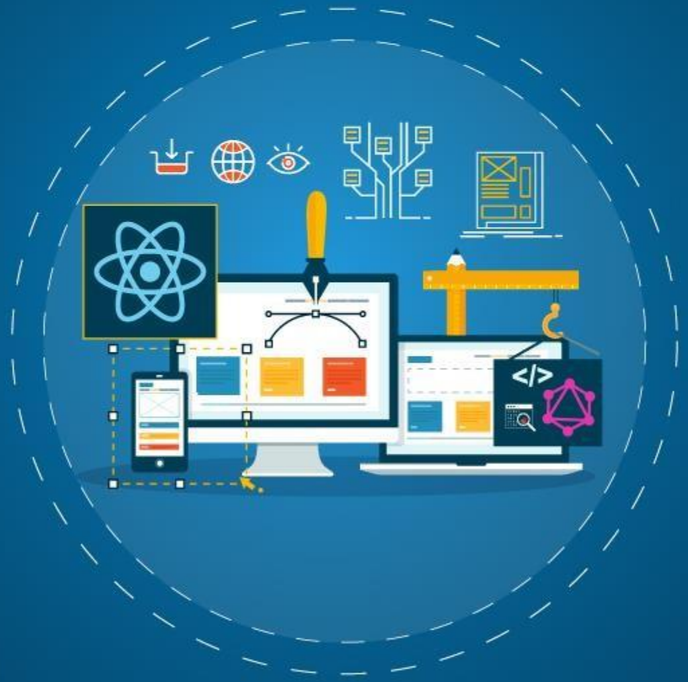
React With Redux Certification Training

COURSE OUTLINE
MODULE 03

1. Introduction to Web Development and React
2. Components and Styling the Application Layout
3. **Handling Navigation with Routes**
4. React State Management using Redux
5. Asynchronous Programming with Saga Middleware



6. React Hooks
7. Fetching Data using GraphQL
8. React Application Testing and Deployment
9. Introduction to React Native
10. Building React Native Applications with APIs

Topics

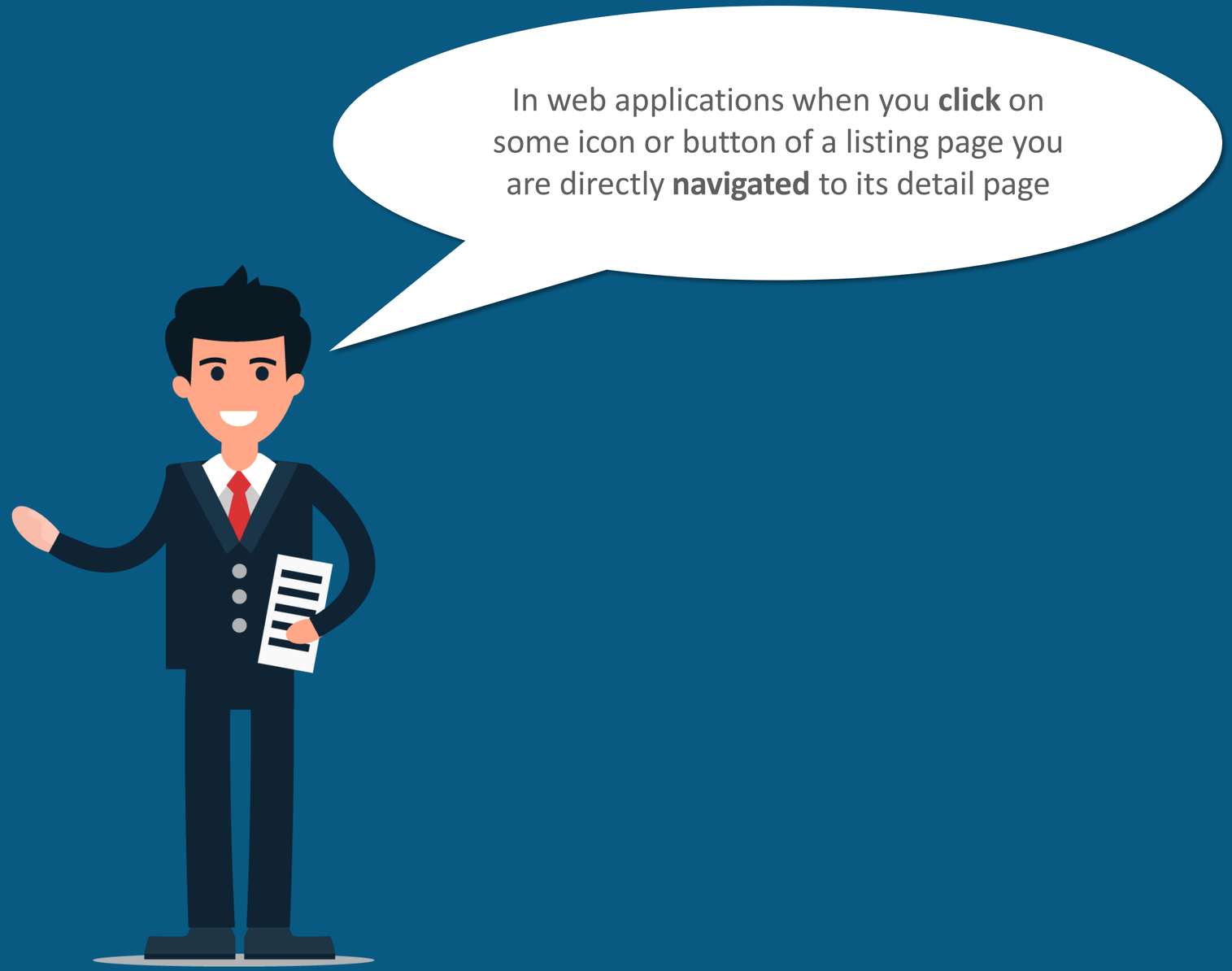
Following are the topics covered in this module:

- Routing
- react-router
- Features of react-router
- Configuration of routing using react-router
- Navigation using Links
- 404 page (Page Not Found)
- URL Parameters
- Nested Routes
- Implementing Styles using NavLink
- Application Programming Interface
- Build a REST API using json-server
- API consumption in React application using Fetch() method
- Build a dynamic music store application using routing and API connectivity

Objectives

After completion of this module you should be able to:

- Understand Routing
- Implement Routes in React application using react-router
- Navigate the user to different sections of your application via Links
- Display “404 page” on entering wrong path
- Let user navigate to different parts of your application via URL parameters
- Integrate sub routes using Nested Routes and styles using NavLink
- Understand API
- Consume API in your React application

