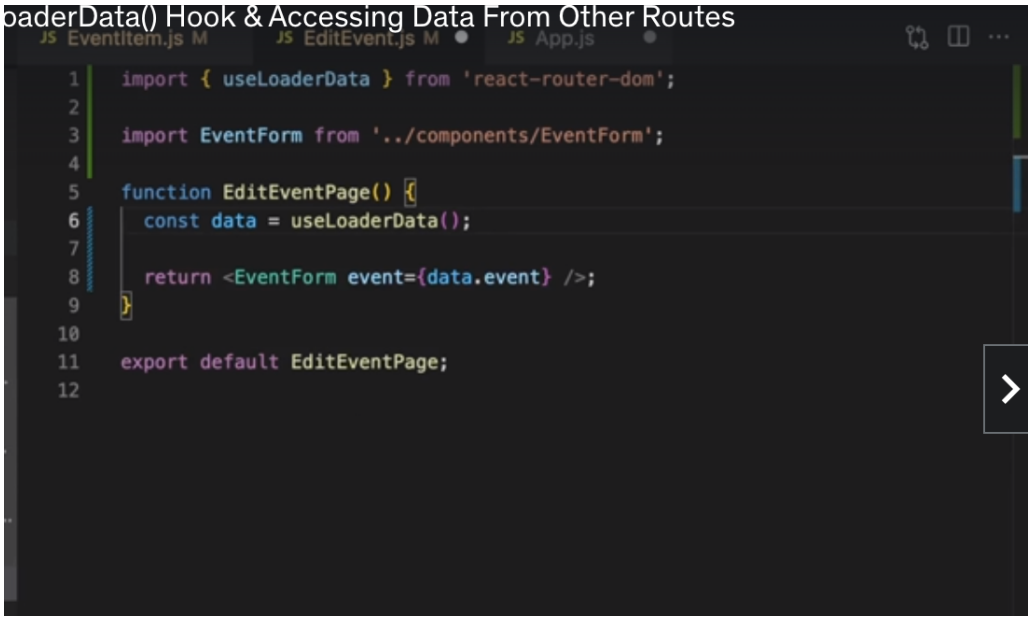
42) useRouteLoaderData() hook and accessing data from other routes?

43) There is a scenario where I want to load the same data when I viewing a particular page and editing a particular page. So in this case with relative routes we can nest the two routes together as shown below.

44) 

45) The above approach allows us to reuse the data and the logic for view as well as edit page.

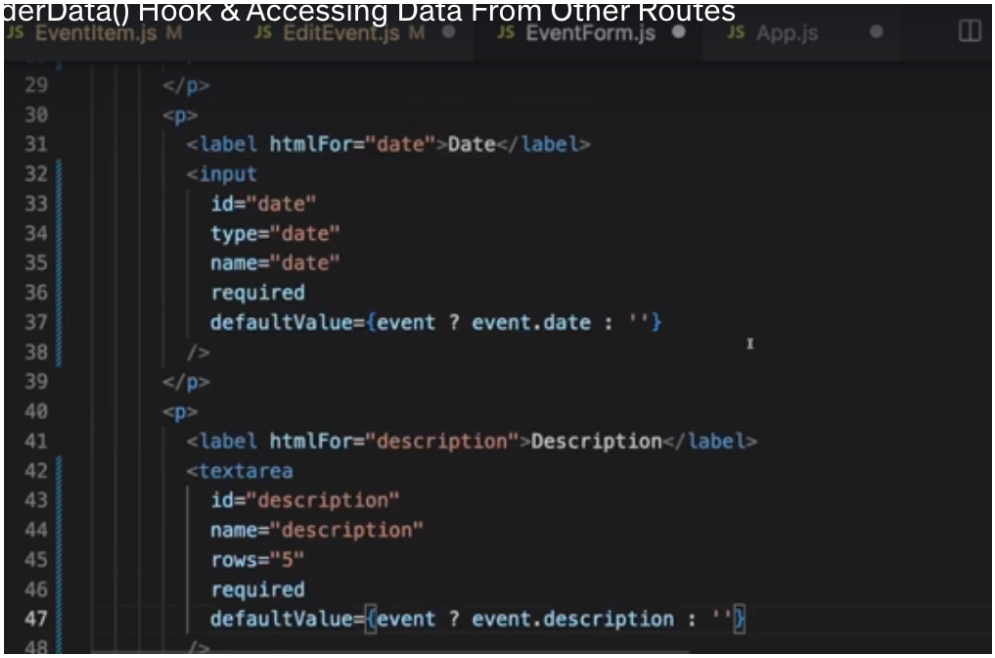
46) Now we can access the date in the edit page using useLoaderDate() hook as shown below

47) 

48) We can then use defaultValue props in the input html tags in the eventForm.js component.

