

Chapter 9 – Optimising App

Why to use hook in React when we have normal JavaScript function?

Because in hooks we can define state variables so that react can keep track of these variables and triggers reconciliation on every state change. When it comes to a normal JavaScript function we can't define state variables inside the function so that React cannot trigger reconciliation process.

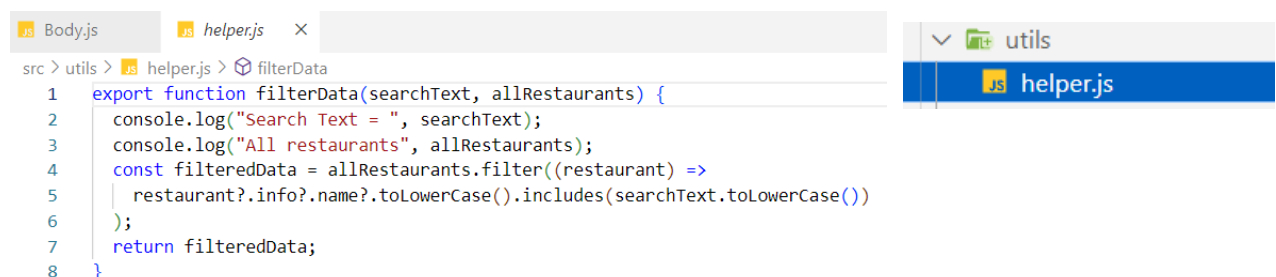
Why should we build our own hook? Why custom hook?

- * **Reusability** - Hooks are JavaScript functions. In a function we write logic once and use it as many times as per the need. This is code reusability. Just like functions we can also reuse React hooks.
- * **Readability**: By looking at the hook we get to know why the hook is built for. We provide meaningful name to the custom hooks so that we get to know what the hook should be doing in our code.
- * **Modularity**: Each react component should have a single responsibility which is to render the data in UI. We should separate functional logics away from the component and offloads this responsibility to a custom hook or a helper function and use them whenever required. This is called modularity meaning breaking down the code into smaller pieces, each having their own responsibility.
- * **Testability**: We break down larger piece of code into smaller chunks that we store in utilities files which enables us to write more test cases.
- * A component should focus solely on a single task for which it is built for even if it has the capability to carry out multiple tasks.

Assuming we have a component that makes an API call to fetch some data and once the data is available, component renders the data in UI. Two tasks are being performed by the component, API call and data rendering. We should offload API call responsibility to a custom hook so that our component will focus only on one specific task which is data rendering in UI.

We have written the code in such a way that it is breaking Modularity.