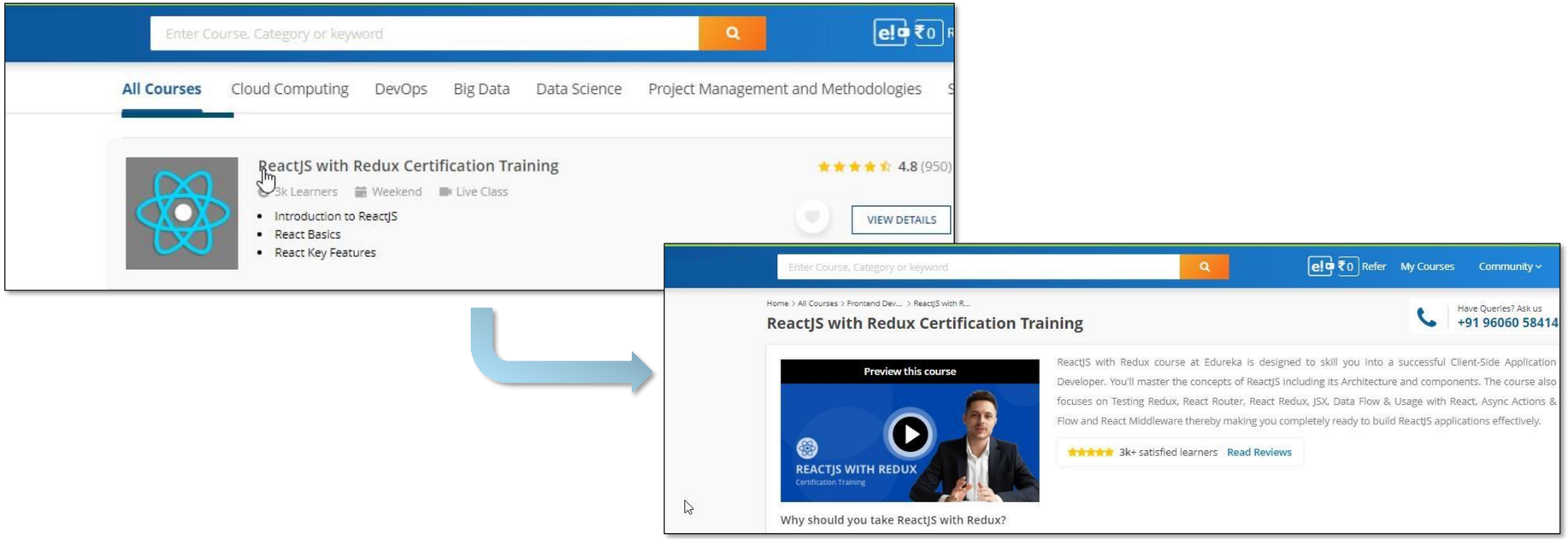


👉 *The transition from a listing page to detail page is possible due to **ROUTING***

Routing

Routing is a process of mapping from the URL to rendered content.

It helps user to move to the different parts of an application, when he enters the URL or clicks an element (link, button, icon, image etc) within the application.



To define a route, we require the ***URL path*** to match with and the ***component*** to render.

react-router

react-router is a popular routing library used to integrate routes in React applications.

This library is split into two parts: ***react-router-dom and react-router-native***

react-router-dom is used to write an application that runs in ***browser*** and *react-router-native* used to write the ***mobile*** applications

You can include react-router using: ***npm install react-router-dom***

Features Of react-router Library

In order to support ***multiple views*** within the ***single page application***, react-router library includes following features:

Navigating backward and forward through the application, maintaining the ***history***, and ***restoring*** the state of the application

Rendering appropriate page components as per the ***URL***

Redirecting the user from one route to the other

Rendering a ***404 page*** when ***none*** of the ***routes match*** the URL

React Router

The ***react-router*** package includes a number of routers that we can take advantage of depending on the platform we are targeting. These include BrowserRouter, HashRouter, and MemoryRouter.

HashRouter is used by static websites to respond the requests of its files

BrowserRouter is used by dynamic websites to handle different URLs

MemoryRouter is used to store the history of the URL, it is used by non-browser applications like React Native applications

For Browser based applications BrowserRouter and HashRouter are a good fit.

Demo 1: Routing Using react-router

Demo: Configuration Of react-router

Open Visual Studio code and configure the react-router using following commands.

npm create-react-app routing → Creates an application: ***app routing***

cd routing → Navigates to folder location

npm install react-router-dom → To install the react-router you need to download the ***react-router-dom package***

npm start → ***Starts*** the server

Demo: Configure The Application Components

Configure ***three*** components: ***App, Customer and Product***

Create a file called App.js within the ***Source folder***

Method that takes input data and returns JSX data

The entire class is exported, so that we can import this component into index.js file

JS Customer.js | JS index.js | JS App.js | JS Product.js

src > JS App.js > App

```
1   import React from 'react'
2   class App extends React.Component {
3     render() {
4       return (
5         <div>
6           <h1>Home</h1>
7         </div>
8       )
9     }
10  }
11
12  export default App
13
```

Imports react library

Data to be displayed when application is routed to App.js

Demo: Configure The Application Components (contd.)

Similarly create ***Customer*** and ***Product*** components.

JS Customer.js | JS index.js ● | JS App.js | JS Product.js ×

src > JS Product.js > ...

```
1  import React from 'react'
2
3  class Product extends React.Component {
4    render() {
5      return <h1>Product</h1>
6    }
7  }
8
9  export default Product
```

JS Customer.js × | JS index.js ● | JS App.js | JS Product.js

src > JS Customer.js > ...

```
1  import React from 'react'
2
3  class Customer extends React.Component {
4    render() {
5      return <h1>Customer</h1>
6    }
7  }
8
9  export default Customer
```

Demo: Import Components In Index.js File

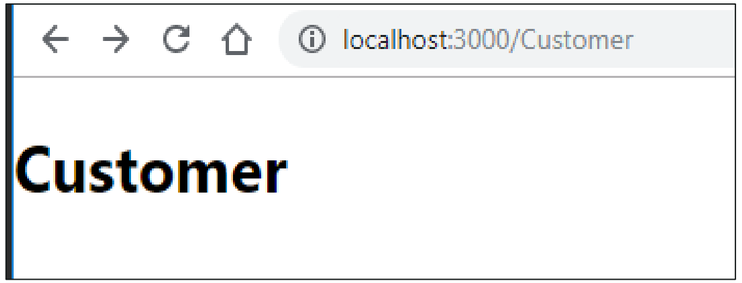
Import the components and ***specify*** the ***path*** and ***component*** in routing section.

```
mport React from 'react'
import ReactDOM from 'react-dom'
import { Route, BrowserRouter as Router } from 'react-router-dom'
import App from './App'
import Customer from './Customer'
import Product from './Product'

const routing = (
  <Router>
    <div>
        <Route exact path="/" component={App} />
        <Route path="/Customer" component={Customer} />
        <Route path="/Product" component={Product} />
    </div>
  </Router>
)
ReactDOM.render(routing, document.getElementById('root'))
```

