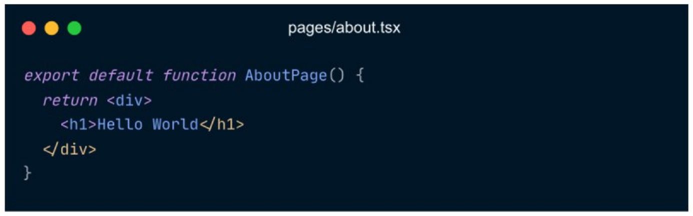
NEXT.JS

pages

- **pages** have been moved to the **app** dir.
- Use **folders** to define **routes**.
- Use a **page.tsx** inside a folder to create UI for the page.

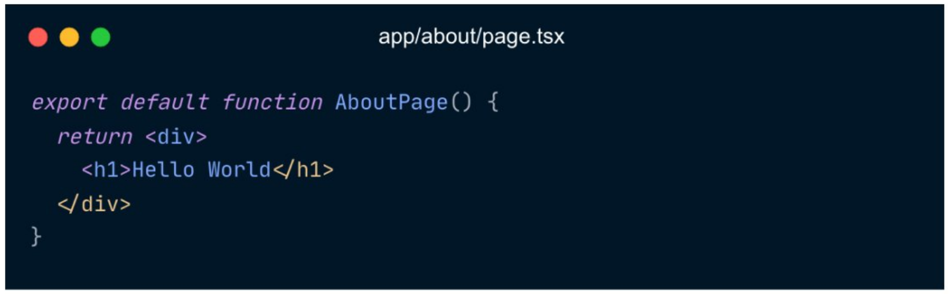
NEXT.js 12



pages/about.tsx

```tsx
export default function AboutPage() {
  return <div>
    <h1>Hello World</h1>
  </div>
}
```

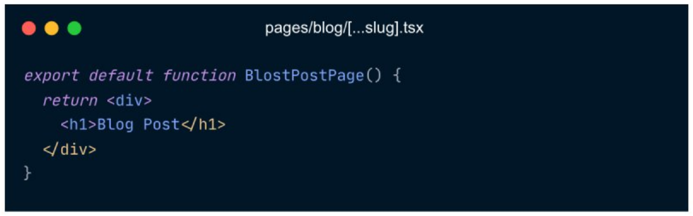
NEXT.js 13



app/about/page.tsx

```tsx
export default function AboutPage() {
  return <div>
    <h1>Hello World</h1>
  </div>
}
```

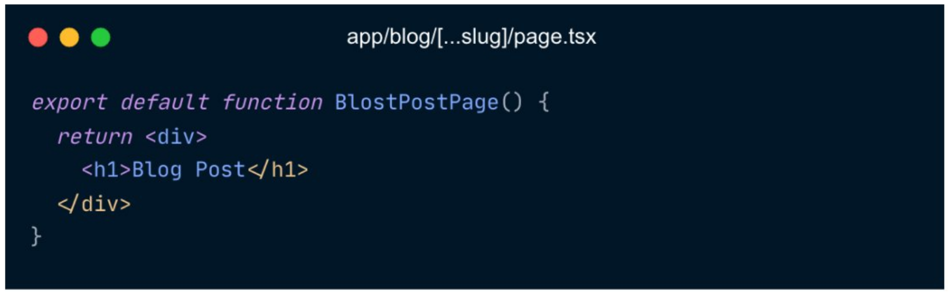
NEXT.js 12



pages/blog/[...slug].tsx

```tsx
export default function BlostPostPage() {
  return <div>
    <h1>Blog Post</h1>
  </div>
}
```

NEXT.js 13



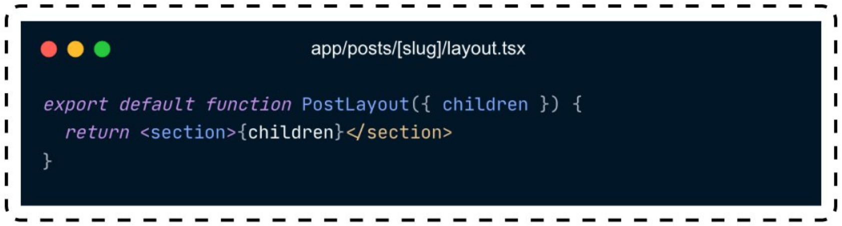
app/blog/[...slug]/page.tsx

```tsx
export default function BlostPostPage() {
  return <div>
    <h1>Blog Post</h1>
  </div>
}
```

Creating UI using files

- In Next.js 13, **folders** are for **routes** and **files** are for UI.
- Special files: **layout.tsx**, **page.tsx**, **loading.tsx**, **error.tsx**, **head.tsx**

NEXT.js 13

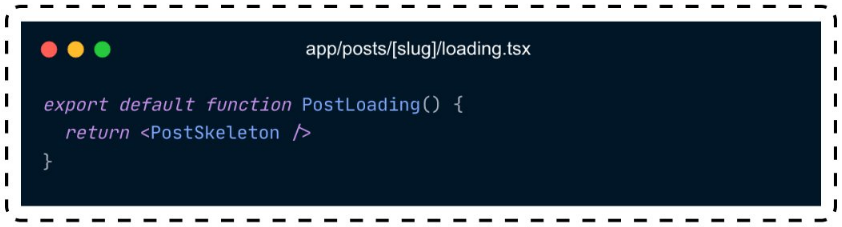


app/posts/[slug]/layout.tsx

```tsx
export default function PostLayout({ children }) {
  return <section>{children}</section>
}
```

- Use **layout.tsx** to show wrapper components.
- You can fetch data inside **layout.tsx**.

NEXT.js 13

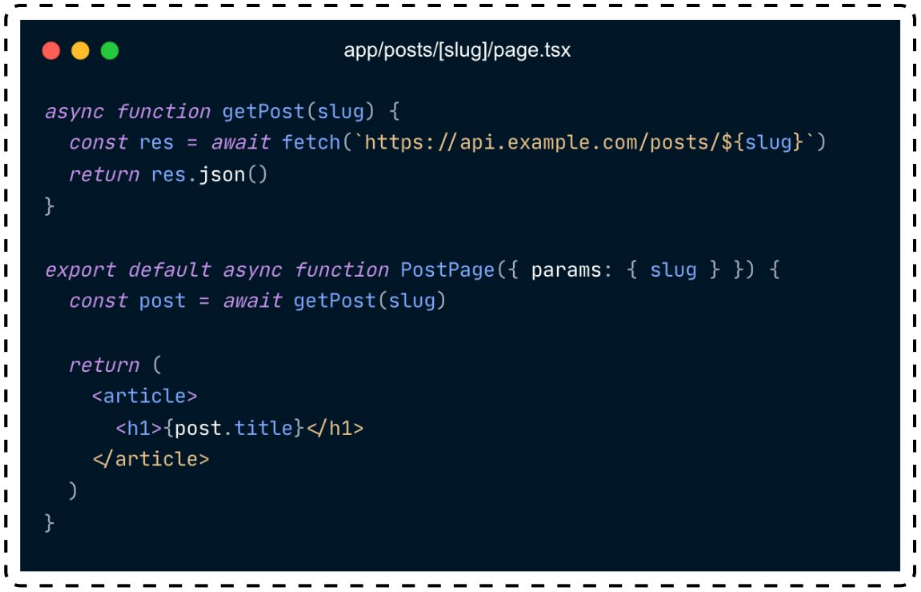


app/posts/[slug]/loading.tsx

```tsx
export default function PostLoading() {
  return <PostSkeleton />
}
```

- Use **loading.tsx** for skeletons and spinners during routing.