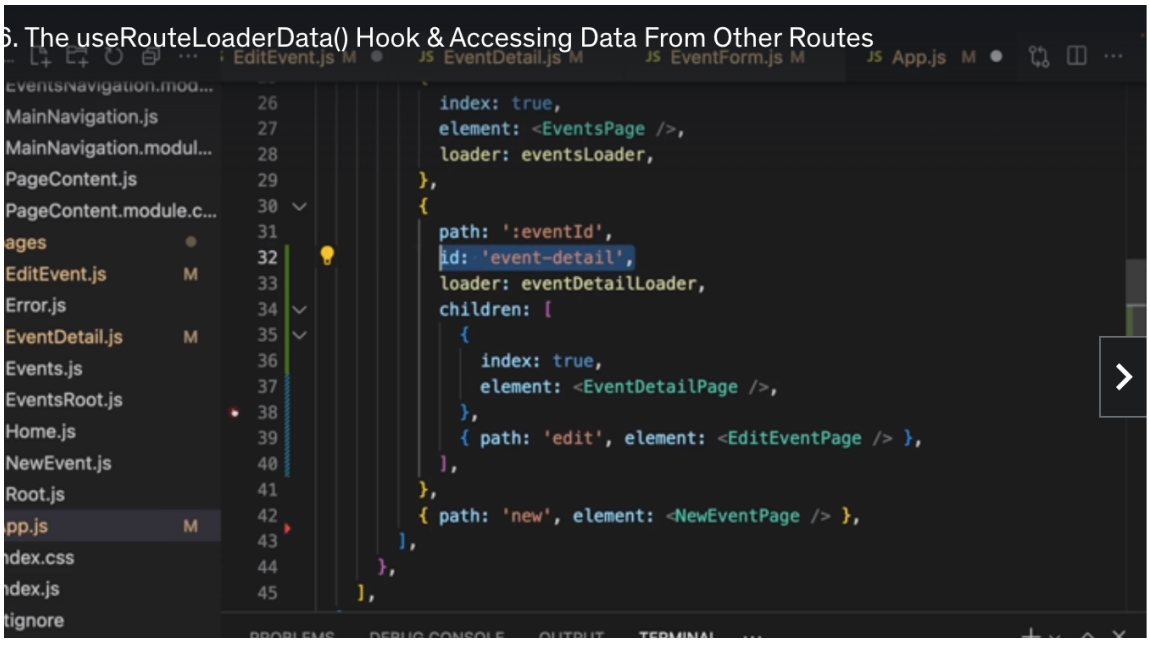
49)

derData() Hook & Accessing Data From Other Routes



50)

51) However there is an issue with the above approach. If we want to use the same loader function with route children we need to map them by id



52)

53) Now we can use the same id for all the components using routeLoader() hook.

54)

```
EditEvent.js M   EventDetail.js M   EventForm.js M   App.js M

1  import { useRouteLoaderData } from 'react-router-dom';
2
3  import EventForm from '../components/EventForm';
4
5  function EditEventPage() {
6    const data = useRouteLoaderData('event-detail');
7
8    return <EventForm event={data.event} />;
9  }
10
11 export default EditEventPage;
12
```

55) Data submission using react router? How can we submit data with react router?
56) We can add something called actions to send data using react router 🙂
57) Steps to send form data using react router DOM:-
58) Before sending any values of a form to an API , we have to make sure that our form HTML tags have name attribute.
59) Import <Form/> component from react-router-dom;
60) This form tag help react router to send to the action function.
61)

```
EventForm.js   NewEvent.js 1, M   Events.js   App.js 1

4
5  function EventForm({ method, event }) {
6    function cancelHandler() {
7      // ...
8    }
9
10   return (
11     <Form method='post' className={classes.form}>
12       <p>
13         <label htmlFor="title">Title</label>
14         <input
15           id="title"
16           type="text"
17           name="title"
18           required
19           defaultValue={event ? event.title : ''}
20         />
21       </p>
22       <p>
23         <label htmlFor="image">Image</label>

PROBLEMS 2   DEBUG CONSOLE   OUTPUT   TERMINAL
```

62) Add method to the Form tag. We can have any type of methods like post, delete, put etc.
63) Now we need to modify our actions function as shown below:-

() Functions

64)

```
export async function action({request, params}) {
  const data = await request.formData();

  const eventData = {
    title: data.get('title'),
    image: data.get('image'),
    date: data.get('date'),
    description: data.get('description'),
  }

  fetch('http://localhost:8080/events', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(eventData)
  });
}
```

65) We can modify that fetch into await and get the response and play with the response for validation of data.
66) Now we can map that action function to the route defined in the app.js as shown below.
67) We can import actions as newEventAction from ‘./pages/NewEvent’:
68) Map the action in the route as show below:-