We have written **Filter Logic** inside Body component. As per **SRP**, Body component should only render data in UI. Body component should not worry about how the data is **Filtered**. We separated this logic out from Body component and placed it inside a helper module so that when Body component needs this module functionality, it gets it from the import.



```
src > utils > js helper.js > filterData
1 export function filterData(searchText, allRestaurants) {
2   console.log("Search Text = ", searchText);
3   console.log("All restaurants", allRestaurants);
4   const filteredData = allRestaurants.filter((restaurant) =>
5     | restaurant?.info?.name?.toLowerCase().includes(searchText.toLowerCase())
6   );
7   return filteredData;
8 }
```

```

JS Body.js x JS helper.js
src > components > JS Body.js > [e] Body > filteredRestaurants.map() callback
1 import RestaurantCards from "../RestaurantCards";
2 import { useState, useEffect } from "react";
3 import ShimmerUI from "../ShimmerUI.js";
4 import { Link } from "react-router-dom";
5 import { filterData } from "../utils/helper";
6 import useOnline from "../utils/useOnline";
7
8 const Body = () => {
9   const [filteredRestaurants, setFilteredRestaurants] = useState([]);
10  const [allRestaurants, setAllRestaurants] = useState([]);
11  const [searchText, setSearchText] = useState("KFC");
12
13  > useEffect(() => { ...
14    }, []);
15
16
17  > async function getRestaurants() { ...
18    }
19
20  const isOnline = useOnline();
21  > if(!isOnline){ ...
22    }
23
24  if (!allRestaurants) return null;
25
26  return allRestaurants.length === 0 ? (
27    <ShimmerUI />
28  ) : (
29    <>
30      <div className="search-container">
31        <input
32          type="text"
33          className="search-input"
34          placeholder="Search"
35          value={searchText}
36          onChange={e => setSearchText(e.target.value)}
37        />
38        <button
39          className="search-btn"
40          onClick={() => {
41            const filteredData = filterData(searchText, allRestaurants);
42            setFilteredRestaurants(filteredData);
43          }}
44        >
45          Search
46        </button>
47      </div>
48      <div className="restaurant-List">
49        {filteredRestaurants?.map((restaurant) => {
50          {
51            console.log(restaurant);
52          }
53          return (
54            <Link to={"/restaurant/" + restaurant.info.id}>
55              <RestaurantCards data={restaurant} key={restaurant.info.id} />
56            </Link>
57          )
58        })}
59      </div>
60    </>
61  );
62
63  >
64  >
65  >
66  >
67  >
68  >
69  >
70  >
71  >
72  >
73  >
74  >
75  >
76  >
77  >
78  >
79  >
80  >
81  >
82  >
83  >
84  >
85  >
86  >
87  >
88  >
89  >
90  >
91  >
92  >
93  >
94  >
95  >
96  >
97  >
98  >
99  >
100 >
101 >
102 >
103 >
104 >
105 >
106 >
107 >
108 >
109 >
110 >
111 >
112 >
113 >
114 >
115 >
116 >
117 >
118 >
119 >
120 >
121 >
122 >
123 >
124 >
125 >
126 >
127 >
128 >
129 >
130 >
131 >
132 >
133 >
134 >
135 >
136 >
137 >
138 >
139 >
140 >
141 >
142 >
143 >
144 >
145 >
146 >
147 >
148 >
149 >
150 >
151 >
152 >
153 >
154 >
155 >
156 >
157 >
158 >
159 >
160 >
161 >
162 >
163 >
164 >
165 >
166 >
167 >
168 >
169 >
170 >
171 >
172 >
173 >
174 >
175 >
176 >
177 >
178 >
179 >
180 >
181 >
182 >
183 >
184 >
185 >
186 >
187 >
188 >
189 >
190 >
191 >
192 >
193 >
194 >
195 >
196 >
197 >
198 >
199 >
200 >
201 >
202 >
203 >
204 >
205 >
206 >
207 >
208 >
209 >
210 >
211 >
212 >
213 >
214 >
215 >
216 >
217 >
218 >
219 >
220 >
221 >
222 >
223 >
224 >
225 >
226 >
227 >
228 >
229 >
230 >
231 >
232 >
233 >
234 >
235 >
236 >
237 >
238 >
239 >
240 >
241 >
242 >
243 >
244 >
245 >
246 >
247 >
248 >
249 >
250 >
251 >
252 >
253 >
254 >
255 >
256 >
257 >
258 >
259 >
260 >
261 >
262 >
263 >
264 >
265 >
266 >
267 >
268 >
269 >
270 >
271 >
272 >
273 >
274 >
275 >
276 >
277 >
278 >
279 >
280 >
281 >
282 >
283 >
284 >
285 >
286 >
287 >
288 >
289 >
290 >
291 >
292 >
293 >
294 >
295 >
296 >
297 >
298 >
299 >
300 >
301 >
302 >
303 >
304 >
305 >
306 >
307 >
308 >
309 >
310 >
311 >
312 >
313 >
314 >