NEXT.js 13



app/posts/[slug]/page.tsx

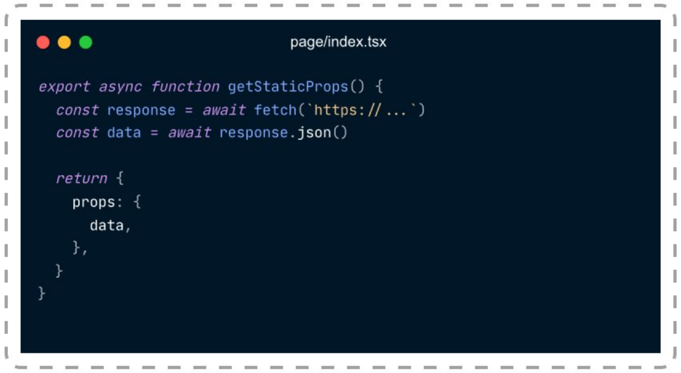
```tsx
async function getPost(slug) {
  const res = await fetch(`https://api.example.com/posts/${slug}`)
  return res.json()
}

export default async function PostPage({ params: { slug } }) {
  const post = await getPost(slug)

  return (
    <article>
      <h1>{post.title}</h1>
    </article>
  )
}
```

- Use **page.tsx** to show page components.
- You can fetch data inside **page.tsx**.
- Note: The default export is an **async** function when fetching data.

getStaticProps

- Extract your data fetching into a custom function.
- Call your data fetcher directly inside your components.
- Data returned by fetchData will be cached until manually revalidated.

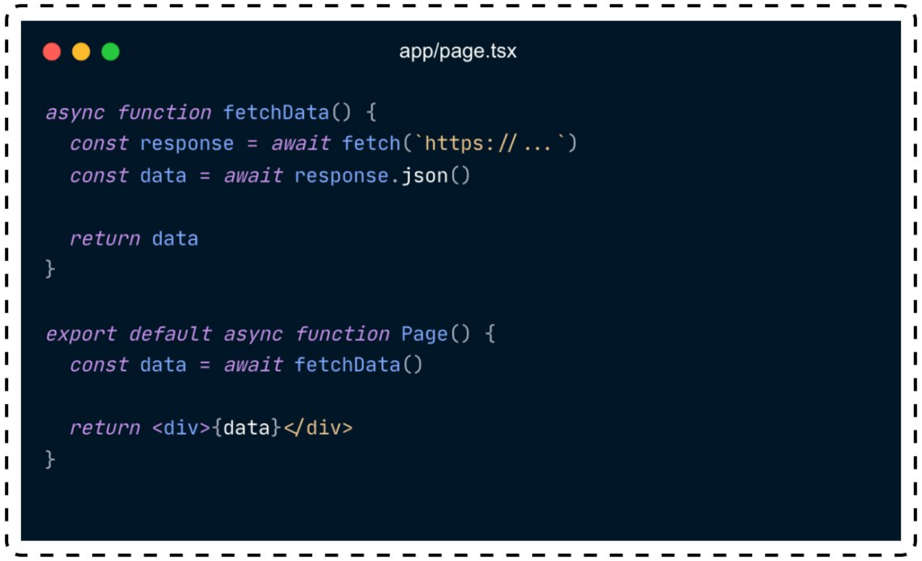
NEXT.js 12



page/index.tsx

```
export async function getStaticProps() {
  const response = await fetch(`https:// ...`)
  const data = await response.json()

  return {
    props: {
      data,
    },
  }
}
```

NEXT.js 13



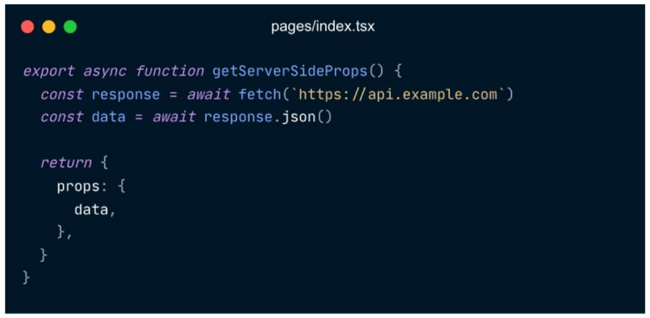
app/page.tsx

```
async function fetchData() {
  const response = await fetch(`https:// ...`)
  const data = await response.json()

  return data
}

export default async function Page() {
  const data = await fetchData()

  return <div>{data}</div>
}
```

getServerSideProps

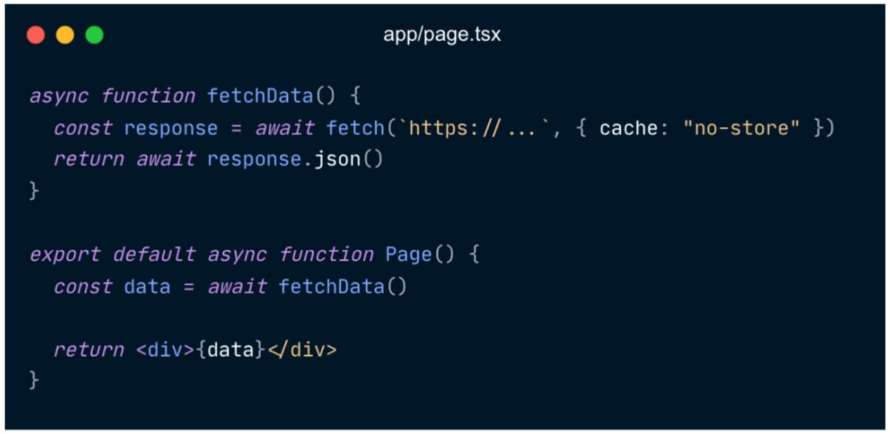
- Extract your data fetching into a custom function.
- Call your data fetcher directly inside your components.
- Use **{ cache: “no-store” }** in **fetch**.
- **{ cache: “no-store” }** tells fetch to never cache this requests. Rerun it again every time I ask for this page. Similar to gSSP.