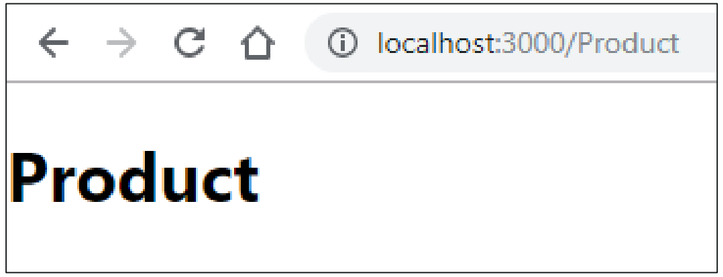
Check The Output

Open the browser, check the routing of application at ***localhost:3000*** using the paths.

Home component rendered at path **/**



Customer component rendered at path ***/Customer***



Product component rendered at path ***/Product***

Navigation Using Links

Manage Navigations Using Links

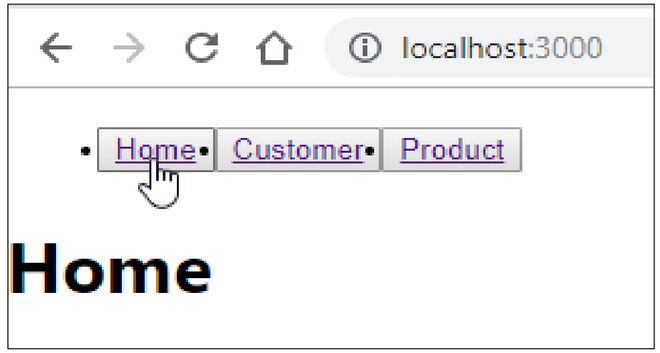
Link component will let you see the routes rendered directly on the screen by clicking the component, instead of mentioning path in URL.

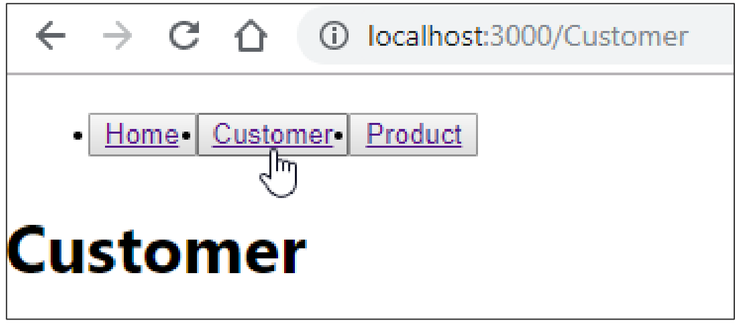
```
const routing = (
<Router>
<div>
<ul>
<button type="button" >
<li>

</li>
</button>
<button type="button">
<li>

</li>
</button>
<button type="button">
<li>

</li>
</button>
</ul>



</div>
</Router>
)
```

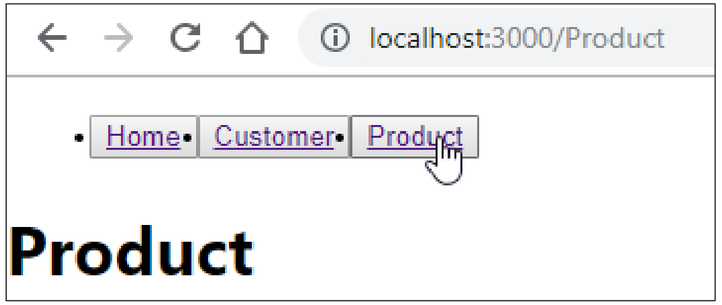
Declaration of path and its corresponding component

Output: Navigation Using Links

On clicking the ***buttons,*** respective component is automatically displayed with its URL.







404 Page

404 Page

01

When user navigates to wrong path, to indicate wrong path, to the user ***404 page*** is used. It is also Known by the term ***Page Not Found***

02

In order to add this page to the website we need to import ***Switch*** component provided by react-router

03

Switch component renders the respective component only if ***path matches*** otherwise it displays 404 page

Configure 404 Page

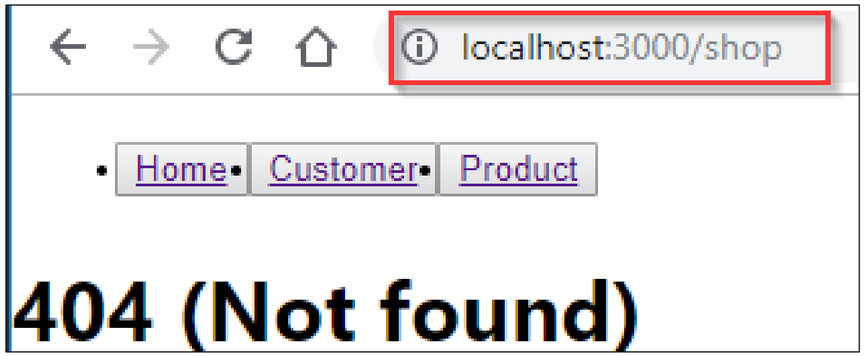
1 Create a component called Notfound

```
mer.js    JS index.js    JS Notfound.js ●    JS App.js
Notfound.js > ...
import React from 'react'

const Notfound = () => <h1>404 (Not found) </h1>

export default Notfound
```

2 Add Switch component to index.js

```
import Notfound from './Notfound'

const routing = (
  <Router>
    <div>
      <ul>
      <button type="button" >
        <li>
          <Link to="/">Home</Link>
        </li>
        </button>
        <button type="button">
        <li>
          <Link to="/Customer">Customer</Link>
        </li>
        </button>
        <button type="button">
        <li>
          <Link to="/Product">Product</Link>
        </li>
        </button>
      </ul>
      <switch>
        <Route exact path="/" component={App} />
        <Route path="/Customer" component={Customer} />
        <Route path="/Product" component={Product} />
        <Route component={Notfound} />
        </switch>
    </div>
  </Router>
```

3 Check the working by entering wrong path



The URLs that you've seen so far are all ***static***.
But applications will use both ***static and dynamic routes***.
Now its time to learn how to ***pass*** dynamic URL segments into your components.

