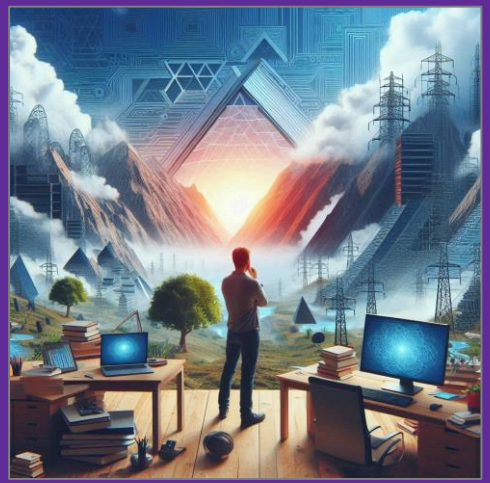
NEXT.JS

Turn-key Fullstack React



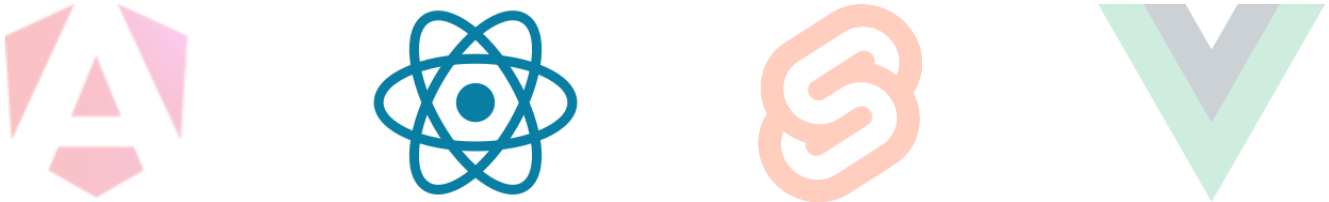
~~Homies~~ Friends

"~~Dude~~, I've got a brilliant product idea..."

~~Peepz~~

Let’s Just Set Up the Dev Project…

Tech choice: Frontend framework



Tech choice: Build Tool

...

...

Grunt

Vite

Snowpack

Webpack

Broccoli

Gulp

Rollup

Browserify

Babel

Parcel

...

Tech choice: Routing

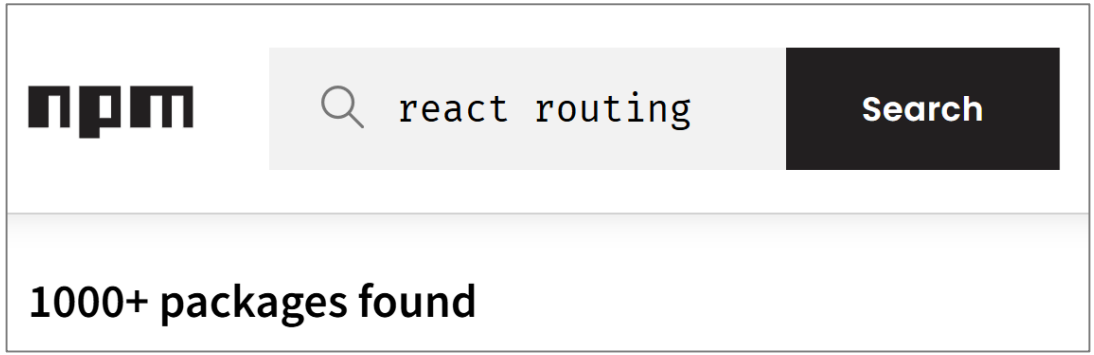
React Navigation

React Router

Reach Router

Wouter

...



Sebastian Nilsson

Northbridge

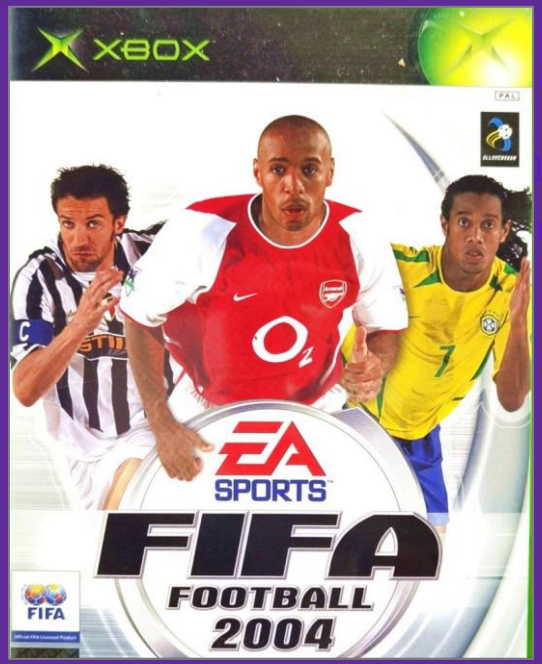
Tech Leader Consultant at Northbridge

- 20+ years in tech
  - Software engineering
  - Leadership
- Focus on Microsoft-ecosystem
  - .NET
  - Azure
- Lots of web technologies
  - TypeScript
  - React

Sebastian Nilsson

Northbridge

Game Credits - Audio & Support:
FIFA 2004



More tech choices

- **Package manager**: npm, yarn, pnpm
- **Running TypeScript**: ts-node, Webpa
- **CSS handling**: import .css, inline CSS
- **Testing**: Jest, Mocha, Jasmine, Karma
- **Data fetching**: Fetch API, Axiom, Supe
- **Developer Experience**: Fast Refresh,
- **Server side rendering**
- **Bundling/Minification**
- **Image optimization**
- **Font handling**
- **…**