NEXT.js 12


pages/index.tsx

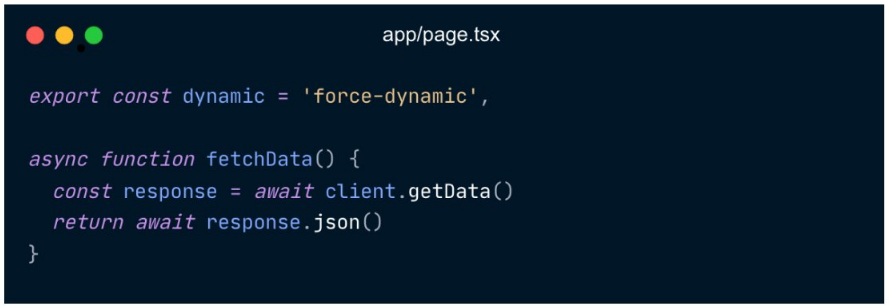
```
export async function getServerSideProps() {
  const response = await fetch(`https://api.example.com`)
  const data = await response.json()

  return {
    props: {
      data,
    },
  }
}
```

NEXT.js 13


app/page.tsx

```
async function fetchData() {
  const response = await fetch(`https://...`, { cache: "no-store" })
  return await response.json()
}

export default async function Page() {
  const data = await fetchData()

  return <div>{data}</div>
}
```

NEXT.js 13


app/page.tsx

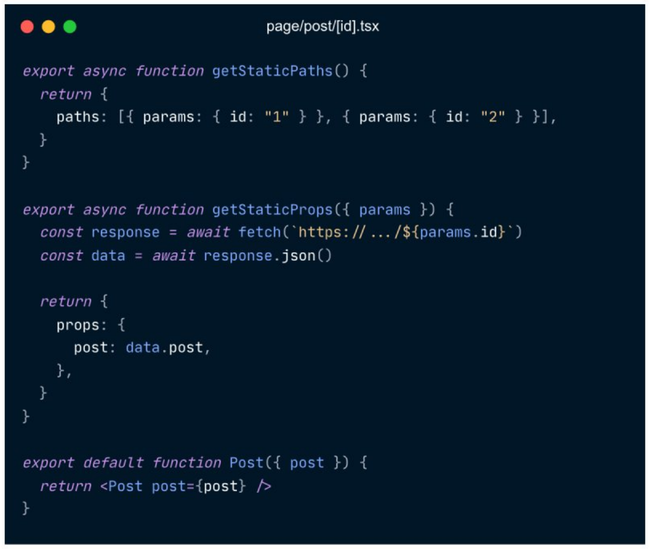
```
export const dynamic = 'force-dynamic',

async function fetchData() {
  const response = await client.getData()
  return await response.json()
}
```

- If you’re using a custom API client or cannot use fetch, you can use the **dynamic** segment option.

getStaticPaths

- **getStaticPaths** renamed to **generateStaticParams**
- The paths.params key is gone.

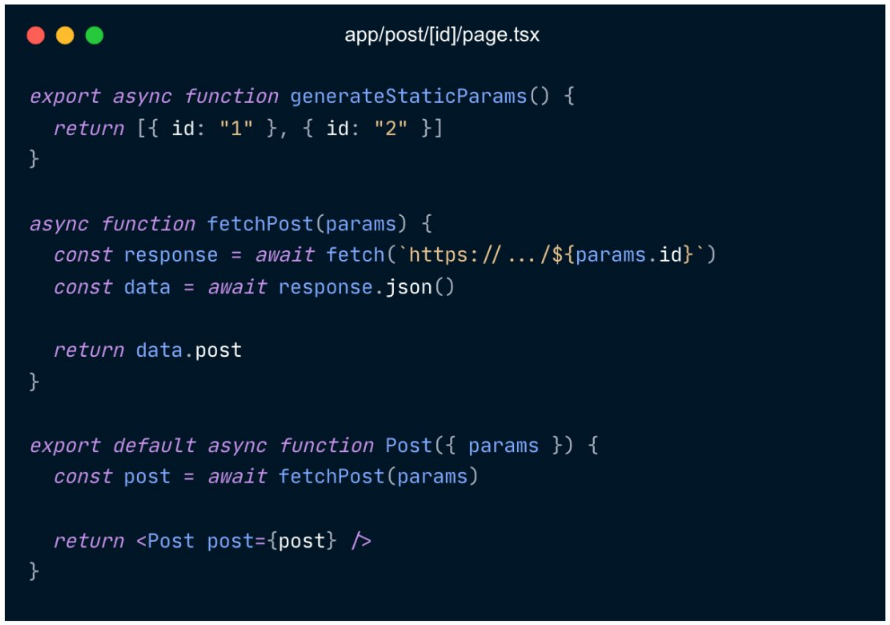
NEXT.js 12



page/post/[id].tsx

```
export async function getStaticPaths() {
  return {
    paths: [{ params: { id: "1" } }, { params: { id: "2" } }],
  }
}

export async function getStaticProps({ params }) {
  const response = await fetch(`https://.../${params.id}`)
  const data = await response.json()

  return {
    props: {
      post: data.post,
    },
  }
}

export default function Post({ post }) {
  return <Post post={post} />
}
```

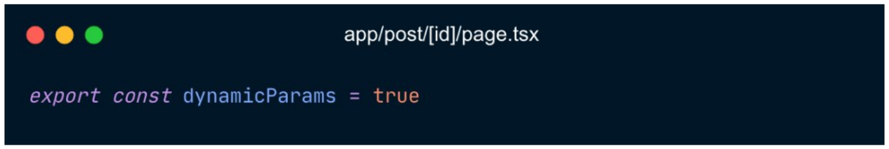
NEXT.js 13



app/post/[id]/page.tsx

```
export async function generateStaticParams() {
  return [{ id: "1" }, { id: "2" }]
}

async function fetchPost(params) {
  const response = await fetch(`https://.../${params.id}`)
  const data = await response.json()

  return data.post
}

export default async function Post({ params }) {
  const post = await fetchPost(params)

  return <Post post={post} />
}
```

NEXT.js 13



app/post/[id]/page.tsx

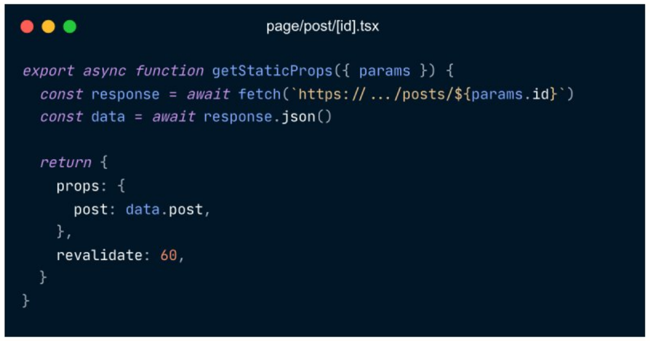
```
export const dynamicParams = true
```