URL Parameter

URL Parameter

URL Parameters are parameters whose values are set dynamically in a page's URL.

01 URL parameters help the ***search engines*** to handle ***parts*** of your site as per your URL, in order to crawl your site more efficiently

02 URL parameters are also used to ***pass variables*** from one webpage to the other

03 These parameters represent the folders within URL string

04 Let us assume that we have customers with Ids 111 and 112, so now we will see how to reach customers with their Ids

Configuration Of URL Parameter: Customer.js

Let us render ***customer component along with id***, open ***Customer.js file*** and add the below code.

```
import React from 'react'

class Customer extends React.Component
{ render() {
const { params } =
this.props.match return (
<div>
<h1>Customer</h1>
<p>{params.id}</p>
</div
)
}
}

export default Customer
```

Collects the URL parameters

Displays the data associated with URL parameter

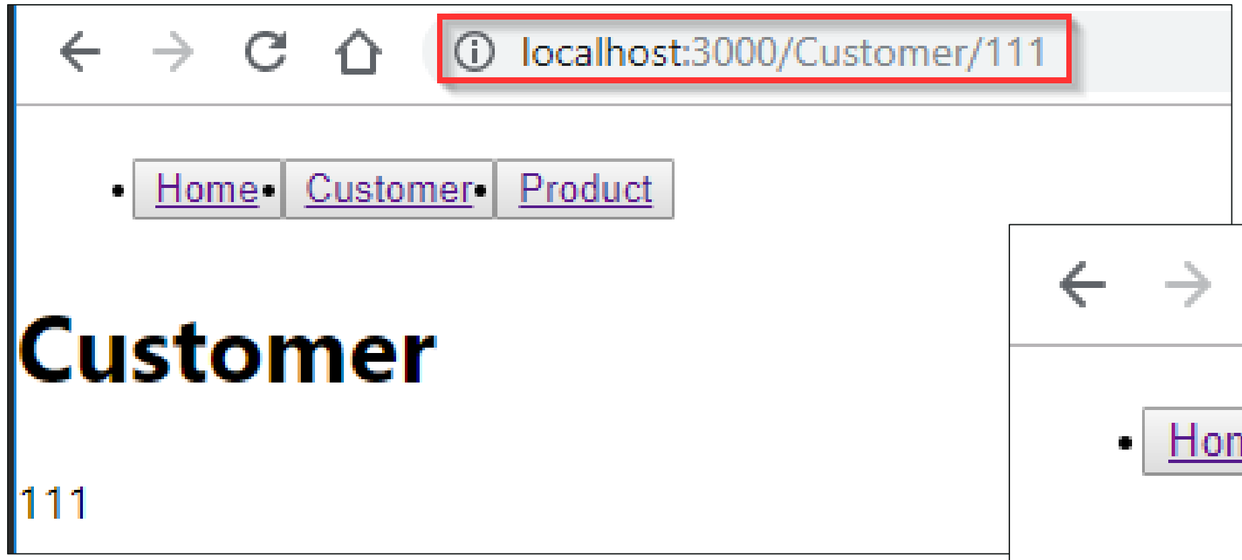
Configuration Of URL Parameter: Index.js

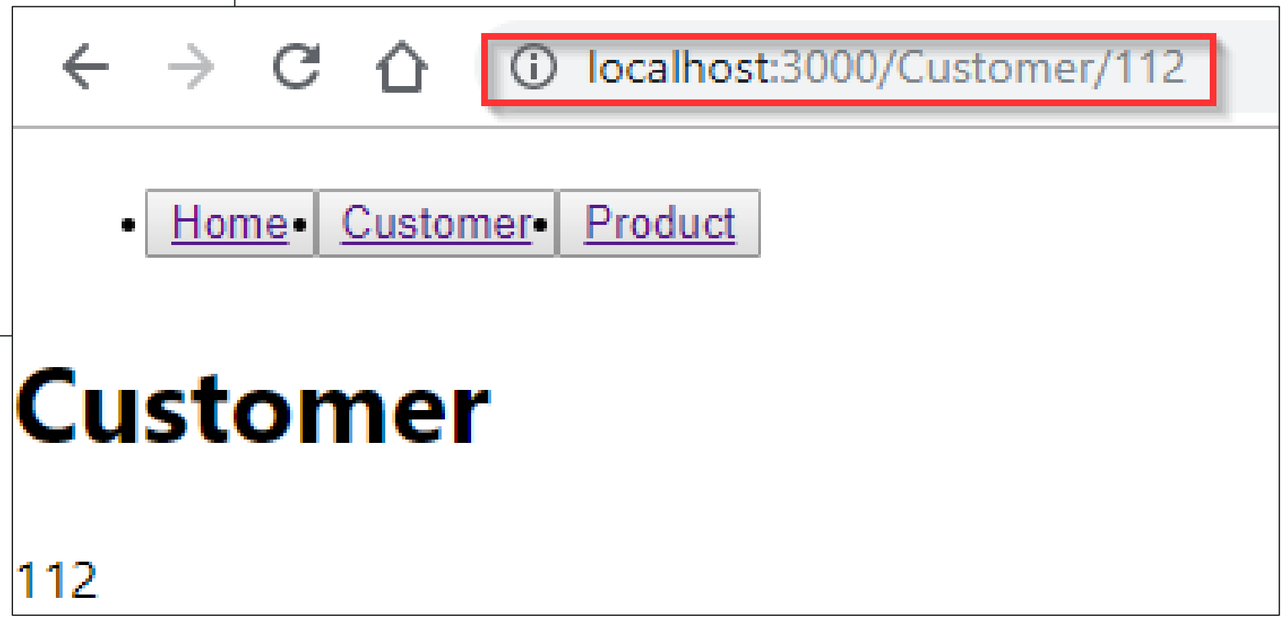
Open ***Index.js file*** and configure the correct path of the ***URL parameter*** in ***Switch*** section of routing.

```
10  const routing = (
11    <Router>
12      <div>
13        <ul>
14        <button type="button" >
15          <li>
16            <Link to="/">Home</Link>
17          </li>
18          </button>
19          <button type="button">
20          <li>
21            <Link to="/Customer">Customer</Link>
22          </li>
23          </button>
24          <button type="button">
25          <li>
26            <Link to="/Product">Product</Link>
27          </li>
28          </button>
29        </ul>
30        <Switch>
31          <Route exact path="/" component={App} />
32          <Route path="/Customer/:id" component={Customer} />
33          <Route path="/Product" component={Product} />
34          <Route component={Notfound} />
35          </Switch>
36      </div>
37    </Router>
```

Configuration Of URL Parameter: Output

Type localhost:3000/Customer/<id no> to check the working of the code.