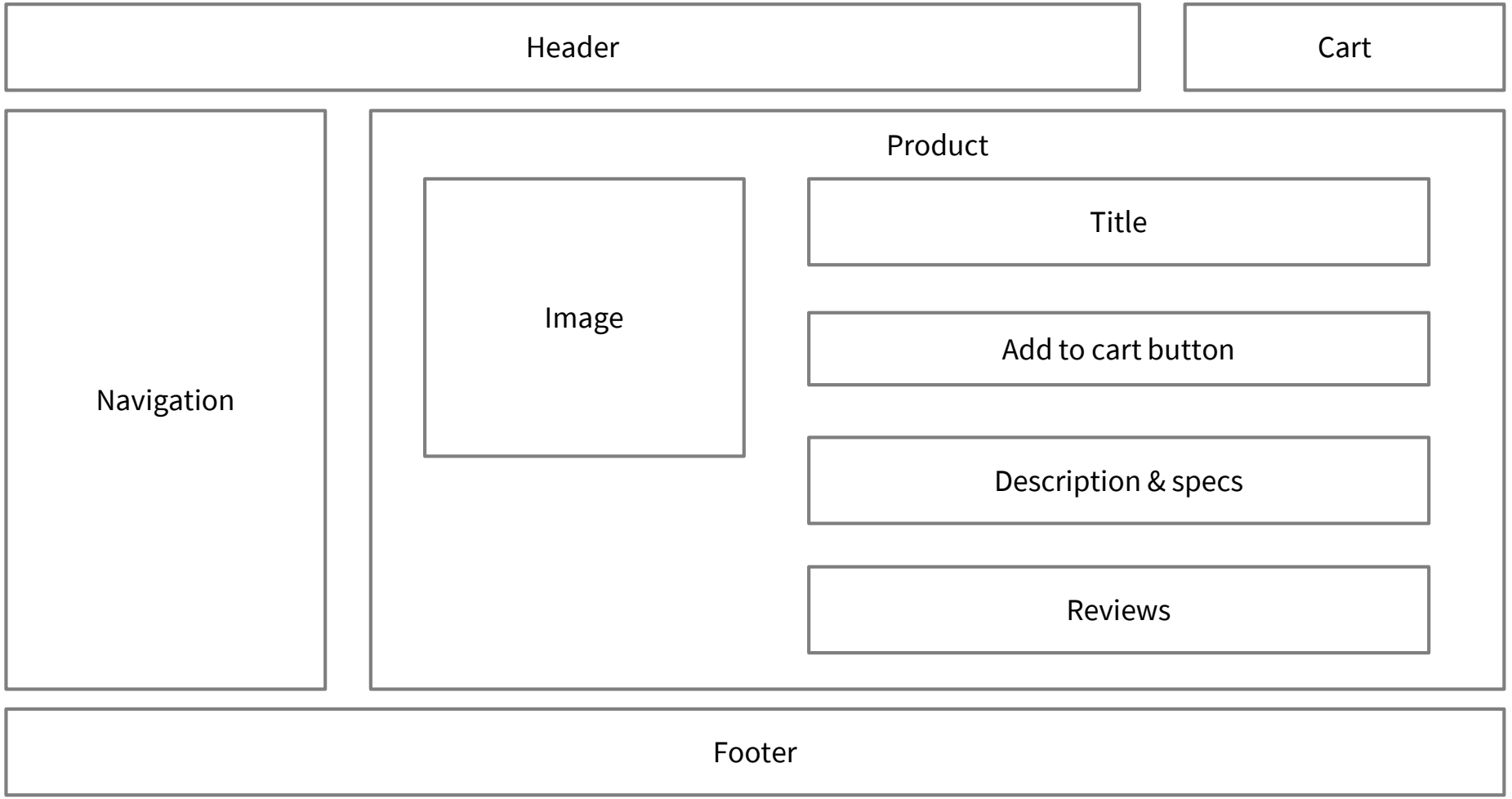
Let’s imagine that brilliant product

Product:

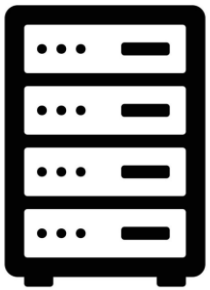
E-com website

Requirements:

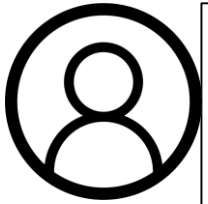
- Server-side rendering
  - Performance
  - SEO
  - Logged in/out UX
- Need interactive components
  - Add to cart
  - Add comments
  - ♥ Cookie consent
- Modern web best practices


Header
Cart
Navigation
Product
Image
Title
Add to cart button
Description & specs
Reviews
Footer

Classic React SPA (Single page app)



Server



```
<html>
<head><!-- ... --></head>
<body>

<div id="app"></div>

<script src="/react.js">
<script src="/app.js">
</body>
```

SPA downsides

- SEO challenges
- Initial load time
- Janky state restore
- Linking and navigation
- Janky auth experience
- Security considerations
- Analytics & tracking
- JavaScript disabled clients

React JSX -- SPA or SSR

```
<Product>
  <ProductImage />
  <ProductDescription />
  {
    product.ratings.length > 0 ?
    <ProductRating /> : <></>
  }
  <ProductComments />
</Product>

<SeeMoreProducts />
```

</>

```
<div id="product">
  <img src="product.jpg" />
  <!-- Description ... →

  <!-- Ratings skipped -->

  <div id="product-comments">
    <!-- ... -->
  </div>
</div>
```

Browsers run React/JavaScript predictably

How do we make the server run React/JavaScript predictably?

That brings us to Next.js

Next.js key features

- **React 19 support**
- **File based routing**
- **Server-side rendering (SSR)**
- **Incremental Static Regeneration (ISR)**
- **HTML streaming:** Send content ASAP
- **Route handlers:** API endpoints
- **Server actions:** Server-side implicit API endpoints
- **Built-in optimizations:** Images, Fonts, Scripts
- **Middleware:** Routing/redirects, auth, i18n, experimentation
- **CSS support**: CSS Modules, Global CSS, Sass, CSS-in-JS, Tailwind CSS

Why SSR (Server-side rendering)?