- **fallback** replaced by **dynamicParams**

Incremental Static Regeneration

- Use **{ next: { revalidate: number } }** in **fetch**

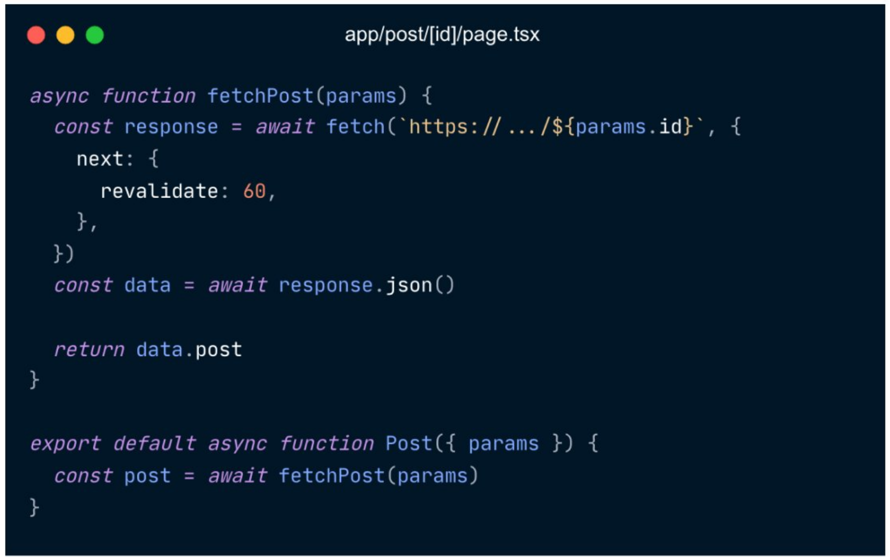
NEXT.js 12



page/post/[id].tsx

```
export async function getStaticProps({ params }) {
  const response = await fetch(`https://.../posts/${params.id}`)
  const data = await response.json()

  return {
    props: {
      post: data.post,
    },
    revalidate: 60,
  }
}
```

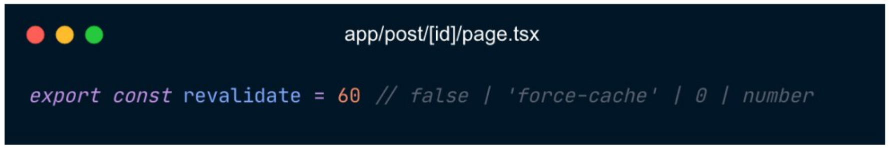
NEXT.js 13



app/post/[id]/page.tsx

```
async function fetchPost(params) {
  const response = await fetch(`https://.../${params.id}`, {
    next: {
      revalidate: 60,
    },
  })
  const data = await response.json()

  return data.post
}

export default async function Post({ params }) {
  const post = await fetchPost(params)
}
```

NEXT.js 13



app/post/[id]/page.tsx

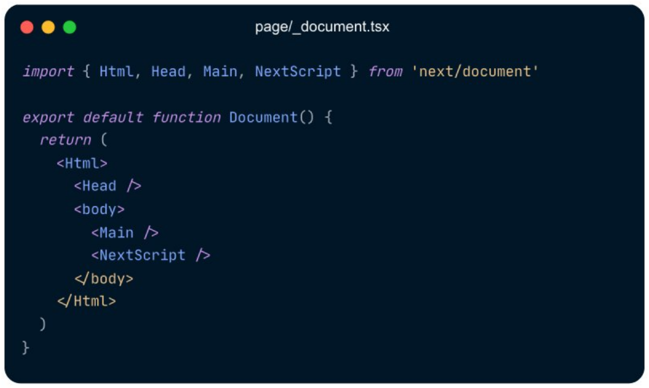
```
export const revalidate = 60 // false | 'force-cache' | 0 | number
```

- If you're using a custom API client or cannot use fetch, you can use the **revalidate** segment option.

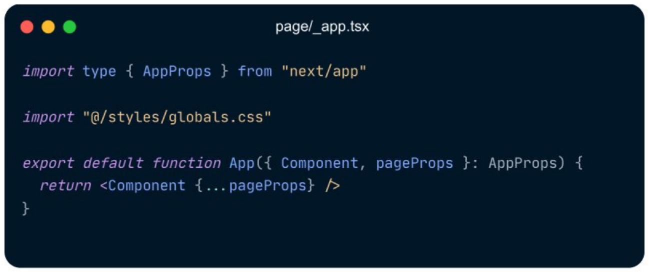
_document.tsx and _app.tsx

- Move **_document.tsx** and **_app.tsx** to root layout at **app/layout.tsx**.
- Add **<html>** and **<body>** tags. **<head>** is optional.
- Import your global stylesheet in the root layout.

NEXT.JS 12

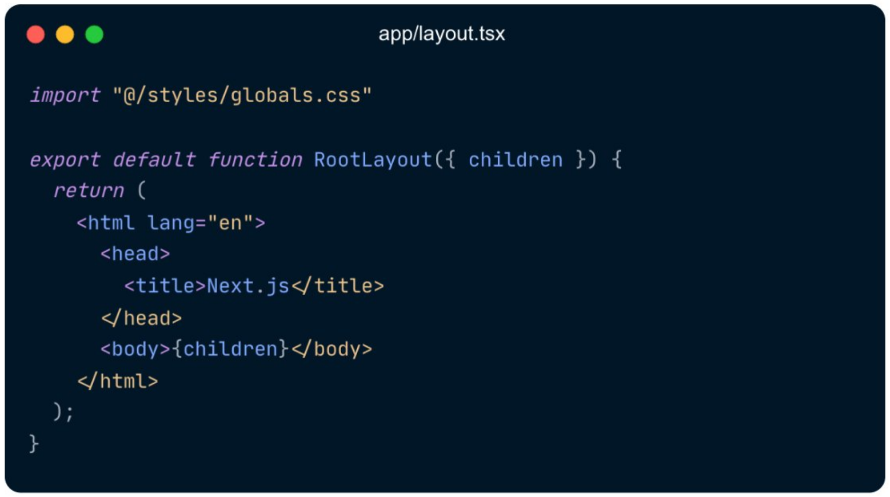

page/_document.tsx

```
import { Html, Head, Main, NextScript } from 'next/document'

export default function Document() {
  return (
    <Html>
      <Head />
      <body>
        <Main />
        <NextScript />
      </body>
    </Html>
  )
}
```


page/_app.tsx

```
import type { AppProps } from "next/app"

import "@/styles/globals.css"

export default function App({ Component, pageProps }: AppProps) {
  return <Component {...pageProps} />
}
```

NEXT.JS 13


app/layout.tsx

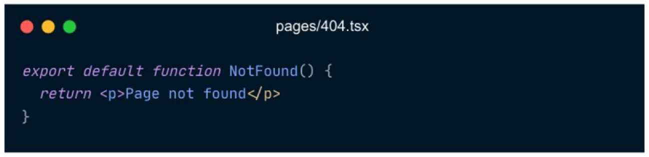
```
import "@/styles/globals.css"

export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <head>
        <title>Next.js</title>
      </head>
      <body>{children}</body>
    </html>
  );
}
```

- The root layout is required. It wraps all your pages.
- You can have multiple root layouts using layout groups.

page/404.tsx

- The **404.tsx** is now replaced with the **not-found.tsx** file.
- To show a 404, return **notFound()** inside your **page.tsx**.