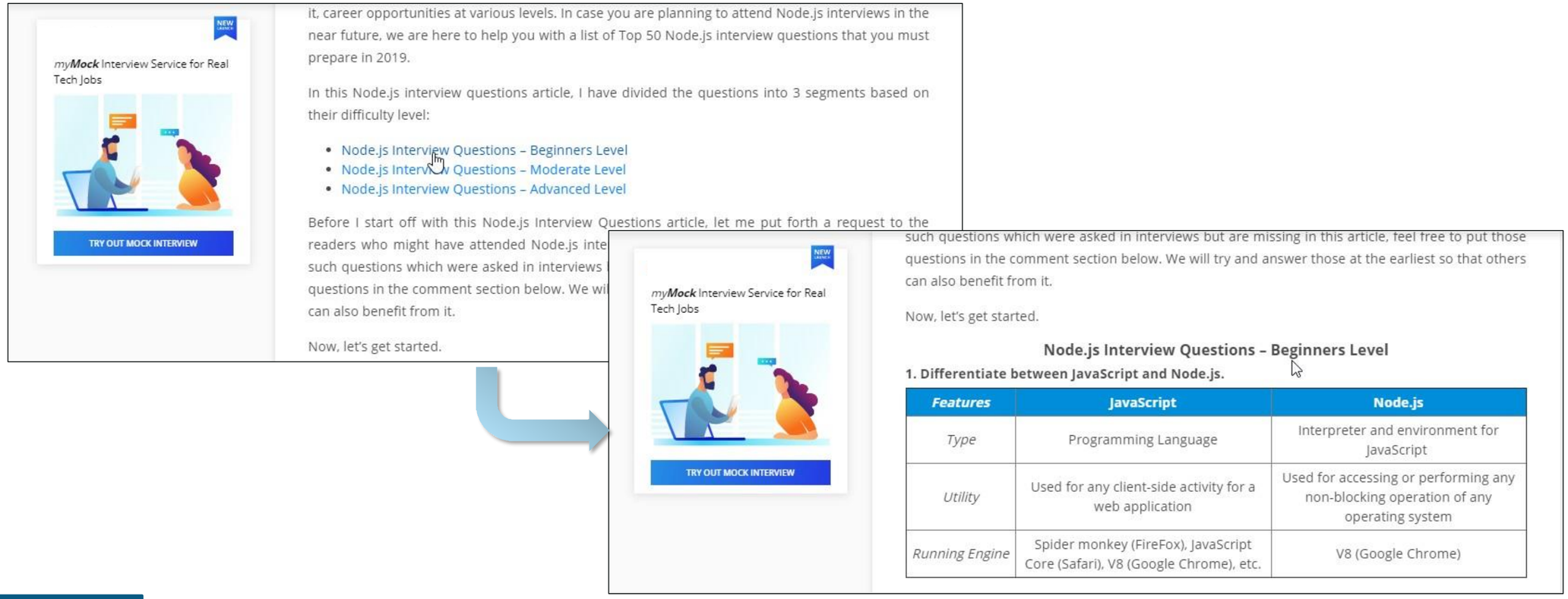




Nested Routes

Nested Routes

- ***Nested Routes*** is a process of defining sub-routes, where user is navigated to different sections of web page on clicking the key words
- Example: Webpages displaying blogs have different sections of topics, if you want to go to the specific topic, then you can just click on it and you will be navigated to details section of the topic



Demo 2: Nested Routes

Demo: Configuration Of Nested Routing

Open ***Product.js file***, import the ***react-router*** components to implement the ***sub routes*** inside the ***Product Component***.
Add the below code in Product.js file to check the ***Nested Routing***.

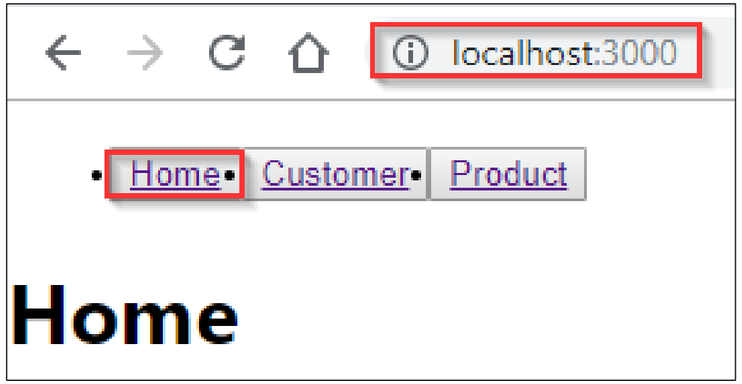
```
import React from 'react'
import { Route, Link} from 'react-router-dom'

const Products = ({ match }) => <p>{match.params.id}</p>

class Product extends React.Component {
render() {
const { url } = this.props.match
return (
<div>
<h1>Product</h1>
<strong>select a Product</strong>
<ul>
<li>
<Link to ="/Product/Secret, Alchemist, SCOOP"> Books</Link>
</li>
<li>
<Link to ="/Product/Addgel, Trimax, Cello"> Pen</Link>
</li>
</ul>
<Route path= "/Product/:id" component ={Products} />
</div>
)}}
export default Product
```

Items to be displayed

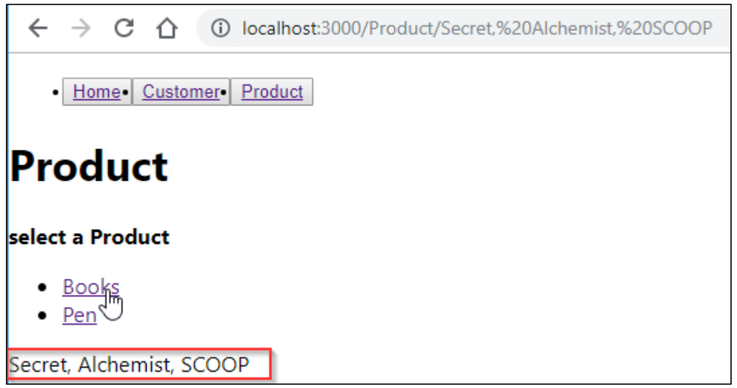
Demo: Output

Run your application at localhost:3000 and click on the tabs shown below to get the specified output.