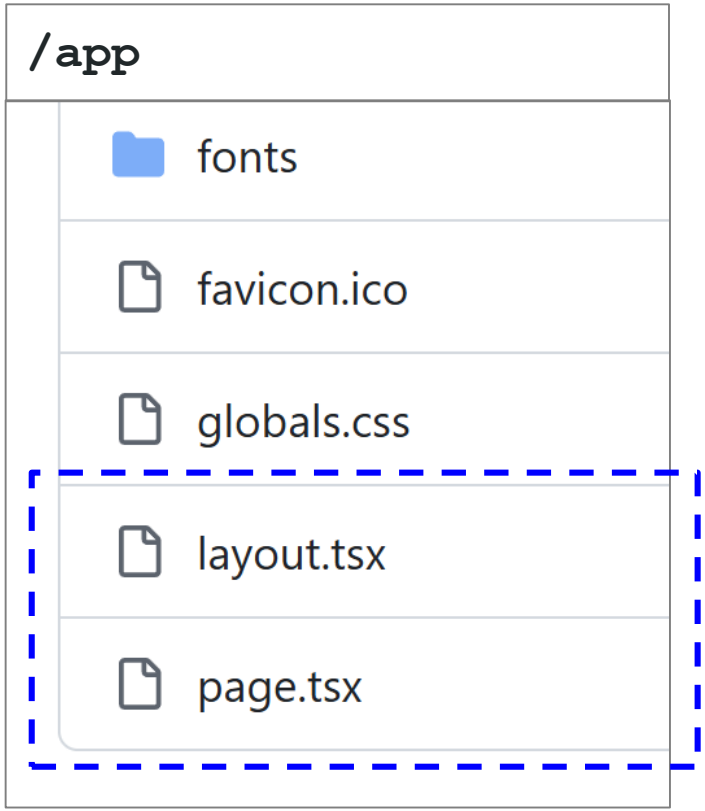
Scenario: Localized content from external CMS

Client-side rendering *(Don't do this!)*

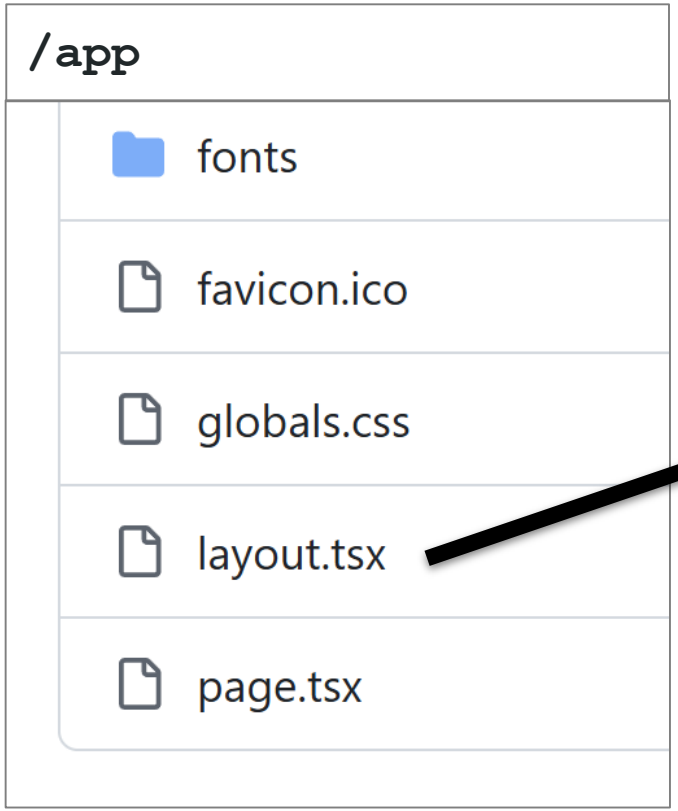
Server-side rendering

Routing

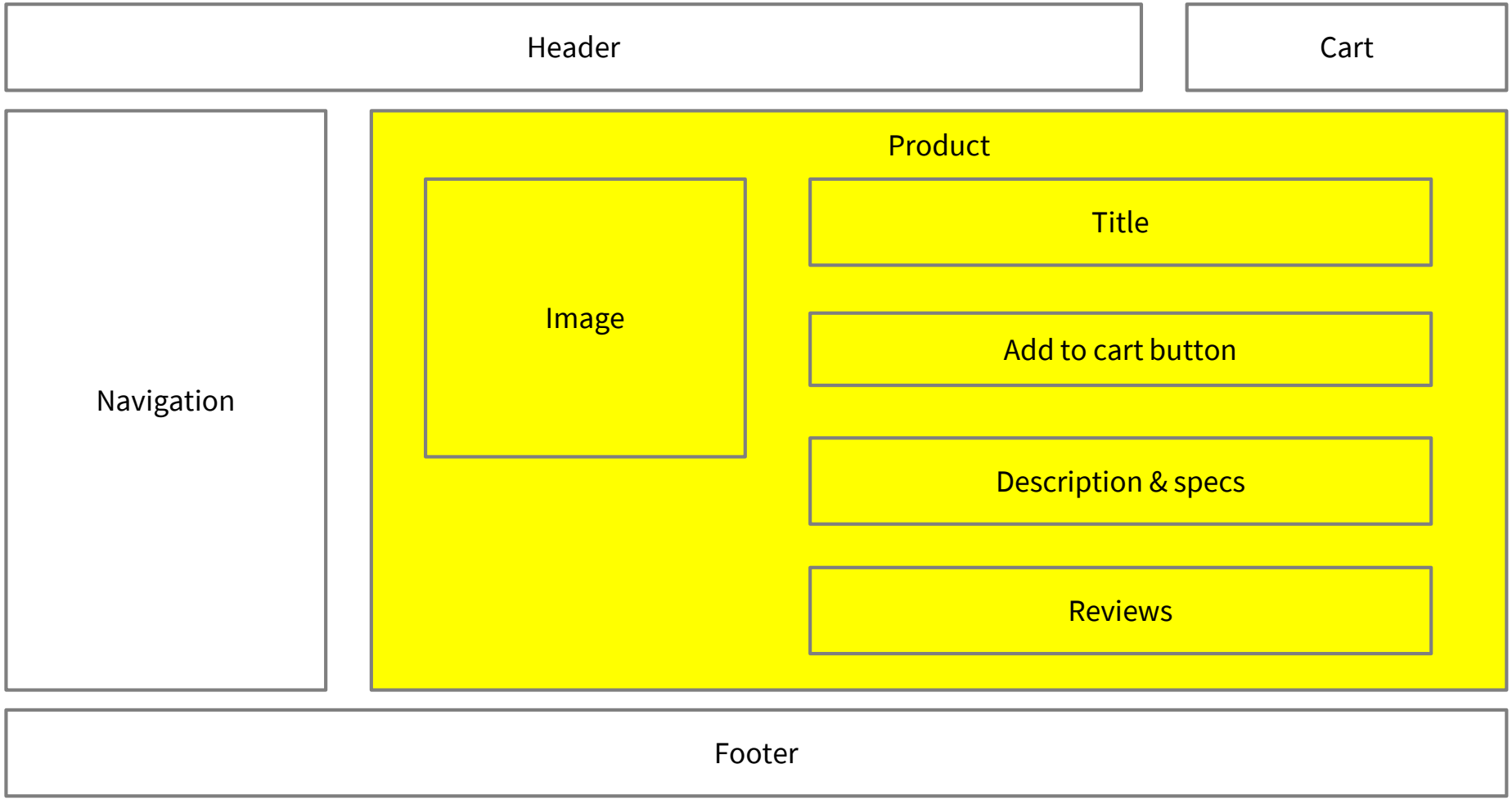
File based routing (“App router”)