69)

```
        loader: eventDetailLoader,
        children: [
          {
            index: true,
            element: <EventDetailPage />,
          },
          { path: 'edit', element: <EditEventPage /> },
        ],
      },
      { path: 'new', element: <NewEventPage />, action: newEventAction },
    ],
  },
],
},
);
```

70) After the POST request was successful we might want to redirect the user to a new page.
71) That redirection can be done using a utility called redirect() as shown below:-

72)
```
  });

  if (!response.ok) {
    throw json({ message: 'Could not save event.' }, { status: 500 });
  }

  return redirect('/events');
}
```

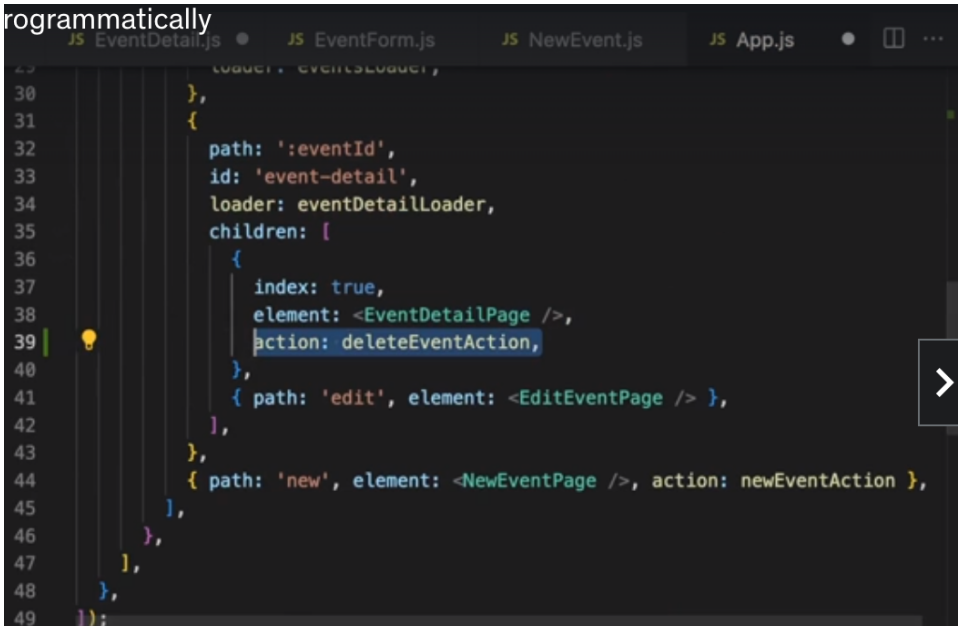
73) How to submit data programmatically?
74) Different way of triggering action function can be via paths.
75) An example to programmatically perform delete operation.
76) Lets say we have an eventDetail component and inside the eventDetail component we have an eventItem component where we have a delete button to delete an event.
77) Lets say I have a prompt function called starteDeletehandler in EventItem.js where I want to delete an item prompt would be shown.
78) I can now add a delete action function inside EventDetail.js function to delete an item as shown below:-

ogrammatically

79)
```
JS EventDetail.js ●   JS EventForm.js   JS NewEvent.js   JS App.js 1 ●

29
30  export async function action({ params }) {
31    const eventId = params.eventId;
32    const response = await fetch('http://localhost:8080/events/' + eventId);
33
34    if (!response.ok) {
35      throw json(
36        { message: 'Could not delete event.' },
37        {
38          status: 500,
39        }
40      );
41    }
42    return redirect('/events');
43  }
44
```

80) Now we can import this deleteEventAction as shown below:-

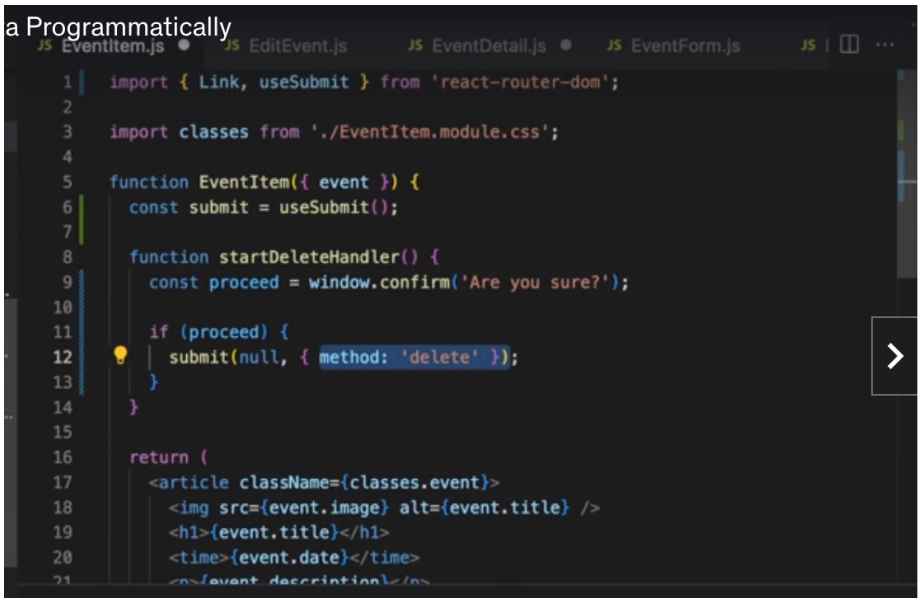
rogrammatically

81) 

