

Implementing Responsiveness in React

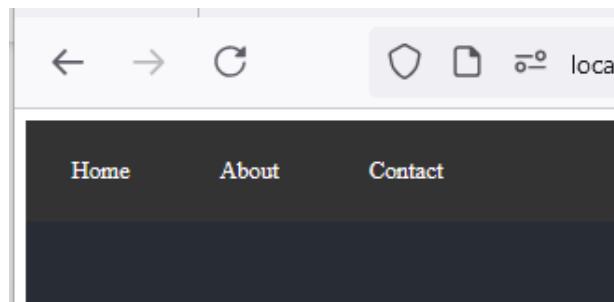
Introduction

Responsiveness is a part of any complete web application, we have to remember **mobile first**, and the importance of prioritizing layouts on smaller devices like cell phones. This is especially important in commercial applications in which effective rendering of products leads to a good user experience when accessing the site or web app on a mobile device. Hopefully a good user experience will lead to a sale of a product or service.


Traditionally, we integrate responsiveness by using key features of HTML/CSS, like Flexbox and media queries. In this tutorial, you will see how to use JSX to apply styles depending on the status of the application, and so enable responsiveness in terms of a hamburger menu. This is a small and specific example, however you can use the techniques presented to add your own extended strategy for responsive layouts.

1) The Requirement

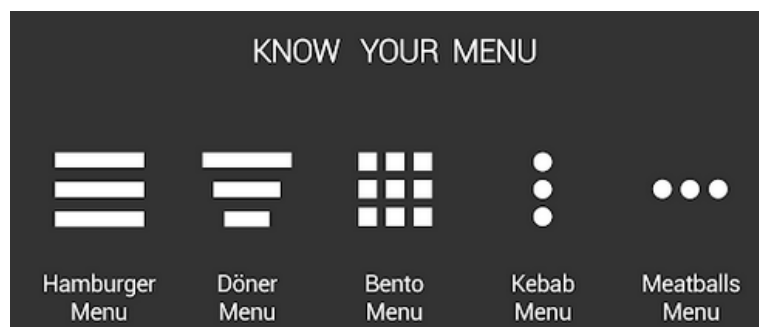
At higher breakpoints, like a full screen desktop, there is space to display menu options horizontally, spaced apart on a navigation bar.



The hamburger menu, the lines icon, should discreetly appear when the screen width is less. This icon should appear at lower breakpoints to save space on smaller screen widths.

There are many styles and icons available to represent the menu bar with a hamburger menu, current design philosophy is to mute this icon and tuck it into the user interface. 

Aside, there are specific references in web design that apply to menu bar icons that operate on an expand/contract basis.



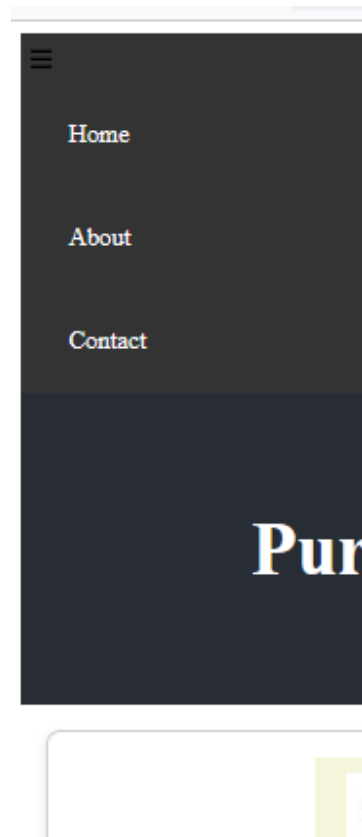
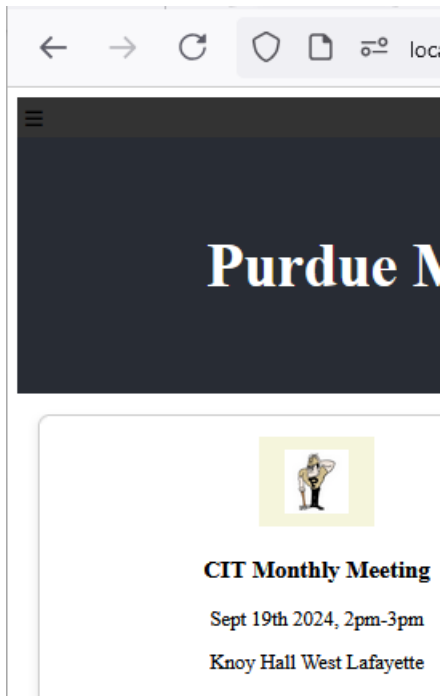
Implementing Responsiveness in React

The typical breakpoint at which the horizontal menu morphs into a hamburger icon? 768 pixels wide.

At this point, the user focus is entirely on the content of the current page and should be able to see that content most effectively without any unnecessary additions to the layout.

The hamburger menu is a small, recognized icon that expands when the user clicks it, and opens the menu bar vertically.