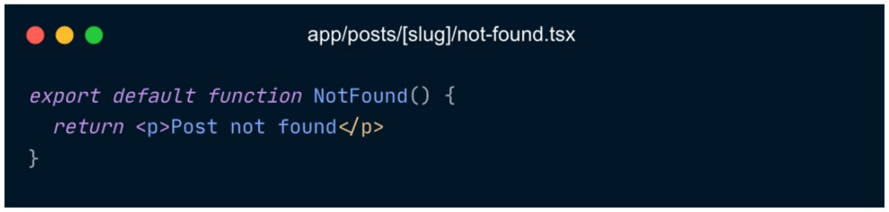
NEXT.js 12



pages/404.tsx

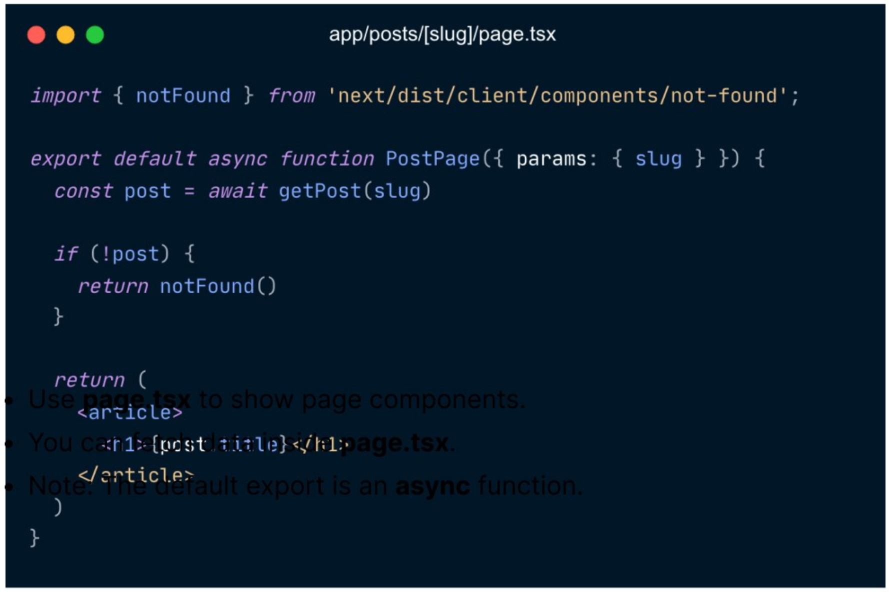
```
export default function NotFound() {
  return <p>Page not found</p>
}
```

NEXT.js 13



app/posts/[slug]/not-found.tsx

```
export default function NotFound() {
  return <p>Post not found</p>
}
```



app/posts/[slug]/page.tsx

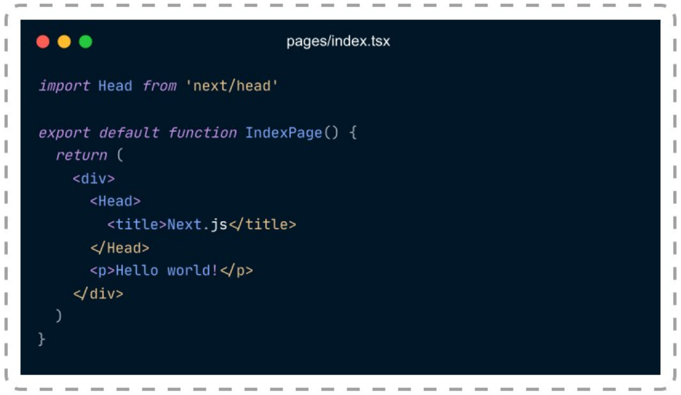
```
import { notFound } from 'next/dist/client/components/not-found';

export default async function PostPage({ params: { slug } }) {
  const post = await getPost(slug)

  if (!post) {
    return notFound()
  }

  return (
    <article>
      <h1>{post.title}</h1>
    </article>
  )
}
```

- Use **page.tsx** to show page components.
- You can fetch data inside **page.tsx**.
- Note: The default export is an **async** function.

next/head

- **next/head** is now a special **head.tsx** file inside your routes.
- **head.tsx** is a server component. You can fetch data inside **head.tsx**.

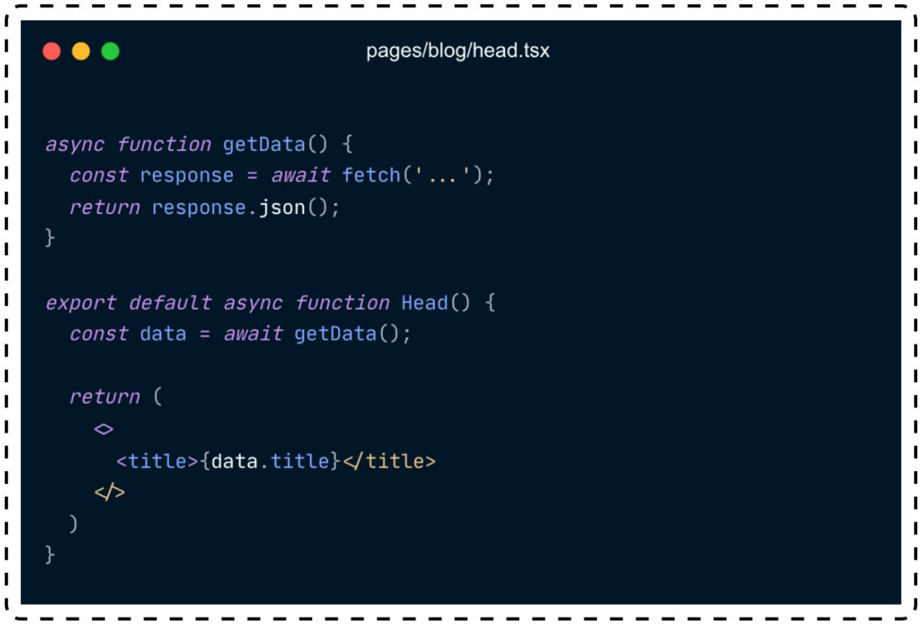
NEXT.JS 12



pages/index.tsx

```tsx
import Head from 'next/head'

export default function IndexPage() {
  return (
    <div>
      <Head>
        <title>Next.js</title>
      </Head>
      <p>Hello world!</p>
    </div>
  )
}
```

NEXT.JS 13



pages/blog/head.tsx

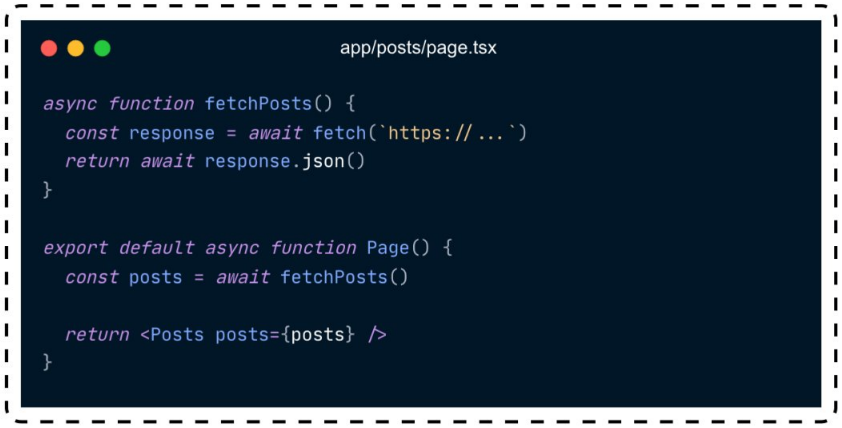
```tsx
async function getData() {
  const response = await fetch('...');
  return response.json();
}

export default async function Head() {
  const data = await getData();

  return (
    <>
      <title>{data.title}</title>
    </>
  )
}
```