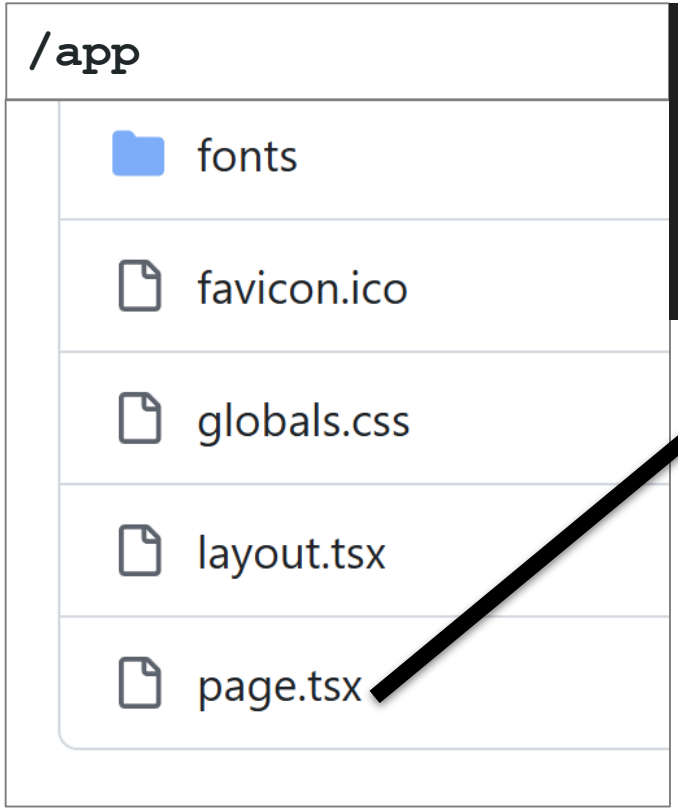
File based routing: layout.tsx




```tsx
import type { Metadata } from "next";
import "./globals.css";

export const metadata: Metadata = {
  title: "Next Level E-com",
  description: "Catchy slogan driving traffic",
};

export default function RootLayout({
  children,
}: {
  children: React.ReactNode;
}) {
  return (
    <html lang="en">
      <body>{children}</body>
    </html>
  );
}
```

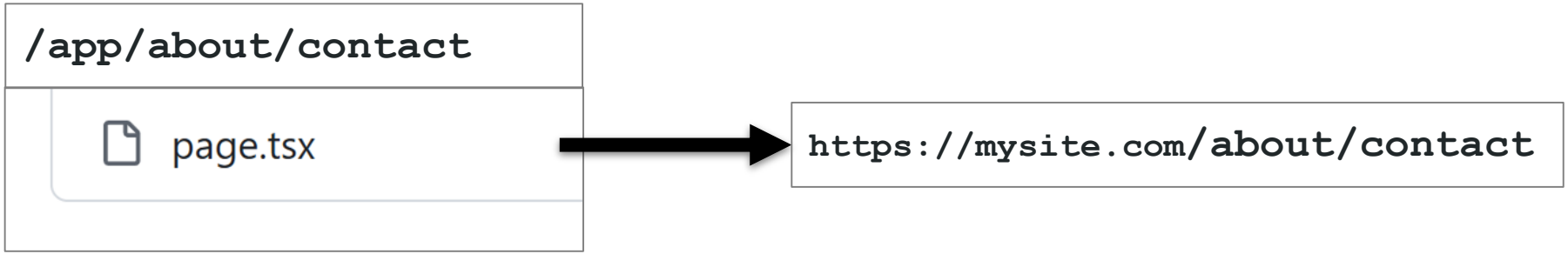

Header
Cart
Navigation
Product
Image
Title
Add to cart button
Description & specs
Reviews
Footer

File based routing: page.tsx



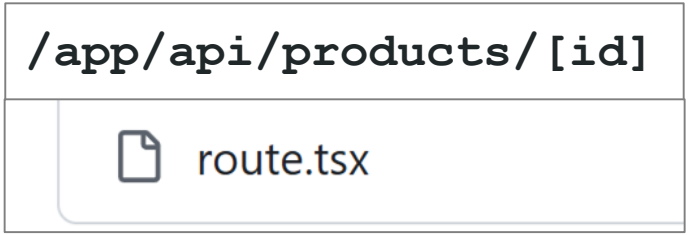

```tsx
export default async function Page() {
  const data = (await fetch("https://api.example.com/product"))

  return (
    <div>
      <h1>{data.title}</h1>
      <div>{data.description}</div>
```