315 >
316 >
317 >
318 >
319 >
320 >
321 >
322 >
323 >
324 >
325 >
326 >
327 >
328 >
329 >
330 >
331 >
332 >
333 >
334 >
335 >
336 >
337 >
338 >
339 >
340 >
341 >
342 >
343 >
344 >
345 >
346 >
347 >
348 >
349 >
350 >
351 >
352 >
353 >
354 >
355 >
356 >
357 >
358 >
359 >
360 >
361 >
362 >
363 >
364 >
365 >
366 >
367 >
368 >
369 >
370 >
371 >
372 >
373 >
374 >
375 >
376 >
377 >
378 >
379 >
380 >
381 >
382 >
383 >
384 >
385 >
386 >
387 >
388 >
389 >
390 >
391 >
392 >
393 >
394 >
395 >
396 >
397 >
398 >
399 >
400 >
401 >
402 >
403 >
404 >
405 >
406 >
407 >
408 >
409 >
410 >
411 >
412 >
413 >
414 >
415 >
416 >
417 >
418 >
419 >
420 >
421 >
422 >
423 >
424 >
425 >
426 >
427 >
428 >
429 >
430 >
431 >
432 >
433 >
434 >
435 >
436 >
437 >
438 >
439 >
440 >
441 >
442 >
443 >
444 >
445 >
446 >
447 >
448 >
449 >
450 >
451 >
452 >
453 >
454 >
455 >
456 >
457 >
458 >
459 >
460 >
461 >
462 >
463 >
464 >
465 >
466 >
467 >
468 >
469 >
470 >
471 >
472 >
473 >
474 >
475 >
476 >
477 >
478 >
479 >
480 >
481 >
482 >
483 >
484 >
485 >
486 >
487 >
488 >
489 >
490 >
491 >
492 >
493 >
494 >
495 >
496 >
497 >
498 >
499 >
500 >
501 >
502 >
503 >
504 >
505 >
506 >
507 >
508 >
509 >
510 >
511 >
512 >
513 >
514 >
515 >
516 >
517 >
518 >
519 >
520 >
521 >
522 >
523 >
524 >
525 >
526 >
527 >
528 >
529 >
530 >
531 >
532 >
533 >
534 >
535 >
536 >
537 >
538 >
539 >
540 >
541 >
542 >
543 >
544 >
545 >
546 >
547 >
548 >
549 >
550 >
551 >
552 >
553 >
554 >
555 >
556 >
557 >
558 >
559 >
560 >
561 >
562 >
563 >
564 >
565 >
566 >
567 >
568 >
569 >
570 >
571 >
572 >
573 >
574 >
575 >
576 >
577 >
578 >
579 >
580 >
581 >
582 >
583 >
584 >
585 >
586 >
587 >
588 >
589 >
590 >
591 >
592 >
593 >
594 >
595 >
596 >
597 >
598 >
599 >
600 >
601 >
602 >
603 >
604 >
605 >
606 >
607 >
608 >
609 >
610 >
611 >
612 >
613 >
614 >
615 >
616 >
617 >
618 >
619 >
620 >
621 >
622 >
623 >
624 >
625 >
626 >
627 >
628 >
629 >
630 >
631 >
632 >
633 >
634 >
635 >
636 >
637 >
638 >
639 >
640 >
641 >
642 >
643 >
644 >
645 >
646 >
647 >
648 >
649 >
650 >
651 >
652 >
653 >
654 >
655 >
656 >
657 >
658 >
659 >
660 >
661 >
662 >
663 >
664 >
665 >
666 >
667 >
668 >
669 >
670 >
671 >
672 >
673 >
674 >
675 >
676 >
677 >
678 >
679 >
680 >
681 >
682 >
683 >
684 >
685 >
686 >
687 >
688 >
689 >
690 >
691 >
692 >
693 >
694 >
695 >
696 >
697 >
698 >
699 >
700 >
701 >
702 >
703 >
704 >
705 >
706 >
707 >
708 >
709 >
710 >
711 >
712 >
713 >
714 >
715 >
716 >
717 >
718 >
719 >
720 >
721 >
722 >
723 >
724 >
725 >
726 >
727 >
728 >
729 >
730 >
731 >
732 >
733 >
734 >
735 >
736 >
737 >
738 >
739 >
740 >
741 >
742 >
743 >
744 >
745 >
746 >
747 >
748 >
749 >
750 >
751 >
752 >
753 >
754 >
755 >
756 >
757 >
758 >
759 >
760 >
761 >
762 >
763 >
764 >
765 >
766 >
767 >
768 >
769 >
770 >
771 >
772 >
773 >
774 >
775 >
776 >
777 >
778 >
779 >
780 >
781 >
782 >
783 >
784 >
785 >
786 >
787 >
788 >
789 >
790 >
791 >
792 >
793 >
794 >
795 >
796 >
797 >
798 >
799 >
800 >
801 >
802 >
803 >
804 >
805 >
806 >
807 >
808 >
809 >
810 >
811 >
812 >
813 >
814 >
815 >
816 >
817 >
818 >
819 >
820 >
821 >
822 >
823 >
824 >
825 >
826 >
827 >
828 >
829 >
830 >
831 >
832 >
833 >
834 >
835 >
836 >
837 >
838 >
839 >
840 >
841 >
842 >
843 >
844 >
845 >
846 >
847 >
848 >
849 >
850 >
851 >
852 >
853 >
854 >
855 >
856 >
857 >
858 >
859 >
860 >
861 >
862 >
863 >
864 >
865 >
866 >
867 >
868 >
869 >
870 >
871 >
872 >
873 >
874 >
875 >
876 >
877 >
878 >
879 >
880 >
881 >
882 >
883 >
884 >
885 >
886 >
887 >
888 >
889 >
890 >
891 >
892 >
893 >
894 >
895 >
896 >
897 >
898 >
899 >
900 >
901 >
902 >
903 >
904 >
905 >
906 >
907 >
908 >
909 >
910 >
911 >
912 >
913 >
914 >
915 >
916 >
917 >
918 >
919 >
920 >
921 >
922 >
923 >
924 >
925 >
926 >
927 >
928 >
929 >
930 >
931 >
932 >
933 >
934 >
935 >
936 >
937 >
938 >
939 >
940 >
941 >
942 >
943 >
944 >
945 >
946 >
947 >
948 >
949 >
950 >
951 >
952 >
953 >
954 >
955 >
956 >
957 >
958 >
959 >
960 >
961 >
962 >
963 >
964 >
965 >
966 >
967 >
968 >
969 >
970 >
971 >
972 >
973 >
974 >
975 >
976 >
977 >
978 >
979 >
980 >
981 >
982 >
983 >
984 >
985 >
986 >
987 >
988 >
989 >
990 >
991 >
992 >
993 >
994 >
995 >
996 >
997 >
998 >
999 >
1000 >