- Allowed tags in head.tsx: **<title>, <meta>, <link> and <script>**

Server Components

- Renders on the server / An async function when fetching data
- Use to fetch data / has access to database and node.
- Secure / Can use API keys
- Cannot use browser APIs
- Cannot use useState, useEffect..etc hooks
- No event listeneres (onClick, onSubmit...etc)

NEXT.JS 13

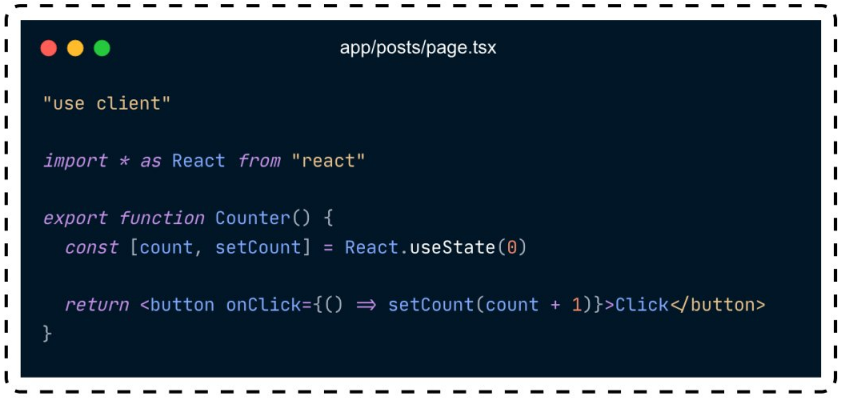

app/posts/page.tsx

```tsx
async function fetchPosts() {
  const response = await fetch(`https://...`)
  return await response.json()
}

export default async function Page() {
  const posts = await fetchPosts()

  return <Posts posts={posts} />
}
```

Client Components

- Renders on the client / Is **NOT** an async function
- Denoted by the “use client” directive
- Cannot use API keys
- Can use browser APIs
- Can use useState, useEffect..etc hooks
- Can use event listeneres (onClick, onSubmit...etc)

NEXT.JS 13

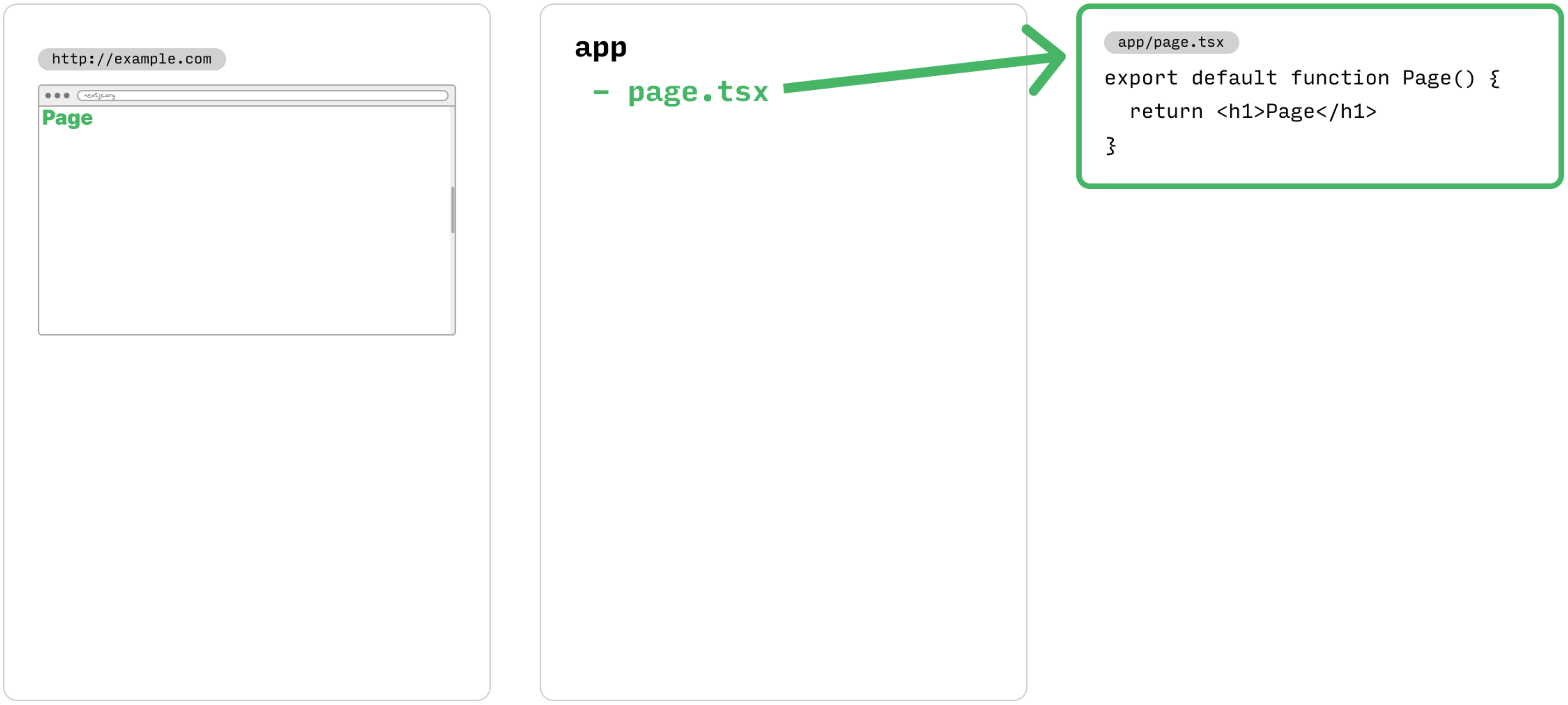

app/posts/page.tsx

```tsx
"use client"

import * as React from "react"

export function Counter() {
  const [count, setCount] = React.useState(0)

  return <button onClick={() => setCount(count + 1)}>Click</button>
}
```

👉 If you have to fetch data or use Node.js APIs, use a server component.

👉 If you have to use **useState/useEffect** or listen to **onClick/onSubmit**, use a client component.