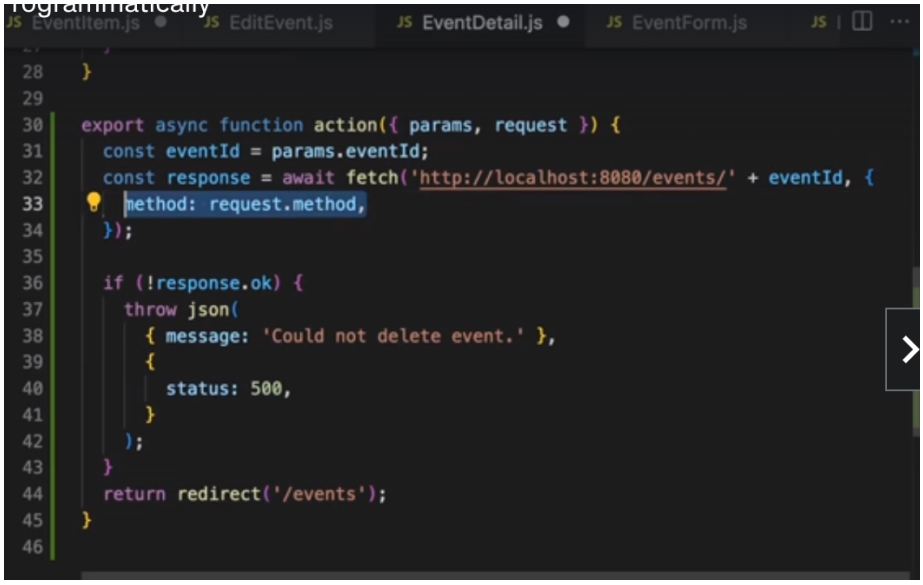
82) How can we now call this in EventItem component which is the child of the parent EventDetail component?

83) We can now use useSubmit hook in this case as shown below:-

a Programmatically

84) 

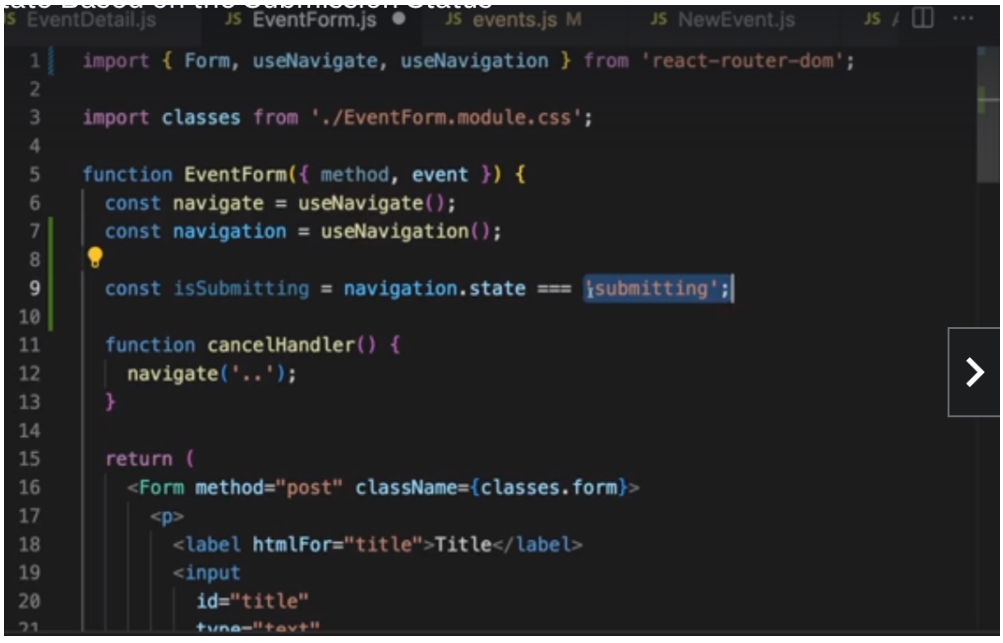
rogrammatically

85) 