```

We have written **API call logic** inside Restaurant Menu component. As per **SRP**, Restaurant Menu component should only render data in UI. Restaurant Menu component should not worry about how the data is **fetches from the API**. We separated this logic out from Restaurant Menu component and placed it inside a custom hook in helper module **useRestaurant.js** so that when Restaurant Menu component needs this module functionality, it gets it from the import.

```
src > utils > useRestaurant.js > useRestaurant
1  import { useState, useEffect } from "react";
2  import { FETCH_MENU_URL } from "../constants";
3
4  var Menus = [];
5
6  // creating a Custom Hook
7  const useRestaurant = (id) => {
8    const [restaurant, setRestaurant] = useState(null);
9    const [menuCategories, setMenuCategories] = useState([]);
10
11    // get Data from API
12    useEffect(() => {
13      getRestaurantInfo();
14    }, []);
15
16    async function getRestaurantInfo() {
17      const data = await fetch(
18        FETCH_MENU_URL +
19        id +
20        "&catalog_qa=undefined&metaData=%7B%22type%22%3A%22RESTAURANT%22%2C%22data%22%3A%7B%22parentId%22%3A%22%22primaryRestaurantId%22%3A%229087%22%22cloudinaryId%22%3A%22dw307jhy1c2t3gmzy5zn%22%2C%22brandId%22%3A%22%22%22dishFamilyId%22%3A%22846613%22%22enabled_flag%22%3A%227D%22%22businessCategory%22%3A%22%22SWIGGY_FOOD%22%2C%22displayLabel%22%3A%22Restaurant%22%27D&submitAction=SUGGESTION"
21      );
22      const json = await data.json();
23      const restaurantData = json.data.cards[0].card.card.info;
24      setRestaurant(restaurantData);
25      const MenuItemList = await json.data.cards[2].groupedCard.cardGroupMap
26        .REGULAR.cards;
27      MenuItemList.slice(1, -2).map((item) => {
28        Menus.push(item.card.card.title);
29      });
30      setMenuCategories(Menus);
31    }
32
33    // return restaurant data and menuCategories data
34    return [restaurant, menuCategories];
35  };
36
37  export default useRestaurant;
```

RestaurantMenu.js

X

useRestaurant.js

X

helper.js

X

```

src > components > RestaurantMenu.js > default
1  import { useEffect, useState } from "react";
2  import { useParams } from "react-router-dom";
3  import { IMG_CDN_URL } from "../constants";
4  import ShimmerUI from "../ShimmerUI";
5  import useRestaurant from "../utils/useRestaurant";
6
7  const RestaurantMenu = () => {
8    const { id } = useParams();
9
10   const [restaurant, menuCategories] = useRestaurant(id)
11
12   if (!restaurant) {
13     return <ShimmerUI />;
14   }
15
16   return ( ...
40  );
41  };
42
43  export default RestaurantMenu;

```

Building Online and Offline features –

If the user has no internet connection then it should show **You are Offline** check your internet connection else it should show **the actual data**.

RestaurantMenu.js

X

useOnline.js

X

useRestaurant.js

X

```

src > utils > useOnline.js > ...
1  import { useEffect, useState } from "react";
2
3  const useOnline = () => {
4    const [isOnline, setIsOnline] = useState(true);
5
6    useEffect(() => {
7      const handleOnline = () => {
8        setIsOnline(true);
9      };
10     const handleOffline = () => {
11       setIsOnline(false);
12     };
13     window.addEventListener("online", handleOnline);
14     window.addEventListener("offline", handleOffline);
15     // clean-up code
16     return () => {
17       window.removeEventListener("online", handleOnline);
18       window.removeEventListener("offline", handleOffline);
19     };
20   }, []);
21   return isOnline;
22 };
23
24 export default useOnline;

```

Code Explanation -

In JavaScript, window object provides a method called `addEventListener` where we attach events. In our case we are attaching **online** and **offline** events to **`addEventListener`** method, also we are providing a callback function as a second argument to `addEventListener` which gets called when user gets online or offline.

We want this event to be called just once for both online and offline. That's why we have put them inside `useEffect` hook.