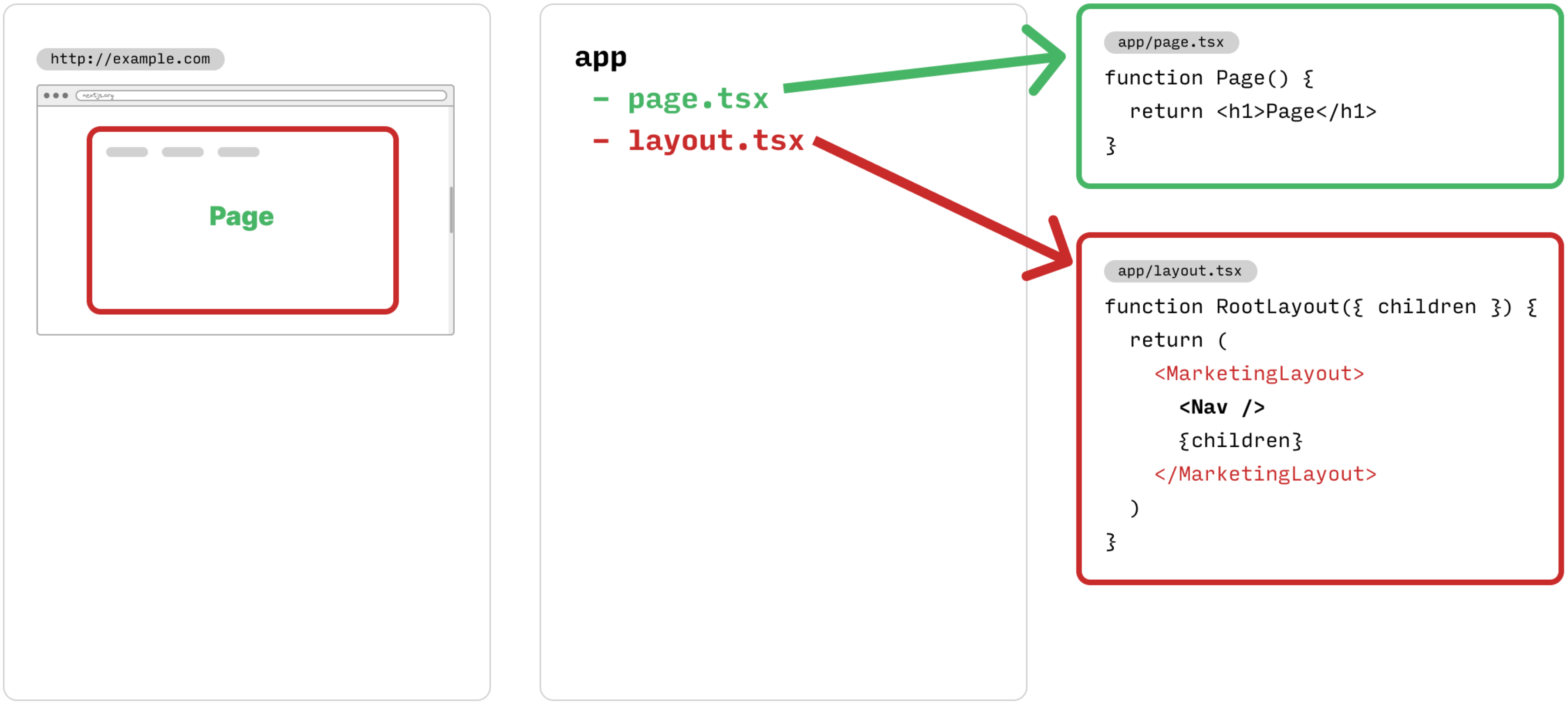
1. This is a page



```
export default function Page() {
  return <h1>Page</h1>
}
```

2. This is a page with a layout

page is wrapped in the root layout

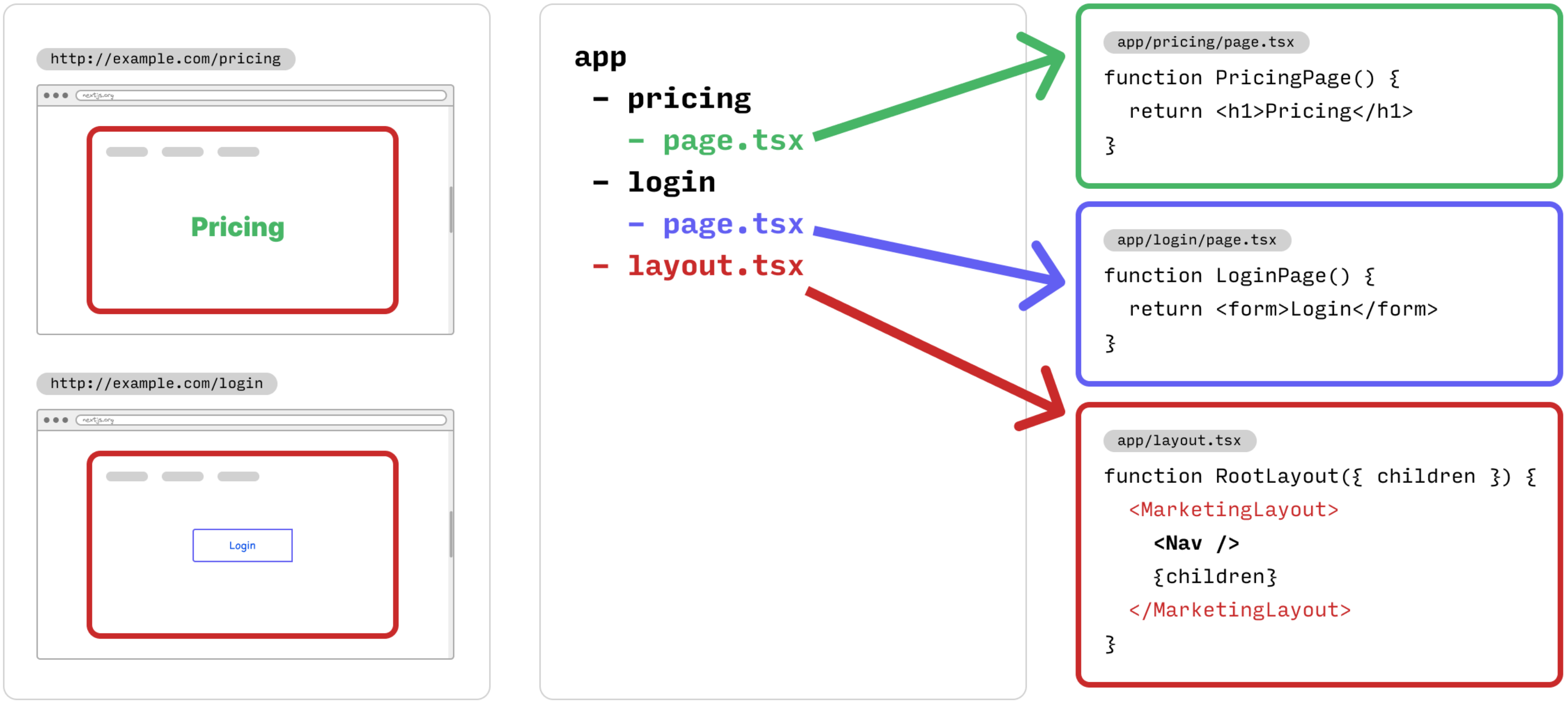


```
function Page() {
  return <h1>Page</h1>
}
```

```
function RootLayout({ children }) {
  return (
    <MarketingLayout>
      <Nav />
      {children}
    </MarketingLayout>
  )
}
```