File based routing: URL conventions



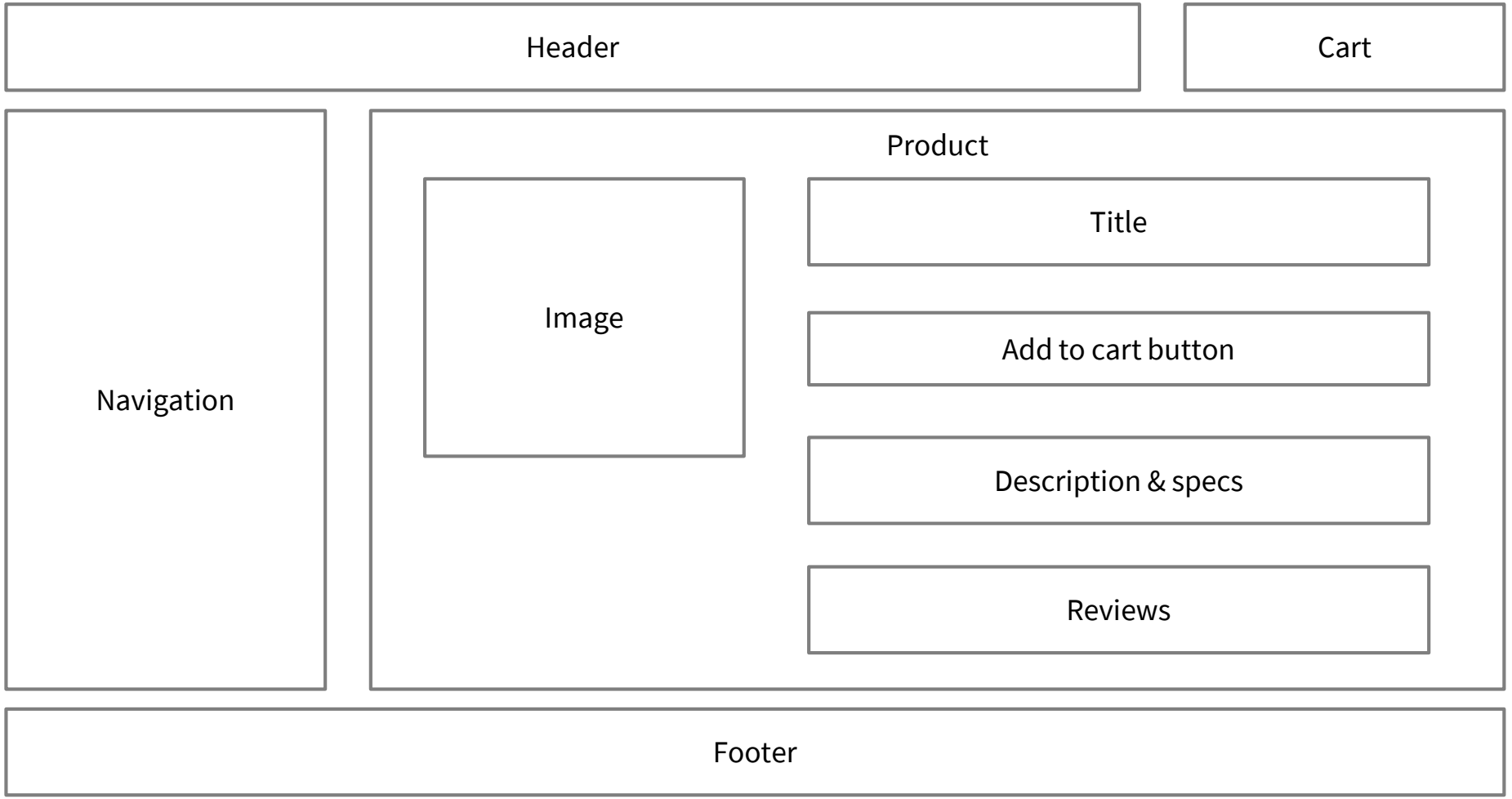
API routes

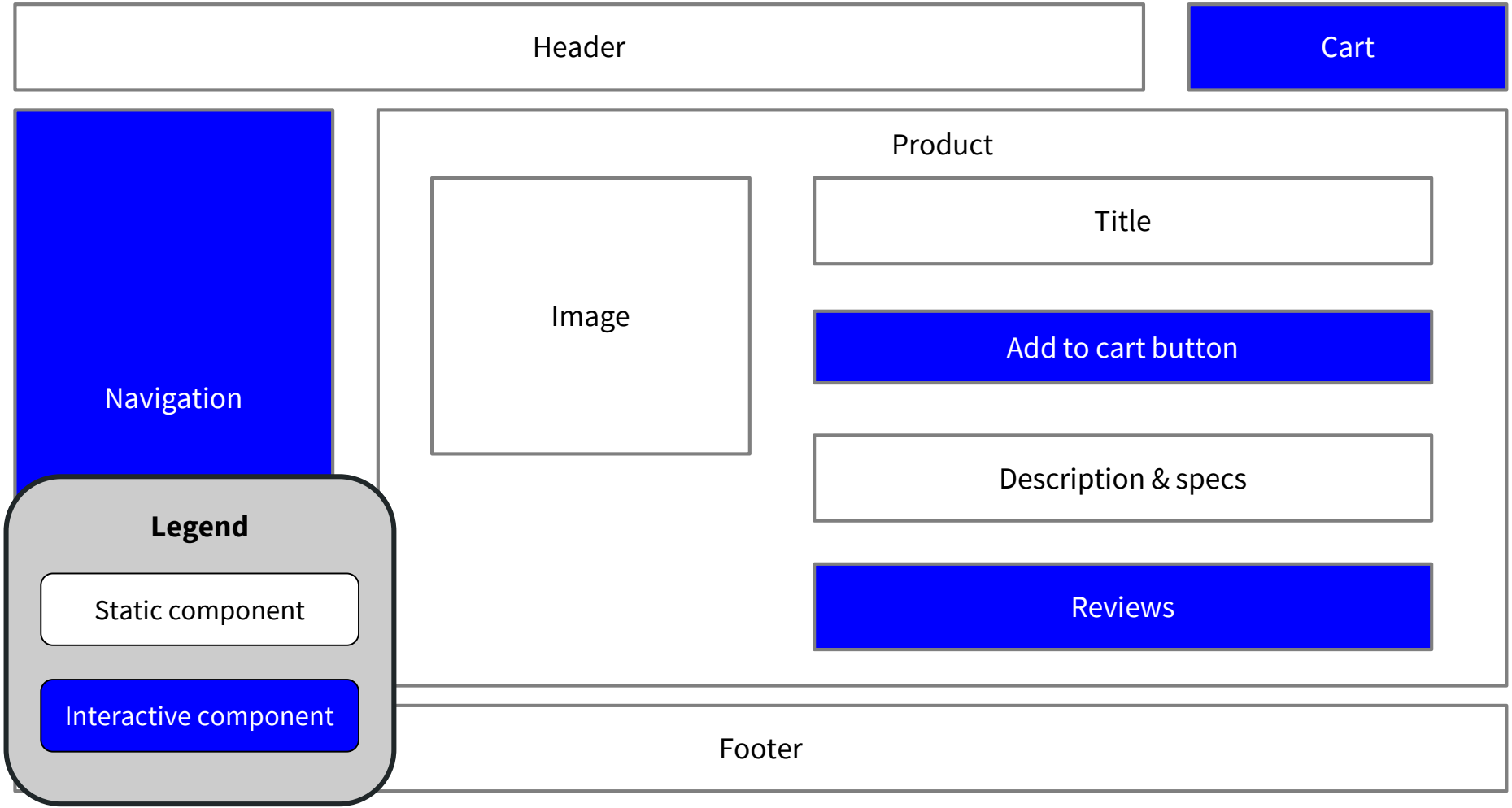
/app/api/products/[id]