The vertical menu remains visible until the user clicks the hamburger menu again, in which case the options are hidden, leaving more space on the page.



Implementing Responsiveness in React

2) Potential Solution - React Hooks, CSS Style Rules

Let's start by modifying the Layout component in Layout.js. We will add a specific layout.css file, as this styling relates only to the layout component. Create this file, layout.css, and may as well keep it in the pages folder with Layout.js. Make sure to bring it into Layout.js, by importing it.

```
import "../Layout.css"; // Specific CSS file for styling
```

We need to ...

- Add a button to the menu bar to represent the hamburger menu
- Set the state of the button to open or closed, depending on the click status. Open should display the menu items and closed should hide them.
- Change the style of the nav bar items so that they display vertically at smaller breakpoints and horizontally at higher breakpoints
- Style the button to disappear at higher breakpoints and become visible at widths below our breakpoint of 768 pixels
- Dynamically modify the styling based on the click status

The Button

You can use the entity code **☰** for the hamburger menu icon, ☰. We will add this to the component, Layout.js.

In layout.css, create a style rule for the button.

```
nav button {  
  background: none;  
  border: none;  
  font-size: 24px;  
  cursor: pointer;  
  display: none; /* Hide the button by default */  
}
```

This styling renders a muted button that blends into the background, and reveals a cursor points when hovered over, indicating that it is clickable. Initially, this button is hidden.

Now let's add this button to Layout.js, inside the component.

Implementing Responsiveness in React

```
return (  
  <>  
    <nav>  
      <button>&#8801;</button>  
      <ul>  
        <li>  
          <link to="/">Home</link>  
        </li>  
      </ul>  
    </nav>  
  </>  
)
```

The State

When the hamburger menu is clicked, the menu options should be visible. Let's use the React `useState` hook to support this.

In `Layout.js`, import `useState`,

```
import React, { useState } from "react";
```

Now add the `useState` hook, with state variable `isOpen`. This hook (and all others) has to be inserted inside a component, so careful with placing the code.

```
const Layout = () => {  
  const [isOpen, setIsOpen] = useState(false);
```

Next step is to integrate the state with style rules that manipulate the navigation bar's menu items.

React hooks allow us to use JSX to set a state in the application depending on the status of a variable. We can use this to turn on and off style rules that apply to the navigation bar. Let's set the `isOpen` Boolean state variable to `true` initially, and then implement a toggle based on when it is clicked. This toggles associates or disassociates a class, `isOpen`, with the nav bar, depending on the status of the `isOpen` variable.

```
const Layout = () => {  
  const [isOpen, setIsOpen] = useState(false);  
  
  return (  
    <>  
      <nav className={isOpen ? "isOpen" : ""}>  
        <button onClick={() => setIsOpen(!isOpen)}>&#8801;</button>  
        <ul>  
          <li>  
            <link to="/">Home</link>  
          </li>  
        </ul>  
      </nav>  
    </>  
  )
```

Implementing Responsiveness in React

This can be used to selectively apply rules, which is exactly what we want, in order to implement the hamburger menu functionality.

The Style Rules

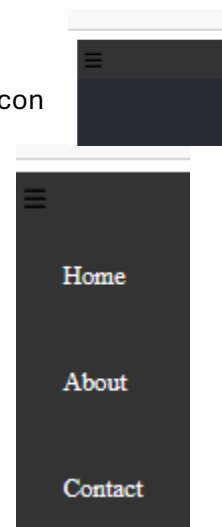
A Flex container is a good option for a menu bar that toggles between vertical and horizontal orientation, so let's add some additional style rules for the nav tag, the ul tag and the li tags – as these are specific to the layout, we will add them to layout.css.

<pre>nav ul { list-style-type: none; padding: 0; margin: 0; display: flex; /* Display is row by default */ }</pre>	Flex containers default to row for flex-direction
<pre>nav li { margin: 10px; }</pre>	You can change this to suit your font and spacing requirements
<pre>.isOpen ul { display: flex; /* Ensure menu displays in row */ }</pre>	Here, we use this isOpen style with the isOpen state variable, modified by media query

We need a media query to handle smaller breakpoints.

The media query supports the smaller breakpoints by revealing the hamburger icon button ...

and setting the flex direction of the navigation bar's ul tag to column instead of row. The list items are flex items, so these will render vertically, in a column.



Implementing Responsiveness in React

The display property is used here as a reset, hiding or showing the ul element, to hide or show the menu bar options as these are the list items.

```
/* Media query for screens smaller than 768px */
@media screen and (max-width: 768px) {
  nav button {
    display: block; /* Show the button on smaller screens */
  }

  nav ul {
    display: none; /* Hide the menu by default on smaller screens */
    flex-direction: column; /* Stack items vertically */
  }

  .isOpen nav ul {
    display: flex; /* Show the menu when isOpen is true */
    flex-direction: column; /* Ensure items stack vertically when open */
  }
}
```

Now this should render a responsive menu bar!