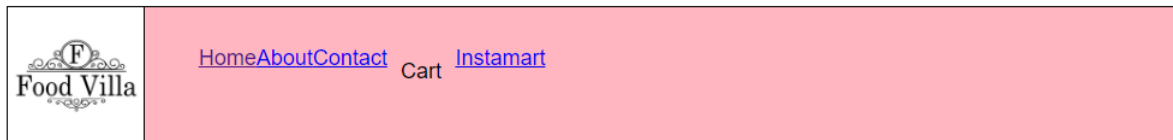
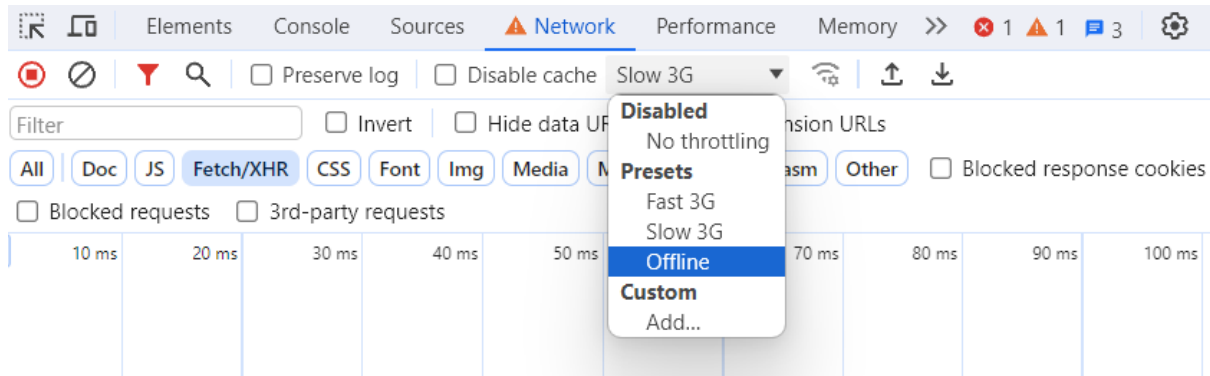
When we navigate from component to component, online and offline event listeners should have been removed but they still exist. It is always a good practice to remove event listeners on components navigation otherwise browser will keep hold to these events which is a big performance hit. In our code we are already handling this via `RemoveEventListener` during **component unload phase**.

```
JS Body.js ×
src > components > JS Body.js > ...
1  import RestaurantCards from "../RestaurantCards";
2  import { useState, useEffect } from "react";
3  import ShimmerUI from "../ShimmerUI.js";
4  import { Link } from "react-router-dom";
5  import { filterData } from "../utils/helper";
6  import useOnline from "../utils/useOnline";
7
8  const Body = () => {
9    const [filteredRestaurants, setFilteredRestaurants] = useState([]);
10   const [allRestaurants, setAllRestaurants] = useState([]);
11   const [searchText, setSearchText] = useState("KFC");
12
13   > useEffect(() => { ...
15     }, []);
16
17   > async function getRestaurants() { ...
28     }
29
30     const isOnline = useOnline();
31     if(!isOnline){
32       |   return <h1>Offline,please check your internet connection !!</h1>
33     }
34
35     if (!allRestaurants) return null;
36
37   > return allRestaurants.length === 0 ? ( ...
39   > ) : ( ...
72   );
73 };
74
75 export default Body;
76
```

Simulating offline online behaviour in chrome browser -

Offline – offline mode

No throttling – online mode



Offline, please check your internet connection !!

Footer

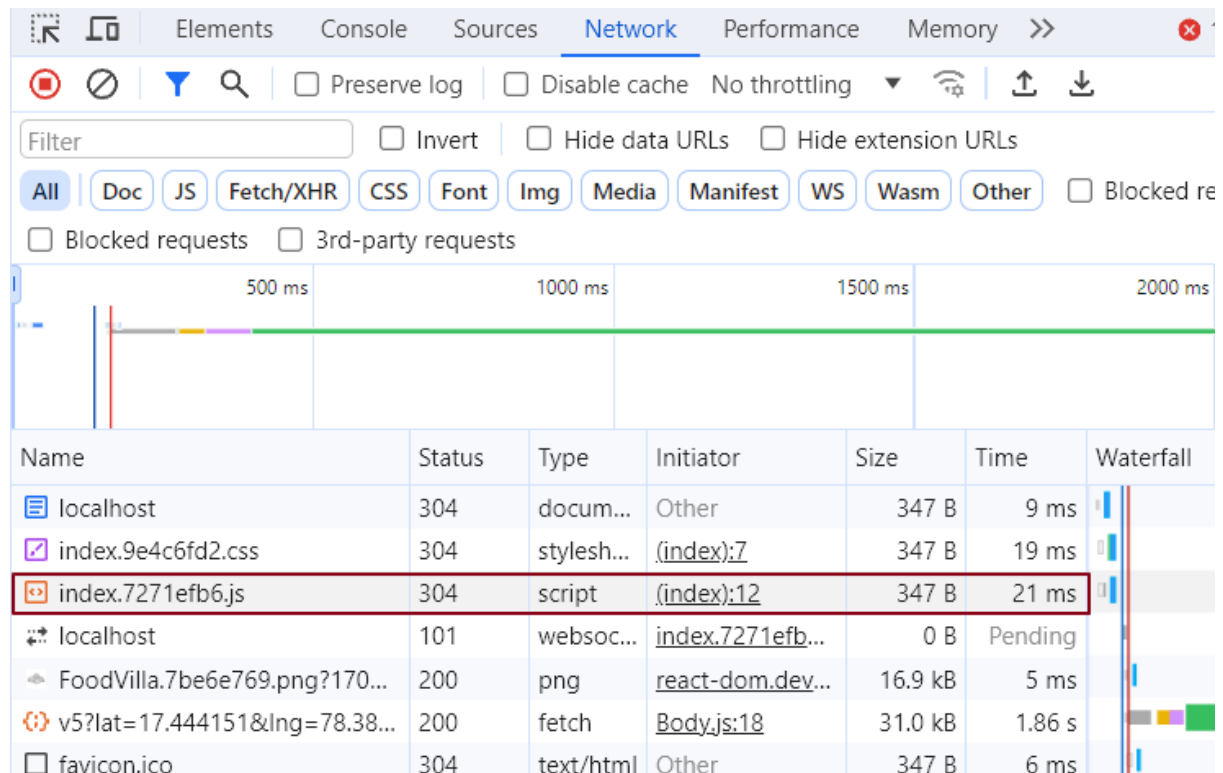
Using `useOnline` hook in header.js -

```
JS useOnline.js JS Header.js X JS Body.js
src > components > JS Header.js > ...
1  import { useState } from "react";
2  import Logo from "../assets/Img/FoodVilla.png";
3  import { Link } from "react-router-dom";
4  import useOnline from "../utils/useOnline";
5
6  const loggedInUser = () => {
7    return true;
8  };
9
10 const Title = () => (
11   <a href="/">
12     <img className="logo" src={Logo} alt="" />
13   </a>
14 );
15
16 const Header = () => {
17   const [isLoggedIn, setIsLoggedIn] = useState(false);
18   const isOnline = useOnline();
19   return (
20     <div className="header">
21       <Title />
22       <div className="nav-items">...
38     </div>
39     <h1>{isOnline ? "Online" : "Offline"}</h1>
40     {isLoggedIn ? (
41       <button onClick={() => setIsLoggedIn(false)}>Log out</button>
42     ) : (
43       <button onClick={() => setIsLoggedIn(true)}>Log in</button>
44     )}
45   </div>
46 );
47 };
48
49 export default Header;
```