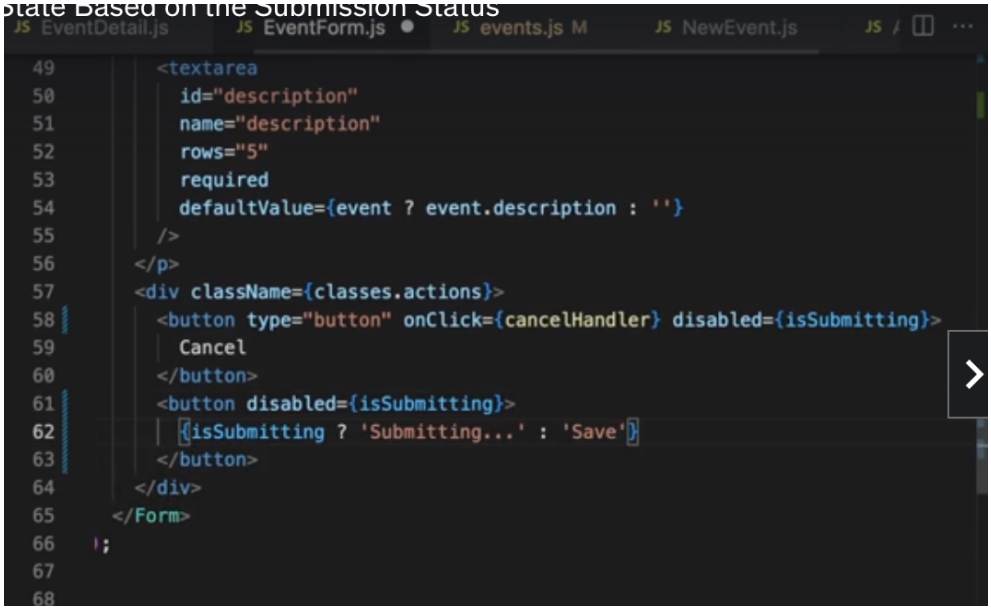
86) As shown above we can perform a delete operation on the method.

87) How to update UI State based on submission status?

88) We can use useNavigation() hook to check the status of route transition for updating the Ui accordingly.

89) 

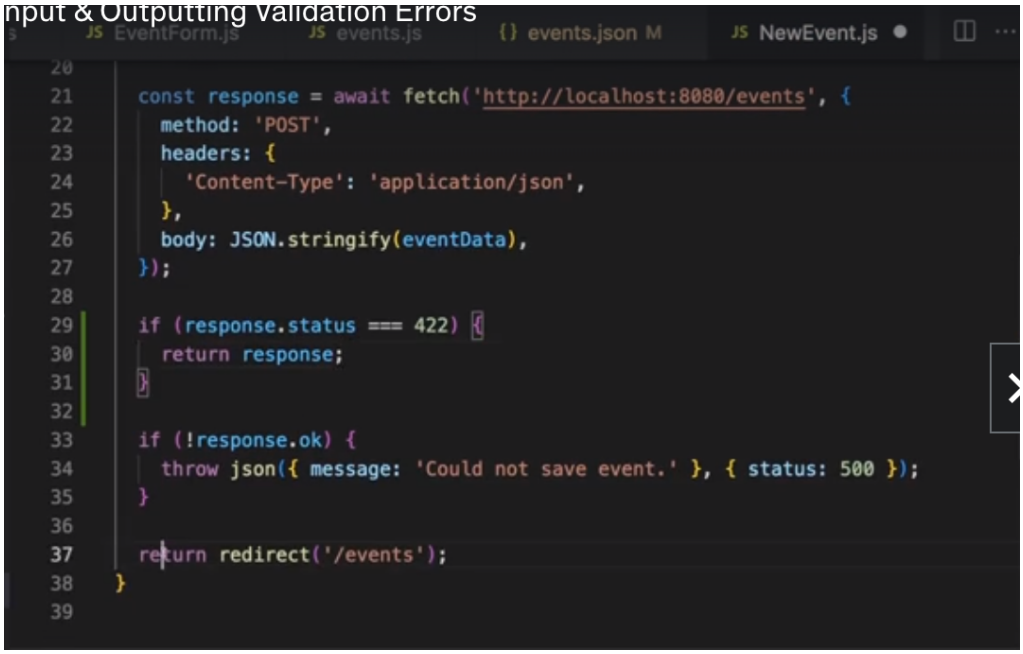
State Based on the Submission Status

90) 

91) How to validate userInputs and show validation Errors ?

92) Lets say I am setting validation status code as 422 in my backend and I want to throw an error in the front end incase user disabled the UI validation and tried to access the form directly.