GET /api/products/123

```ts
import { type NextRequest } from "next/server";

export async function GET(request: NextRequest) {
  const id = request.nextUrl.searchParams.get("id");

  const data = await fetch(`https://api.mysite.com/products/${id}`);
  const posts = await data.json();

  return Response.json(posts);
}
```


Header
Cart
Navigation
Product
Image
Title
Add to cart button
Description & specs
Reviews
Footer


Header
Cart
Navigation
Product
Image
Title
Add to cart button
Description & specs
Reviews
Footer
Legend
Static component
Interactive component

Client-side - Interactive UI

/app/components/AddToCartButton.tsx

```
1   "use client";
2
```

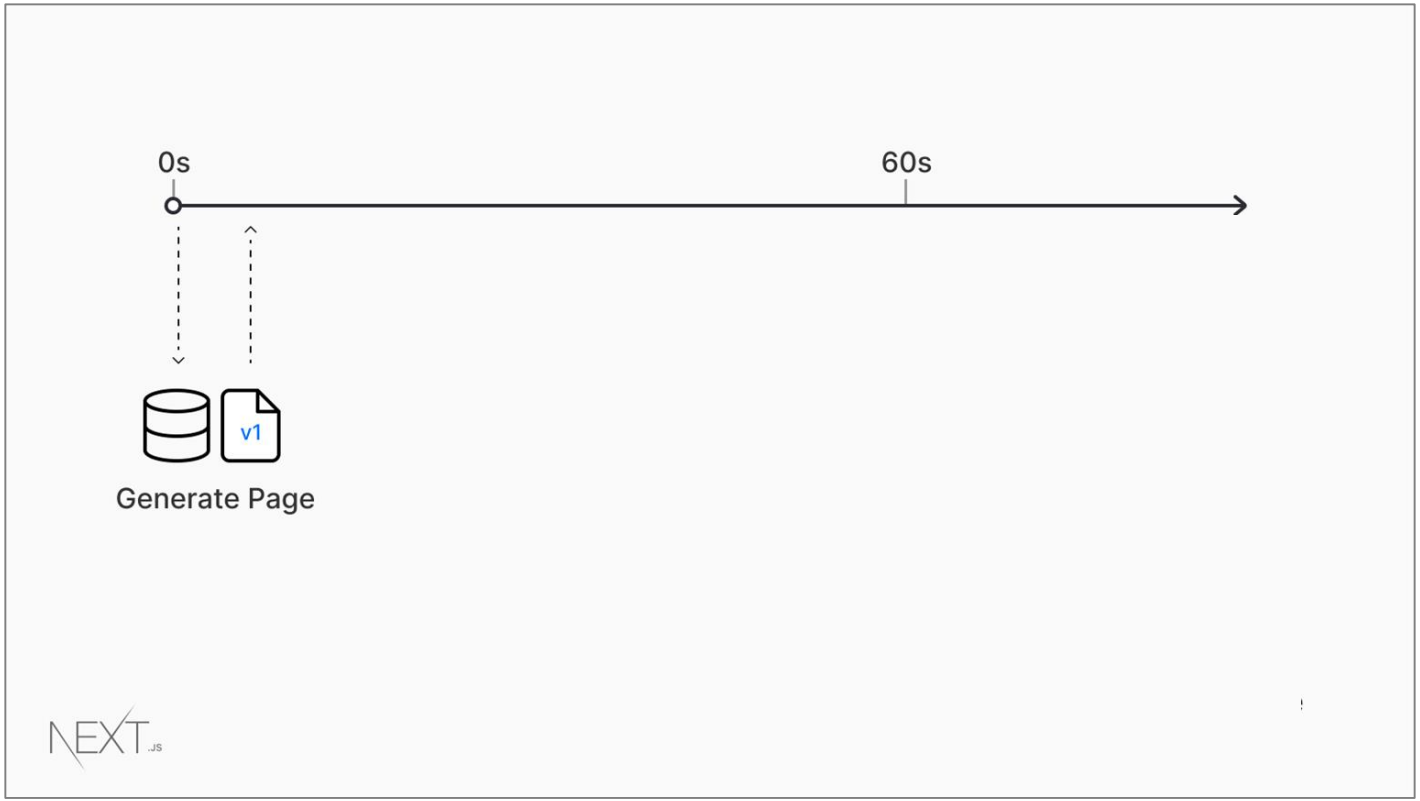
Components: Server-side vs. Client-side

- Server-side don't run React in the browser
  - State, event-handling, and browser APIs **not** available
- Nesting server-side & client-side works
- Server-side are rendered on the server
  - Client-side are rendered **once** (initially) on the server
- Server-side should be used to ensure not leaking secret content