With the help of custom hook, we can reuse the code. We can achieve code reusability.

In our application parcel created only one JavaScript file where all of our code is bundled and minified together.



The size of the index file bundle is less in production build.

In large scale application, there are thousands of components and if we bundle them all together in one file the application will become very slow. Because when the page loads for the first time, all components from the bundle will be loaded at once. To make the application scalable, performant and efficient we need to chunk the big bundle into individual mini bundles and load these bundles on certain conditions.

What is bundling? Who does bundle process?

Bundling is a process of merging imported files or code dependencies into a single optimised & minified file. Bundler such as parcel, web pack does the bundling.

What is Chunking / Code Splitting / Dynamic bundling / Lazy Loading / On demand Loading / Dynamic Import?

If we build a large-scale production ready application, there will be multiple components and bundling them all together in a single JS file will take so much time to render in UI. Therefore, we should break the entire production ready code into smaller chunks. This concept is known as Chunking or code splitting.

Is bundling good?

Bundling is good, but to a certain extent.

We don't have to bundle all components code into a single file and load the file in UI. Instead, we should make logical bundles. This means we should code in such a way that bundle corresponding to a specific use case will only get loaded in UI. Let's try to understand this with an example.

* Make My Trip (MMT) is an application that provides many services but primarily people use it for booking flights. So, when they open MMT they land on flights page because from developer perspective we load only the flight components bundle on app initialisation, we don't load every service bundle altogether. We could have also loaded train booking bundle but user at this moment does not want to see train service. When user wants to book a train and lands on the train section that time only we load the train components bundle. **Bottom line is rather than bundling everything up in one go, developers bundle components based on use cases.**

* In a small-scale application, it does not make sense to split bundle into chunks because one bundle size will be enough for the application and this won't cause any performance issue.

Where do we do on demand Loading?

We do on demand loading of a component in a place where bundle will get loaded on demand.

Let us create a component Instamart and load this component **on demand** inside app.js.

```
JS Instamart.js X
src > components > JS Instamart.js > [🔗] Instamart
1
2  const Instamart = () => {
3    return (
4      <div>
5        <h1>Instamart</h1>
6        <h1>100 of components inside of it.</h1>
7      </div>
8    )
9  }
10
11  export default Instamart
```

```

JS Instamart.js JS App.js ×
src > JS App.js > ...
1  import React, { lazy } from "react";
2  import ReactDOM from "react-dom/client";
3  import Header from "../components/Header";
4  import Body from "../components/Body";
5  import Footer from "../components/Footer";
6  import About from "../components/About.js";
7  import { createBrowserRouter, RouterProvider, Outlet } from "react-router-dom";
8  import Error from "../components/Error";
9  import Contact from "../components/Contact";
10 import RestaurantMenu from "../components/RestaurantMenu";
11
12
13 // import Instamart from "../components/Instamart"; // Normal Import
14
15 const Instamart = lazy(() => import("../components/Instamart")); // Dynamic Import
16
17 const AppLayout = () => {
18   return (
19     <>
20       <Header />
21       <Outlet />
22       <Footer />
23     </>
24   );
25 };
26
27 const appRouter = createBrowserRouter([
28   {
29     path: "/",
30     element: <AppLayout />,
31     errorElement: <Error />,
32     children: [
33 >     { ...
36     },
37 >     { ...
40     },
41 >     { ...
44     },
45 >     { ...
48     },
49     {
50       path: "/instamart",
51       element: (
52         <>
53           <Instamart />
54         </>
55       ),
56     },
57   ],
58 },
59 ]);
60
61 const root = ReactDOM.createRoot(document.getElementById("root"));
62 root.render(<RouterProvider router={appRouter} />);

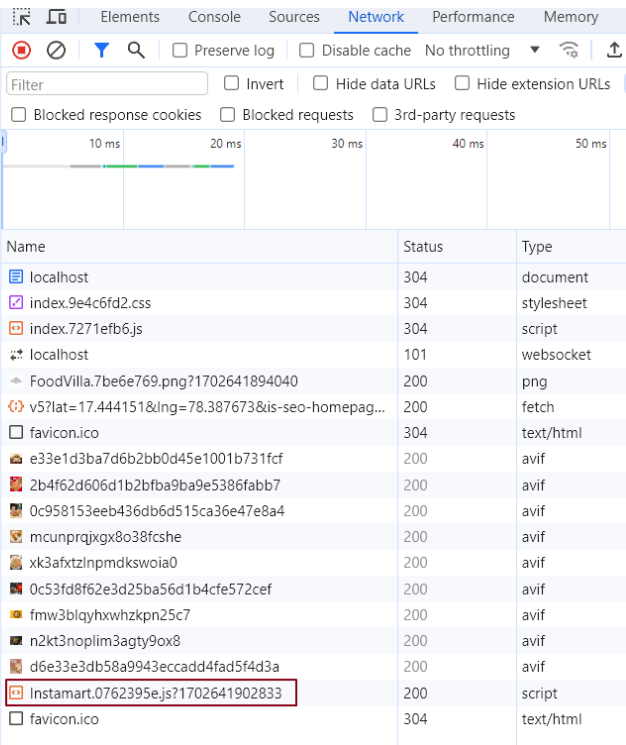
```