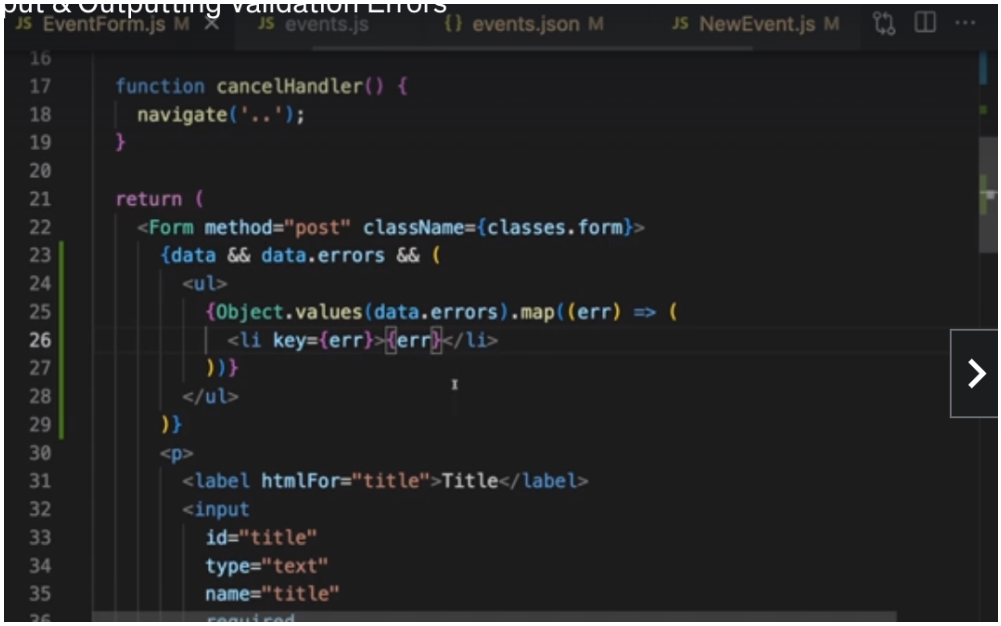
Input & Outputting Validation Errors

93) 

94) Just like we can use return responses in loaders , we can also return responses from actions and use that response data in the component. For accessing the data inside the child component, in this case the EventForm component we can use useActionData () hook as shown below:-

95) So in EventForm.js write const data = useActionData() to access the response data from the delete Event form. We can access the error data as shown below:-

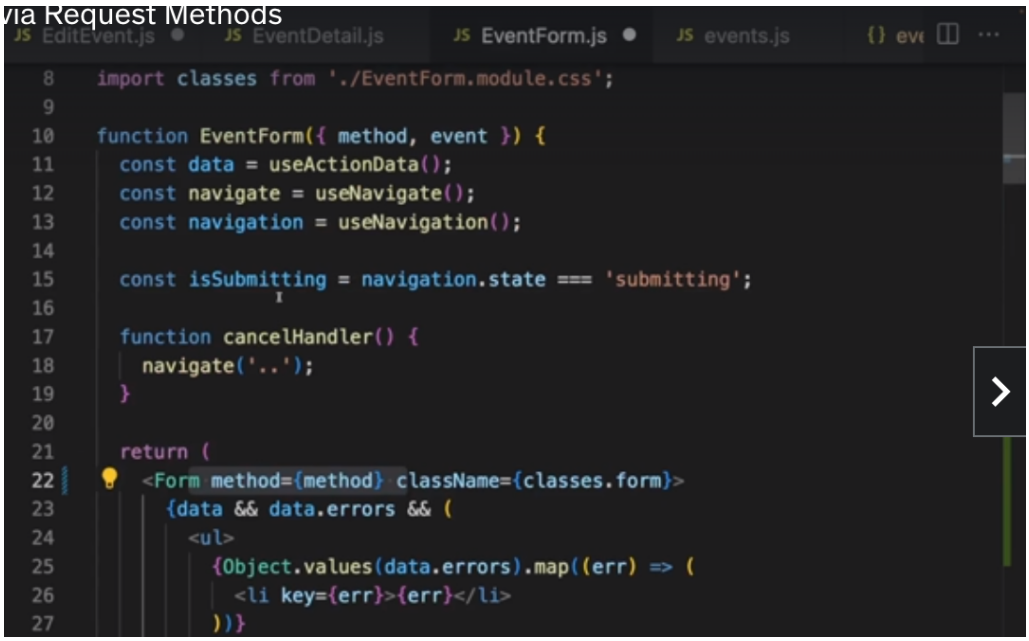
ut & Outputting Validation Errors

96) 

97) Adding edit functionality to EventForm.js. We can reuse actions and action methods whenever we are sending a similar data .