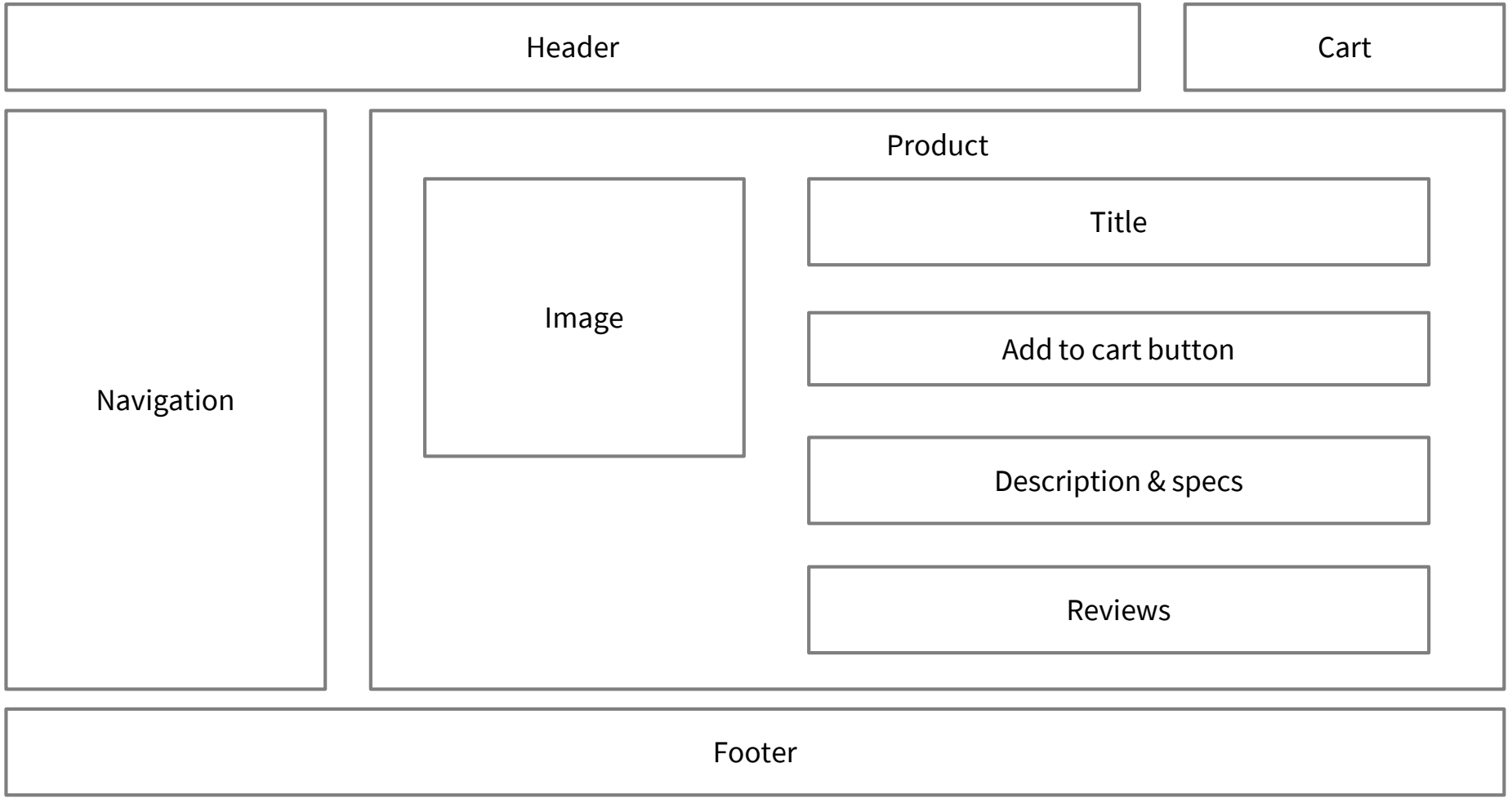
```
{isLoggedIn ? <>Secret content</> : <>Public content</>}
```