[Home](#)
[About](#)
[Contact](#)
[Cart](#)
[Instamart](#)

When we click on Instamart we load the bundle of Instamart component. This is on demand loading

undefined:undefined



Name	Status	Type
localhost	304	document
index.9e4c6fd2.css	304	stylesheet
index.7271efb6.js	304	script
localhost	101	websocket
FoodVilla.7be6e769.png?1702641894040	200	png
v5?lat=17.444151&lng=78.387673&is-seo-homepag...	200	fetch
favicon.ico	304	text/html
e33e1d3ba7d6b2bb0d45e1001b731fcf	200	avif
2b4f62d606d1b2bfba9ba9e5386fabb7	200	avif
0c958153eeb436db6d515ca36e47e8a4	200	avif
mcunprqjxgx8o38fcshe	200	avif
xk3afxtzlnpmdkswioia0	200	avif
0c53fd8f62e3d25ba56d1b4cfe572cef	200	avif
fmw3blqyhxwhzkpn25c7	200	avif
n2kt3noplilm3agty9ox8	200	avif
d6e33e3db58a9943eccadd4fad5f4d3a	200	avif
Instamart.0762395e.js?1702641902833	200	script
favicon.ico	304	text/html

Why undefined:undefined?

When we load instamart component, **instamart component bundle** takes some time to be loaded. Meanwhile react tries to render instamart component which does not exist, meaning this component is inside the bundler and bundler was not loaded. In this process react suspends this rendering. That's why we are getting the error component page in UI. At this moment `err.Status` and `err.statusText` will have `undefined` stored in it.

When we are loading our component on demand, react tries to suspend it.

Instamart card component is a suspense. We don't know whether it will be loaded on UI or not. It will load only when the bundle for this component is loaded.

```

App.js
src > App.js > AppLayout
1  import React, { lazy, Suspense } from "react";
2  import ReactDOM from "react-dom/client";
3  import Header from "../components/Header";
4  import Body from "../components/Body";
5  import Footer from "../components/Footer";
6  import About from "../components/About.js";
7  import { createBrowserRouter, RouterProvider, Outlet } from "react-router-dom";
8  import Error from "../components/Error";
9  import Contact from "../components/Contact";
10 import RestaurantMenu from "../components/RestaurantMenu";
11 import ShimmerUI from "../components/ShimmerUI";
12
13 // import Instamart from "../components/Instamart"; // Normal Import
14
15 const Instamart = lazy(() => import("../components/Instamart")); // Dynamic Import
16
17 const AppLayout = () => {
18   return (
19     <>
20       <Header />
21       <Outlet />
22       <Footer />
23     </>
24   );
25 };
26
27 const appRouter = createBrowserRouter([
28   {
29     path: "/",
30     element: <AppLayout />,
31     errorElement: <Error />,
32     children: [
33       { ...
36     },
37     { ...
40     },
41     { ...
44     },
45     { ...
48     },
49     {
50       path: "/instamart",
51       element: (
52         <Suspense fallback={<ShimmerUI/>}>
53           <Instamart />
54         </Suspense>
55       ),
56     },
57   ],
58 },
59 ]);
60
61 const root = ReactDOM.createRoot(document.getElementById("root"));
62 root.render(<RouterProvider router={appRouter} />);
63

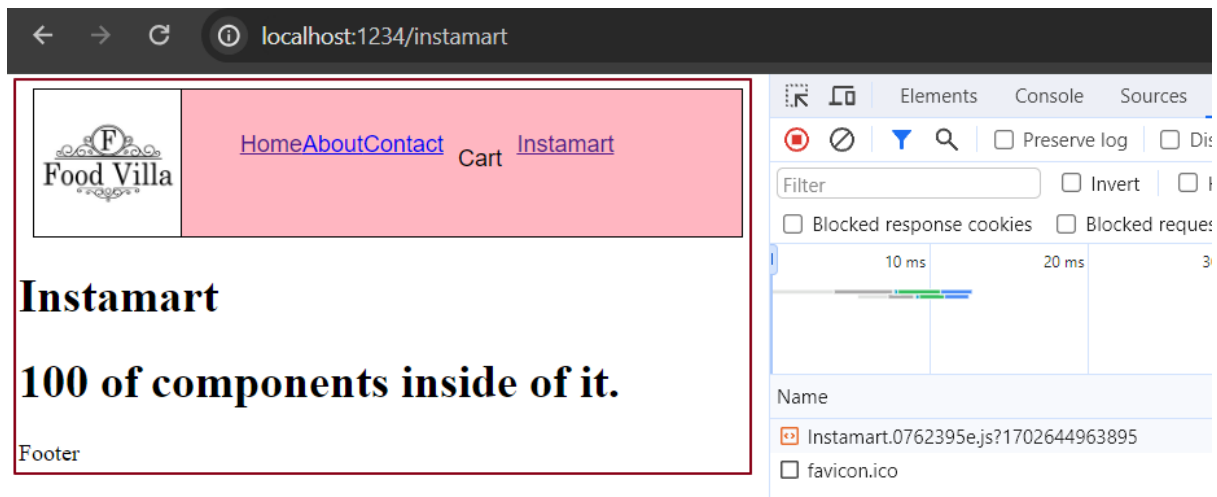
```