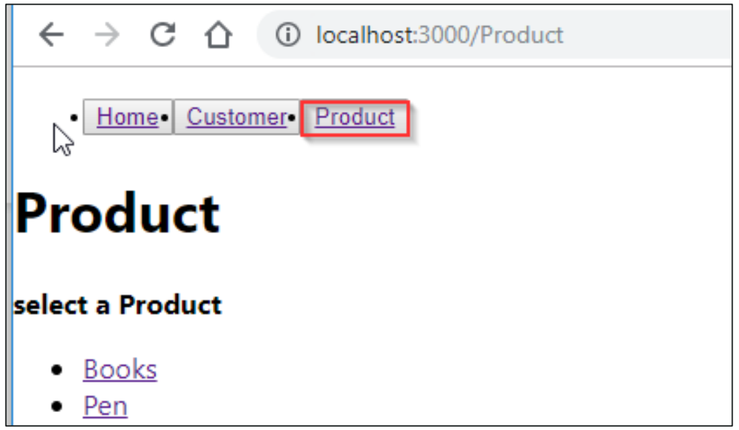
01



03



02

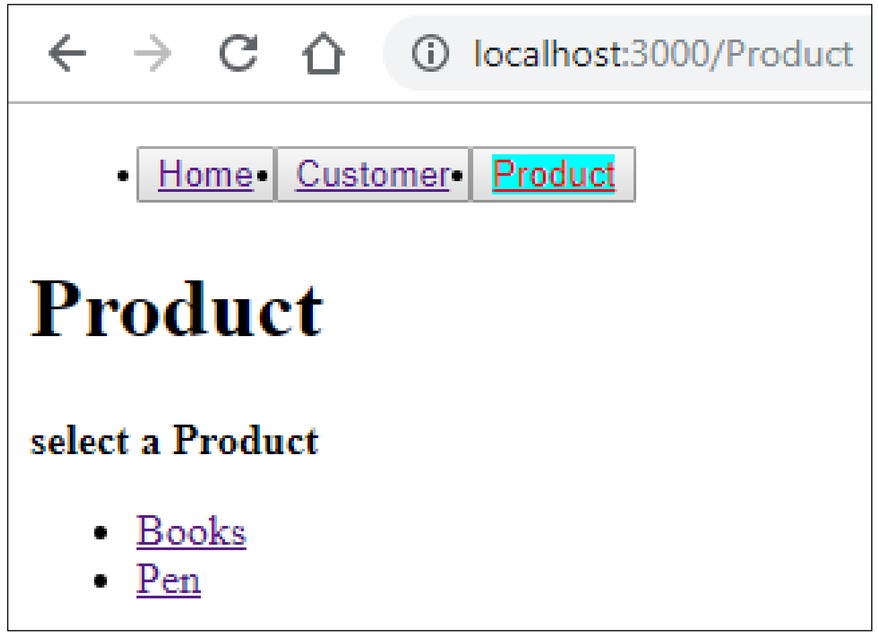


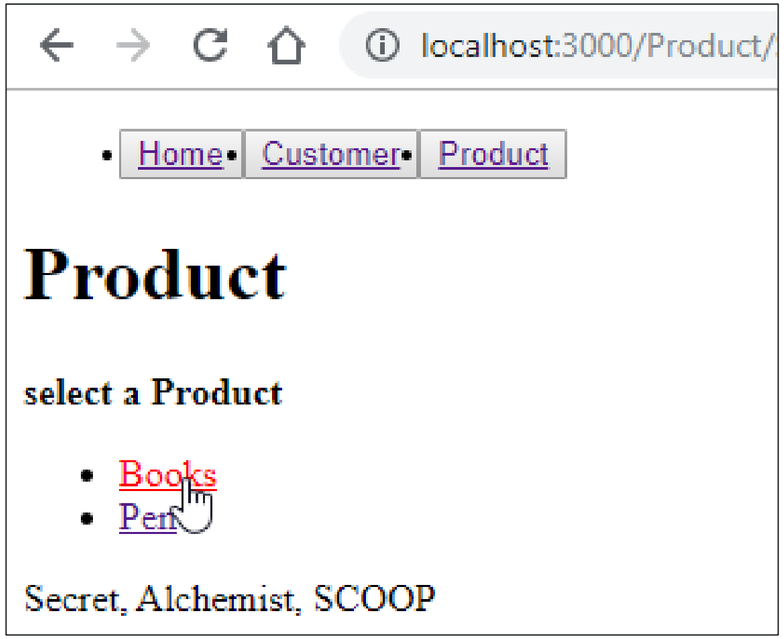
04

Implementing Styles Using NavLink

Implementing Styles Using NavLink

- It is used to ***style*** the ***active routes*** so that user knows in which part of website he or she is currently browsing
- ***Difference*** between Link and NavLink: ***Link*** component is used to ***navigate*** the different routes on the site, but ***NavLink*** is used to add the ***style attributes*** to the ***active routes***





Demo 3: Navigation Using Links

Application Programming Interface

Application Programming Interface (API)

API (Application Program Interface) are the codes that governs the access point of your application to communicate with other application in an agreed way(Request and Response).

- It is a software interface that allows two applications to interact with each other without any user intervention
- ***REST (Representational State Transfer)*** *is an API that allows you to access or manipulate the resource using a set of predefined operations through HTTP protocol methods*
- REST APIs are used in most of the applications as the requests are based on the universal ***HTTP protocol***, and returns the data in ***JSON format*** which most of the programming languages can read

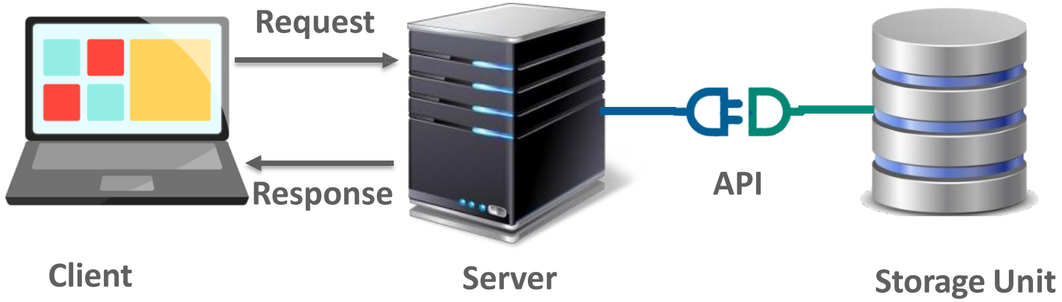


Fig: Flow of data via API

HTTP Methods Used To Interact With Data

REST APIs make use of following ***HTTP Methods*** to interact with the data.

Create	POST/endpoint
Read	GET/endpoint
Update	PATCH/endpoint/:id PUT/endpoint/:id
Delete	DELETE/endpoint/:id

How To Create An API?