3. Two pages sharing a layout

/pricing and /login are both wrapped in the root layout (red box)



```
app
 - pricing
   - page.tsx
 - login
   - page.tsx
 - layout.tsx
```

app/pricing/page.tsx

```
function PricingPage() {
  return <h1>Pricing</h1>
}
```

app/login/page.tsx

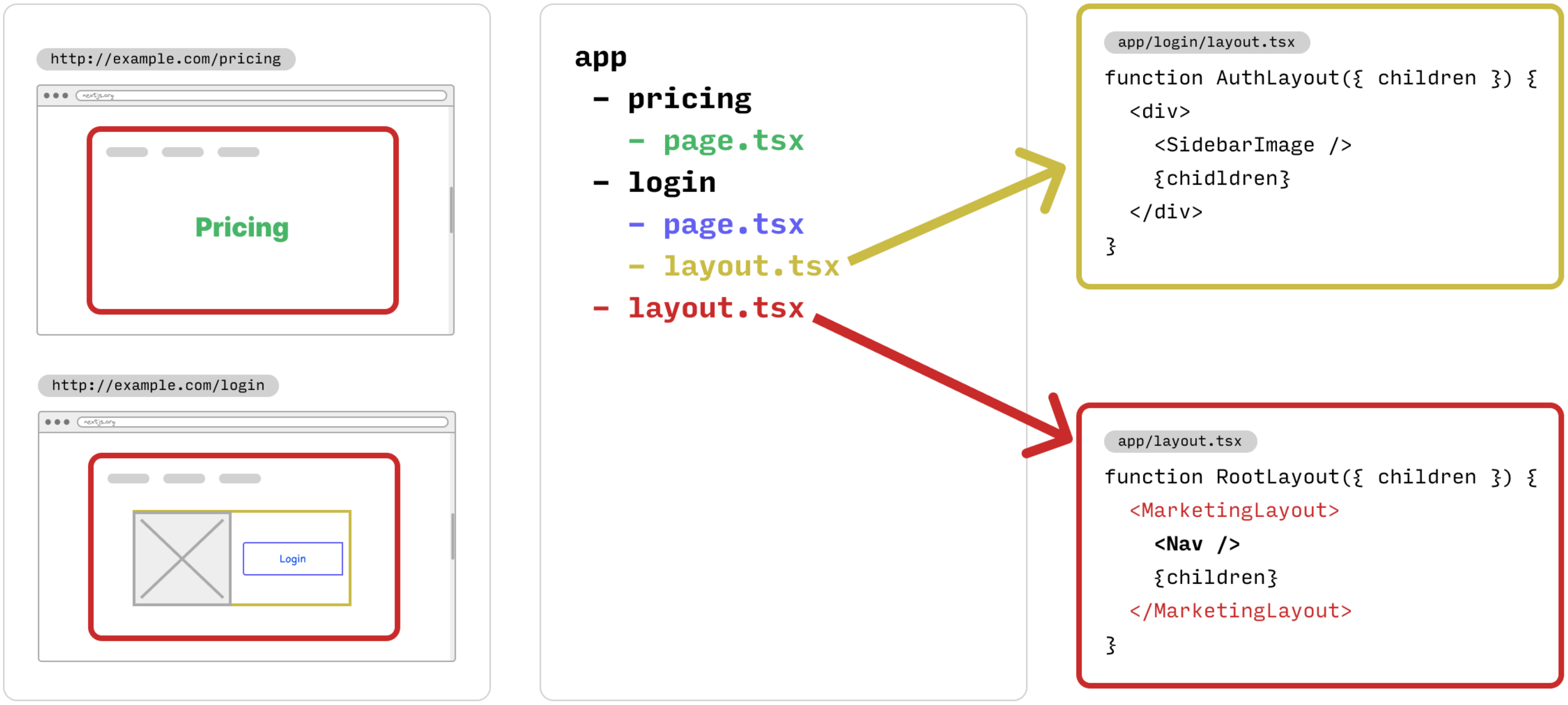
```
function LoginPage() {
  return <form>Login</form>
}
```

app/layout.tsx

```
function RootLayout({ children }) {
  <MarketingLayout>
    <Nav />
    {children}
  </MarketingLayout>
}
```

4. Root Layout and Nested Layouts

/login uses a nested “Auth” layout and is then wrapped in the root layout



```
app
 - pricing
   - page.tsx
 - login
   - page.tsx
   - layout.tsx
 - layout.tsx
```

app/login/layout.tsx

```
function AuthLayout({ children }) {
  <div>
    <SidebarImage />
    {chidldren}
  </div>
}
```

app/layout.tsx

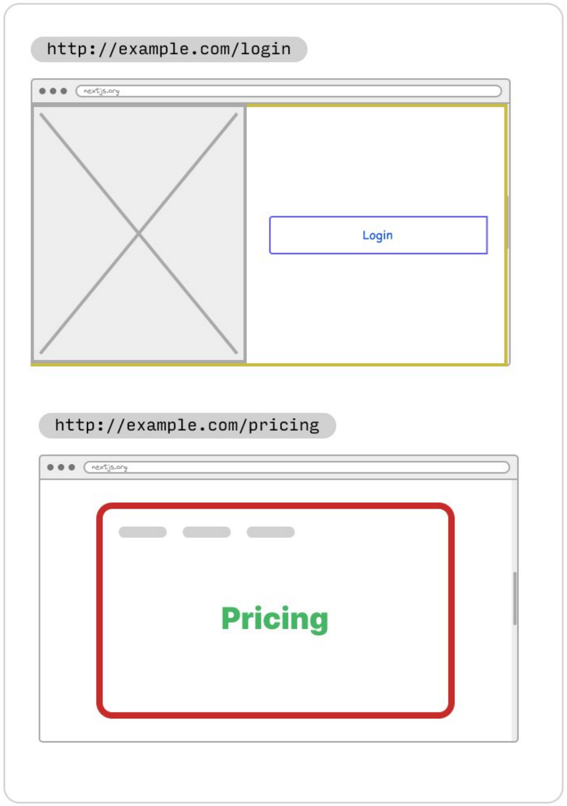
```
function RootLayout({ children }) {
  <MarketingLayout>
    <Nav />
    {children}
  </MarketingLayout>
}
```

We created two sections:
Auth and Marketing

All pages placed in a section share the same layout

5. Grouped Layouts

Two root layouts, one for wrapping marketing pages and one for auth pages



```
app
 - (auth)
   - layout.tsx
   - login
    - page.tsx
 - (marketing)
   - layout.tsx
   - pricing
     - page.tsx
 - layout.tsx
```

app/(auth)/layout.tsx

```
function AuthLayout({ children }) {
  <div>
    <SidebarImage />
    {chidldren}
  </div>
}
```

app/(marketing)/layout.tsx

```
function RootLayout({ children }) {
  <MarketingLayout>
    <Nav />
    {children}
  </MarketingLayout>
}
```