What is suspend?

<suspense> suspends the lazy loaded components until bundle for the component is available in UI

```
{
  path: "/instamart",
  element: (
    <Suspense fallback={<ShimmerUI/>}>
      <Instamart />
    </Suspense>
  ),
},
```

Now when we navigate to Instamart page, we see the contents of it because this time contents are coming from the Instamart component from loaded bundle.



Now React knows Instamart Component will be lazy loaded .so it waits for the promise to get resolved.

```
const Instamart = lazy(() => import("./components/Instamart")); // Dynamic Import
```

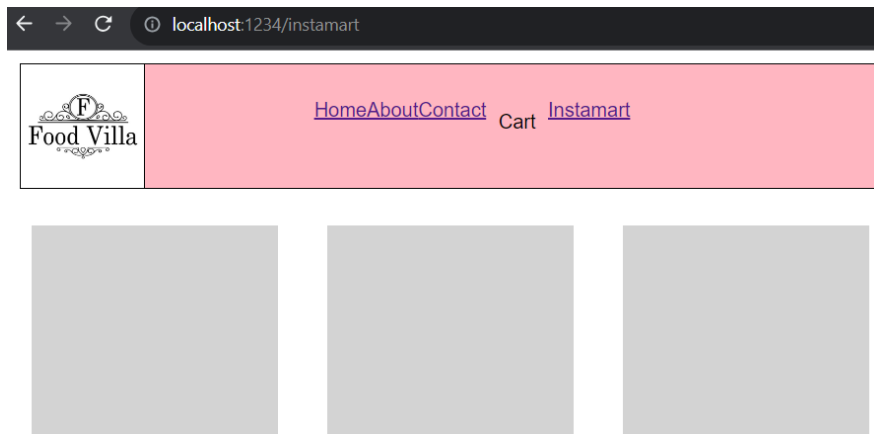
Import () returns a promise and react waits for the promise to get resolved when we wrap a component inside a suspense tag.

What is Fallback in react?

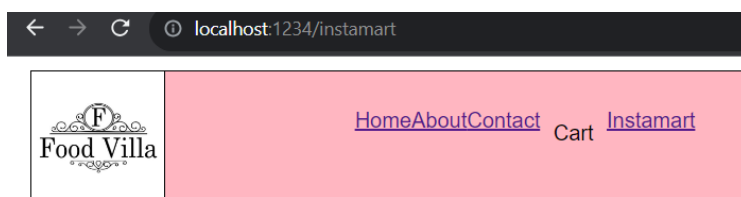
When bundle is not loaded react suspends the component for a while until the bundle is available. During this time for a better user experience we should display a shimmer UI on screen. So, suspense component accepts a prop called fallback where we call shimmer UI component. We can also write JSX in fallback attribute within {}.

```
{
  path: "/instamart",
  element: (
    <Suspense fallback={<ShimmerUI/>}>
      <Instamart />
    </Suspense>
  ),
},
```

Now if we navigate to the instamart page from home page.



When bundle is loaded-



Instamart

100 of components inside of it.

Footer

Why Should you not Lazy load a component inside a component or parent component?

Because if we do, for ever state or prop change, parent component will be rendered. As a result, lazy loading of the child component takes place on every render. So, every time the bundle will be loaded.