While building a frontend web application, if you want to ***connect*** to backend or ***consume*** a remote data using ***API***, you need a server.

We have to make use of ***json-server***

json-server *is a node module that helps you to set up a REST API and provide dynamic access to the remote data*

Building A Rest API Using json-server

- Install json-server globally using the command: ***npm install -g json-server***

```
avi — -bash — 80×24
Last login: Sat Aug 31 11:01:14 on ttys002
(base) Avyaans-MacBook-Pro:~ avi$ npm i -g json-server
```

- Create a JSON file db2.json, it should contain the data that should be rendered by the API

```
src > component > {} db2.json > ...
{
    "products":[
        {
            "id":1,
            "name":"Grand Piano",
            "price":44500,
            "type":"manual",
            "description":"This is a larger baby-grand piano with fuller sound due to its bigger size and longer strings",
            "img":"https://i.ibb.co/wc6qzwW/piano.png"
        },
        {
            "id":2,
            "name":"Electric Guitar",
            "price":11337,
            "type":"Electric",
            "description":"The Bullet Strat With Tremolo HSS is a simple, affordable and practical guitar designed for begi
            "img":"https://i.ibb.co/JsbJrBB/electricguitar.png"
        },
```

Building A Rest API Using json-server (contd.)

- Start the json-server using the command: ***json-server --watch <JSON file name> --port <number>***

```
component — node /usr/local/bin/json-server --watch db2.json --port 6700 — 130×24
(base) Avyaans-MacBook-Pro:component avi$ ls
addProduct.js          db.json                header.js              product.css            products.js
app.js                 db2.json               home.js                productListing.js      routing.js
(base) Avyaans-MacBook-Pro:component avi$ json-server --watch db2.json --port 6700

  \{^_^}/ hi!

  Loading db2.json
  Done

  Resources
  http://localhost:6700/products

  Home
  http://localhost:6700

  Type s + enter at any time to create a snapshot of the database
  Watching...
```

- The watch parameter starts the server in watch mode. It monitors the file changes and updates the API accordingly

Building A Rest API Using json-server (contd.)

- Enter the URL in the browser

localhost:6700/products


```
// 20190831110704
// http://localhost:6700/products

[
  {
    "id": 1,
    "name": "Grand Piano",
    "price": 44500,
    "type": "manual",
    "description": "This is a larger baby-grand piano with fuller sound due to its bigger size and longer strings",
    "img": "https://i.ibb.co/wc6qzwW/piano.png"
  },
  {
    "id": 2,
    "name": "Electric Guitar",
    "price": 11337,
    "type": "Electric",
    "description": "The Bullet Strat With Tremolo HSS is a simple, affordable and practical guitar designed for beginning players and students.",
    "img": "https://i.ibb.co/JsbJrBB/electricguitar.png"
  },
```

How To Consume An API Via React Application

Promises

Promises are used to handle asynchronous operations in JavaScript and provide better error handling (than earlier methods: callbacks and events).

Promise constructor takes only one argument: ***callback function***

var promise = new Promise(function(resolve, reject)

The two arguments that call back function must take are, resolve and reject

If *the operation performed inside the callback function does not go as desired, then it would call* ***reject***

If the operation performed inside the callback function as desired, then it would call **resolve**

Fetch Method

fetch() is the method used to call the API, where we resolve ***promise*** to get the data and display the it in the ***console.log().***

```
componentDidMount(){

        fetch(url,{method:'GET'})
        .then((response) => response.json())
        .then((data) => {
            this.setState({
                products:data
            })
        })
    }
console.log(this.state.products)
```

Endpoint of an API

Sets the value of our state to the output from the API call

Executes fetch() method immediately when component is mounted to make an API request