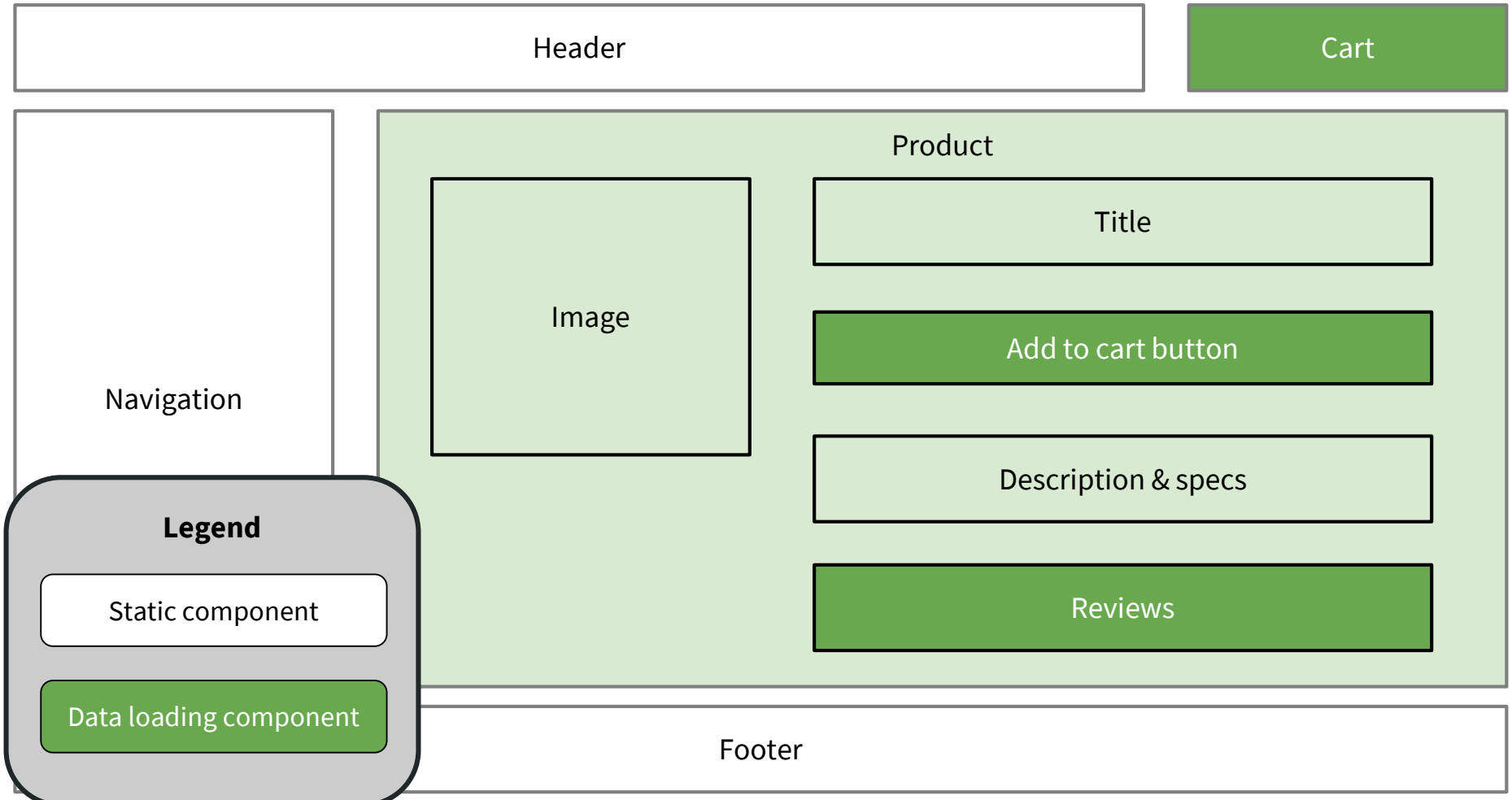
Incremental Static Regeneration (ISR)

Incremental Static Regeneration (ISR)



Loading & streaming

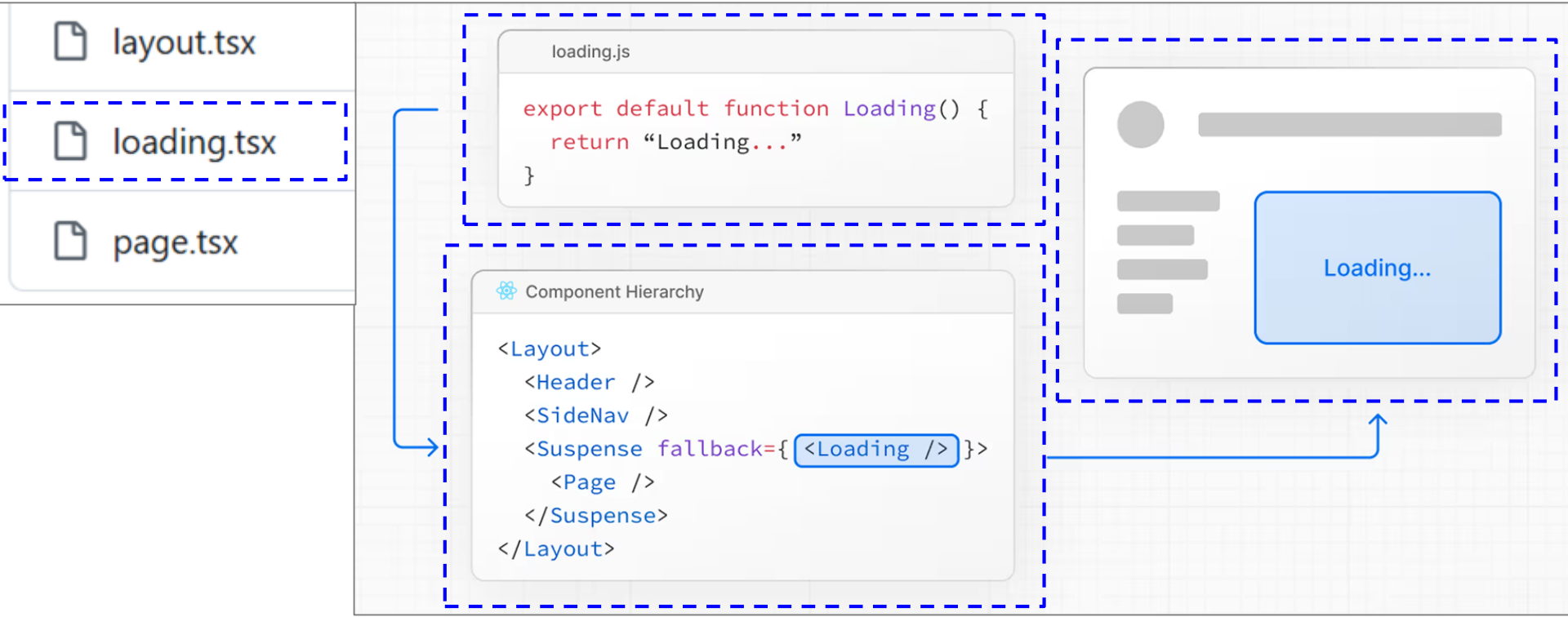

Header
Cart
Navigation
Product
Image
Title
Add to cart button
Description & specs
Reviews
Footer


Header
Cart
Navigation
Product
Image
Title
Add to cart button
Description & specs
Reviews
Footer
Legend
Static component
Data loading component

Loading & streaming

- Show content as fast as possible
  - Stream components when ready
  - Allows displaying fallback until load finished
- Uses React's `<Suspense />`-component
- Manual or automatic usage
  - Manually through `<Suspense />`-component
  - Automatically through Next.js conventions

Loading & streaming: Next.js conventions



Thank you!

https://sebnilsson.com

@sebnilsson