98) We can reuse the same action function by adding method names. We can pass property called method to EventForm component and then pass that method to <Form method={method} className = {classes.form}>.

via Request Methods

99) 

100) This method can be used to extract in the action function as shown below:-

Via Request Methods

```
EventForm.js    events.js    events.json M    NewEvent.js

export default EventForm;

export async function action({ request, params }) {
  const method = request.method;
  const data = await request.formData();

  const eventData = {
    title: data.get('title'),
    image: data.get('image'),
    date: data.get('date'),
    description: data.get('description'),
  };

  let url = 'http://localhost:8080/events';

  if (method === 'patch') {
    const eventId = params.eventId;
    url = 'http://localhost:8080/events/' + eventId;
  }
```

101)

102) PATH should be in capital letters.

103) How to write action functions for components which are common to all the routes?

104) In order to solve this problem we useFetcher() hook. Usually with <Form/> user is automatically redirected to newLetter component and we dont want that behaviour . We want the user to not be redirected to newLetter component.

105) useFetcher() hook is used whenever we dont want to transition to a new route and submit the form details.

nes Work with useFetcher()

```
NewsletterSignup.js M    Newsletter.js

function NewsletterSignup() {
  const fetcher = useFetcher();
  const { data, state } = fetcher;

  useEffect(() => {
    if (state === 'idle' && data && data.message) {
      window.alert(data.message);
    }
  }, [data, state]);

  return (
    <fetcher.Form
      method="post"
      action="/newsletter"
      className={classes.newsletter}
    >
      <input
        type="email"
        placeholder="Sign up for newsletter..."
        aria-label="Sign up for newsletter"
```

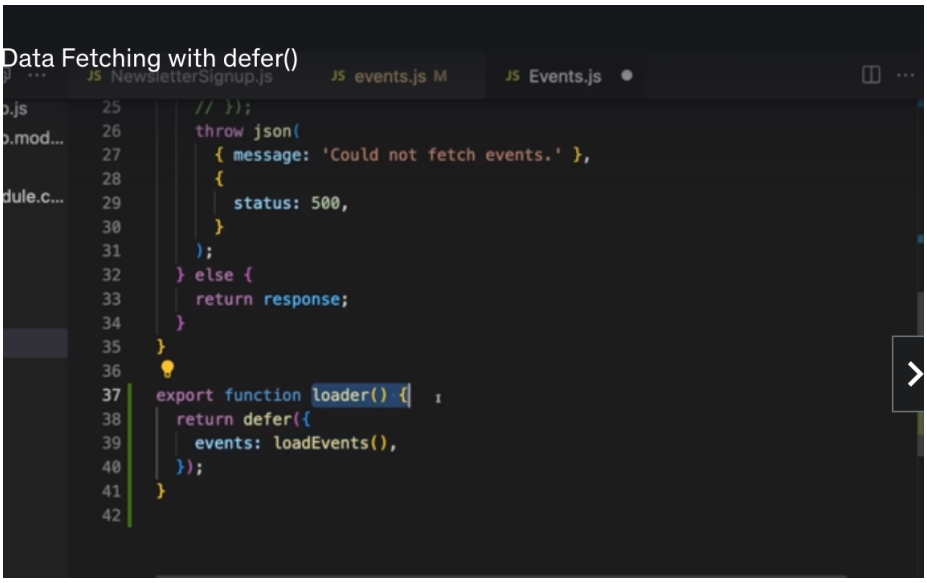
106)

107) Defer() hook..->

108) While we are routed to a new page, we might want to show some components which are independent of data being loaded from the backend API. in this case we can use defer() hook as shown below.

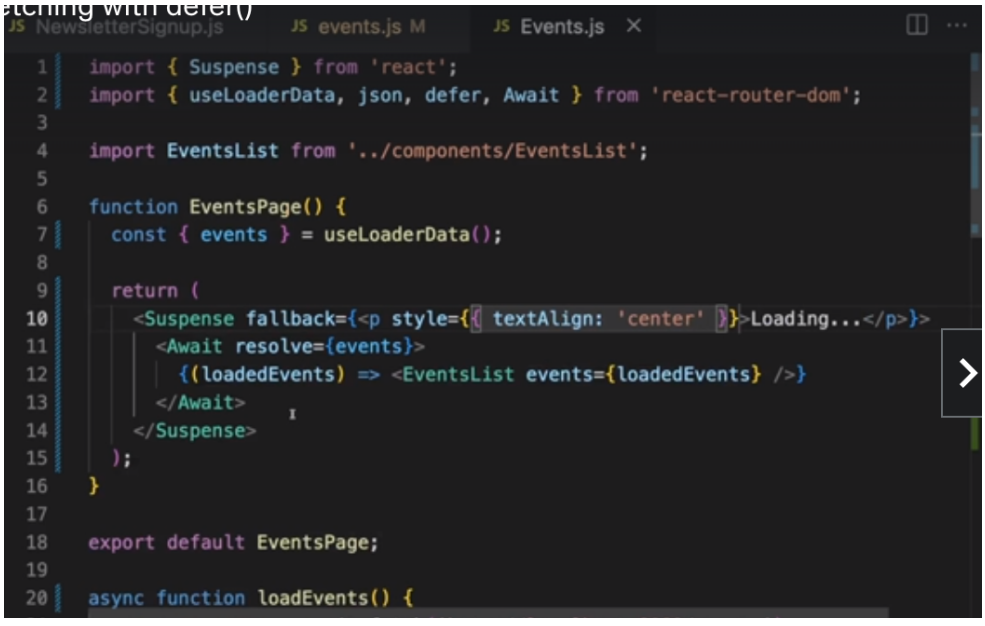
109) Steps to use defer() ;-

110) Step 1:- Create a new function to move all the loader code there.