Makes a GET request to the endpoint

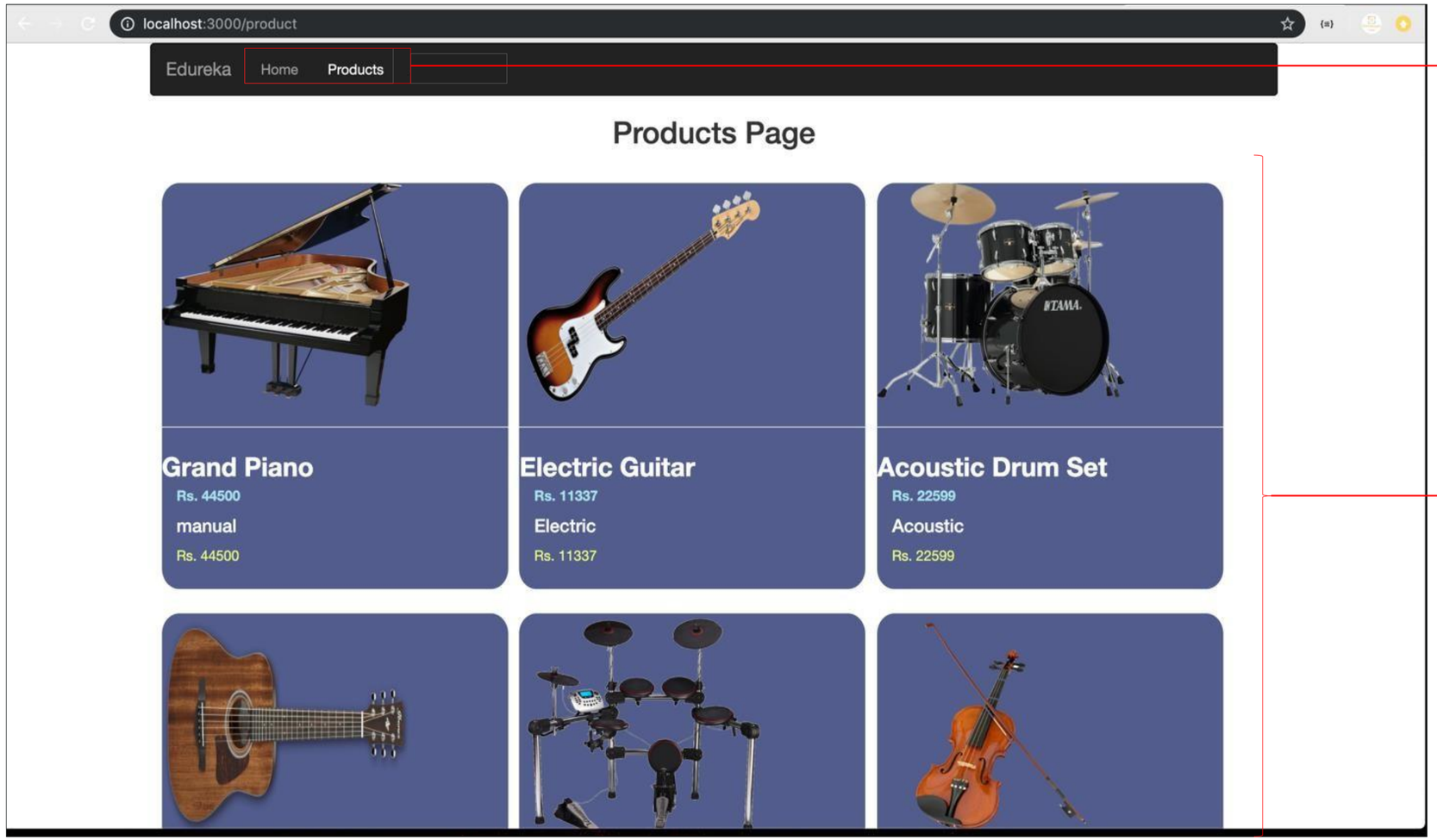
Parses the output to JSON

Displays API data

Demo 4: Dynamic Music Store Application With API Connectivity And Routing

Demo: Product List Page

Product List Page of Application where data is rendered by an API.

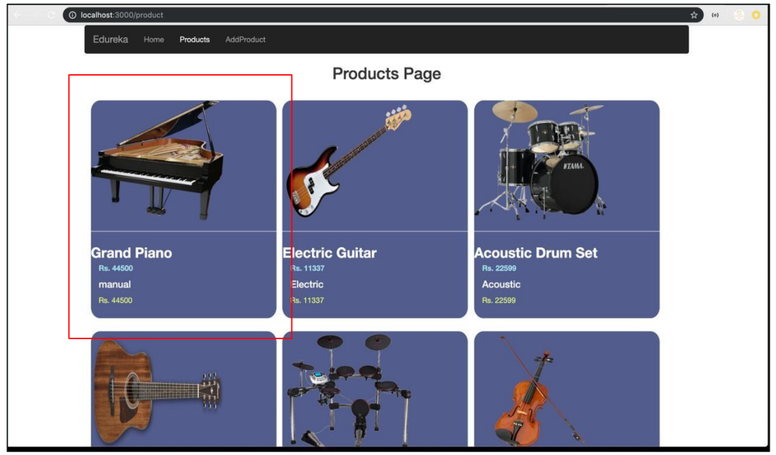


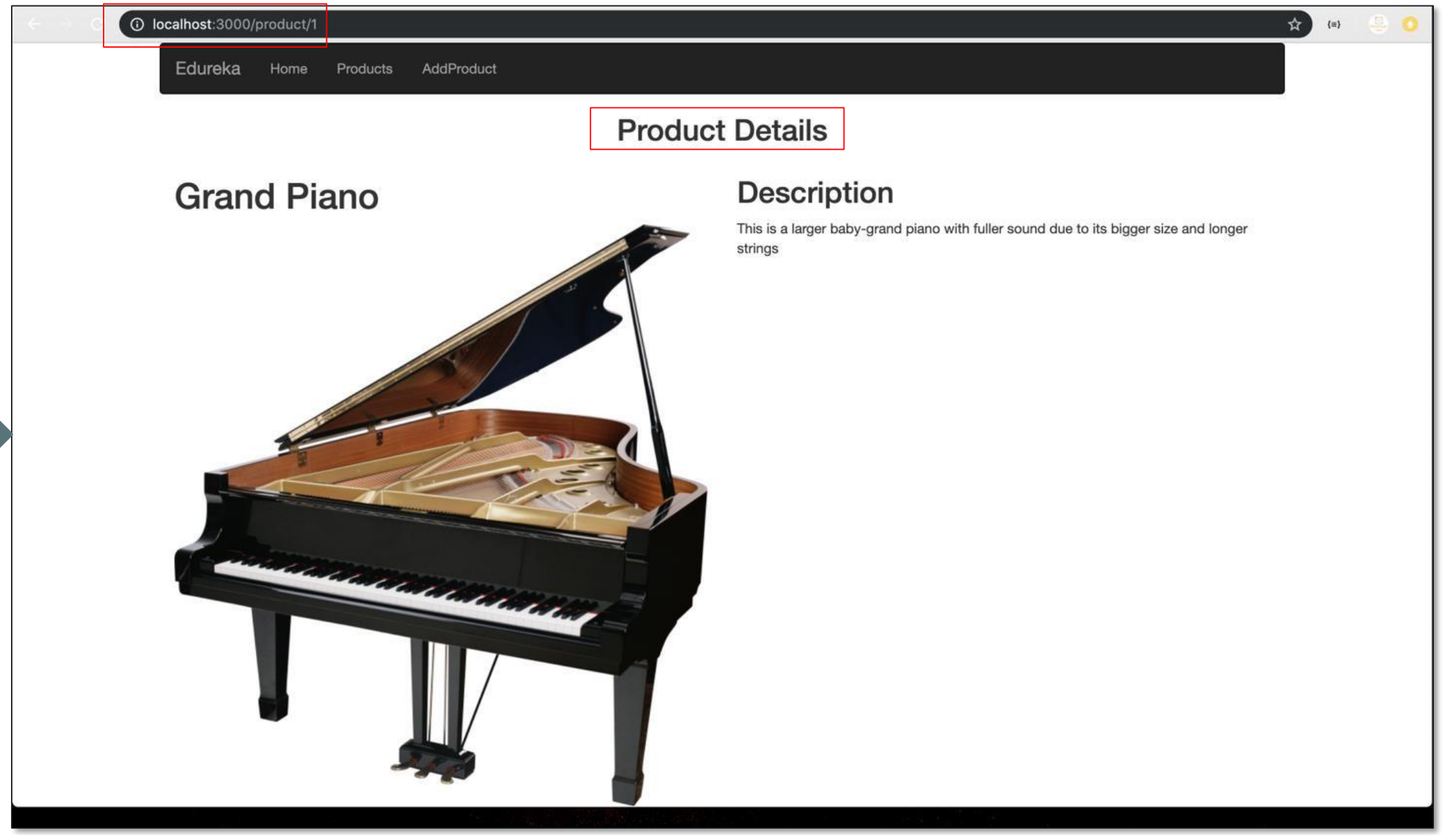
Different Routing sections

Data consumed via API

Demo: Product Details Page

Page Transition from Product List page to Product Details Page.





Questions


Comments
Ratings
Suggestions
FEEDBACK
Survey
Likes
Ideas