Layouts

▼ **Examples**

- layout-component

The React model allows us to deconstruct a page into a series of components. Many of these components are often reused between pages. For example, you might have the same navigation bar and footer on every page.

```
// components/layout.js

import Navbar from './navbar'
import Footer from './footer'

export default function Layout({ children }) {
  return (
    <>
      <Navbar />
      <main>{children}</main>
      <Footer />
    </>
  )
}
```

Examples

Single Shared Layout with Custom App

If you only have one layout for your entire application, you can create a Custom App and wrap your application with the layout. Since the `<Layout />` component is re-used when changing pages, its component state will be preserved (e.g. input values).

```
// pages/_app.js

import Layout from '../components/layout'

export default function MyApp({ Component, pageProps }) {
  return (
    <Layout>
      <Component {...pageProps} />
    </Layout>
  )
}
```

Per-Page Layouts

If you need multiple layouts, you can add a property `getLayout` to your page, allowing you to return a React component for the layout. This allows you to define the layout on a *per-page basis*. Since we're returning a function, we can have complex nested layouts if desired.

```
// pages/index.js

import Layout from '../components/layout'
import NestedLayout from '../components/nested-layout'

export default function Page() {
  return {
    /** Your content */
  }
}

Page.getLayout = function getLayout(page) {
  return (
    <Layout>
      <NestedLayout>{page}</NestedLayout>
    </Layout>
  )
}
```

```
// pages/_app.js

export default function MyApp({ Component, pageProps }) {
  // Use the layout defined at the page level, if available
  const getLayout = Component.getLayout || ((page) => page)

  return getLayout(<Component {...pageProps} />)
}
```

When navigating between pages, we want to *persist* page state (input values, scroll position, etc.) for a Single-Page Application (SPA) experience.

This layout pattern enables state persistence because the React component tree is maintained between page transitions. With the component tree, React can understand which elements have changed to preserve state.

> ***Note**: This process is called reconciliation, which is how React understands which elements have changed.*

With TypeScript

When using TypeScript, you must first create a new type for your pages which includes a `getLayout` function. Then, you must create a new type for your `AppProps` which overrides the `Component` property to use the previously created type.

```
// pages/index.tsx

import type { ReactElement } from 'react'
import Layout from '../components/layout'
import NestedLayout from '../components/nested-layout'

export default function Page() {
  return {
```

```
    /** Your content */
  }
}

Page.getLayout = function getLayout(page: ReactElement) {
  return (
    <Layout>
      <NestedLayout>{page}</NestedLayout>
    </Layout>
  )
}
```

```
// pages/_app.tsx

import type { ReactElement, ReactNode } from 'react'
import type { NextPage } from 'next'
import type { AppProps } from 'next/app'

type NextPageWithLayout = NextPage & {
  getLayout?: (page: ReactElement) => ReactNode
}

type AppPropsWithLayout = AppProps & {
  Component: NextPageWithLayout
}

export default function MyApp({ Component, pageProps }: AppPropsWithLayout) {
  // Use the layout defined at the page level, if available
  const getLayout = Component.getLayout ?? ((page) => page)

  return getLayout(<Component {...pageProps} />)
}
```

Data Fetching

Inside your layout, you can fetch data on the client-side using `useEffect` or a library like SWR. Because this file is not a Page, you cannot use `getStaticProps` or `getServerSideProps` currently.

```
// components/layout.js

import useSWR from 'swr'
import Navbar from './navbar'
import Footer from './footer'

export default function Layout({ children }) {
  const { data, error } = useSWR('/api/navigation', fetcher)

  if (error) return <div>Failed to load</div>
  if (!data) return <div>Loading...</div>
```

```
  return (
    <>
      <Navbar links={data.links} />
      <main>{children}</main>
      <Footer />
    </>
  )
}
```

For more information on what to do next, we recommend the following sections:

Pages: Learn more about what pages are in Next.js.
Custom App: Learn more about how Next.js initialize pages.