111) Step 2:-

112) Step3:- add Await, Suspense components as shown below to show data loading only for certain part of the component.



113)

Fetching with defer()

JS NewsletterSignup.js JS events.js M JS Events.js 1, M

```
17
18  export default EventsPage;
19
20  async function loadEvents() {
21    const response = await fetch('http://localhost:8080/events');
22
23    if (!response.ok) {
24      // return { isError: true, message: 'Could not fetch events.' };
25      // throw new Response(JSON.stringify({ message: 'Could not fetch eve
26      //   status: 500,
27      // });
28      throw json(
29        { message: 'Could not fetch events.' },
30        {
31          status: 500,
32        }
33      );
34    } else {
35      const resData = await response.json();
36      return resData.events;
```

114)

115) Step 4 add the above code for updating the load data logic.

116) How does defer shine when we have multiple http calls?

117) Conder a scenario where we want to load data simultaneously and controll which data needs to be fetched or awaited.

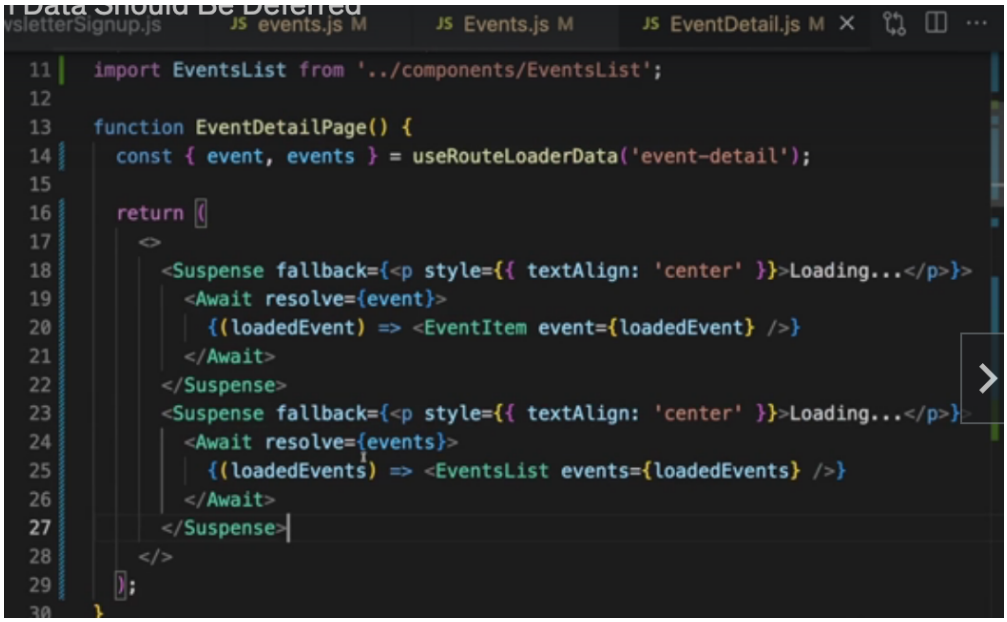
118) Consider the following example where we want to output event data as well as event all events data. This can be done in the following way:-

119) Step one create two different helper functions that load data

```
export async function loader({ request, params }) {
  const id = params.eventId;

  return defer({
    event: loadEvent(id),
    events: loadEvents
  })
}
```

120)

121) Both the components will have their own Await and suspense events as shown below:-


Data Should Be Deferred
vsletterSignup.js
events.js M
Events.js M
EventDetail.js M
import EventsList from '../components/EventsList';
function EventDetailPage() {
const { event, events } = useRouteLoaderData('event-detail');
return (
<>
<Suspense fallback={<p style={{ textAlign: 'center' }}>Loading...</p>}>
<Await resolve={event}>
{(loadedEvent) => <EventItem event={loadedEvent} />}
</Await>
</Suspense>
<Suspense fallback={<p style={{ textAlign: 'center' }}>Loading...</p>}>
<Await resolve={events}>
{(loadedEvents) => <EventsList events={loadedEvents} />}
</Await>
</Suspense>
</>
);
}

122)

123)