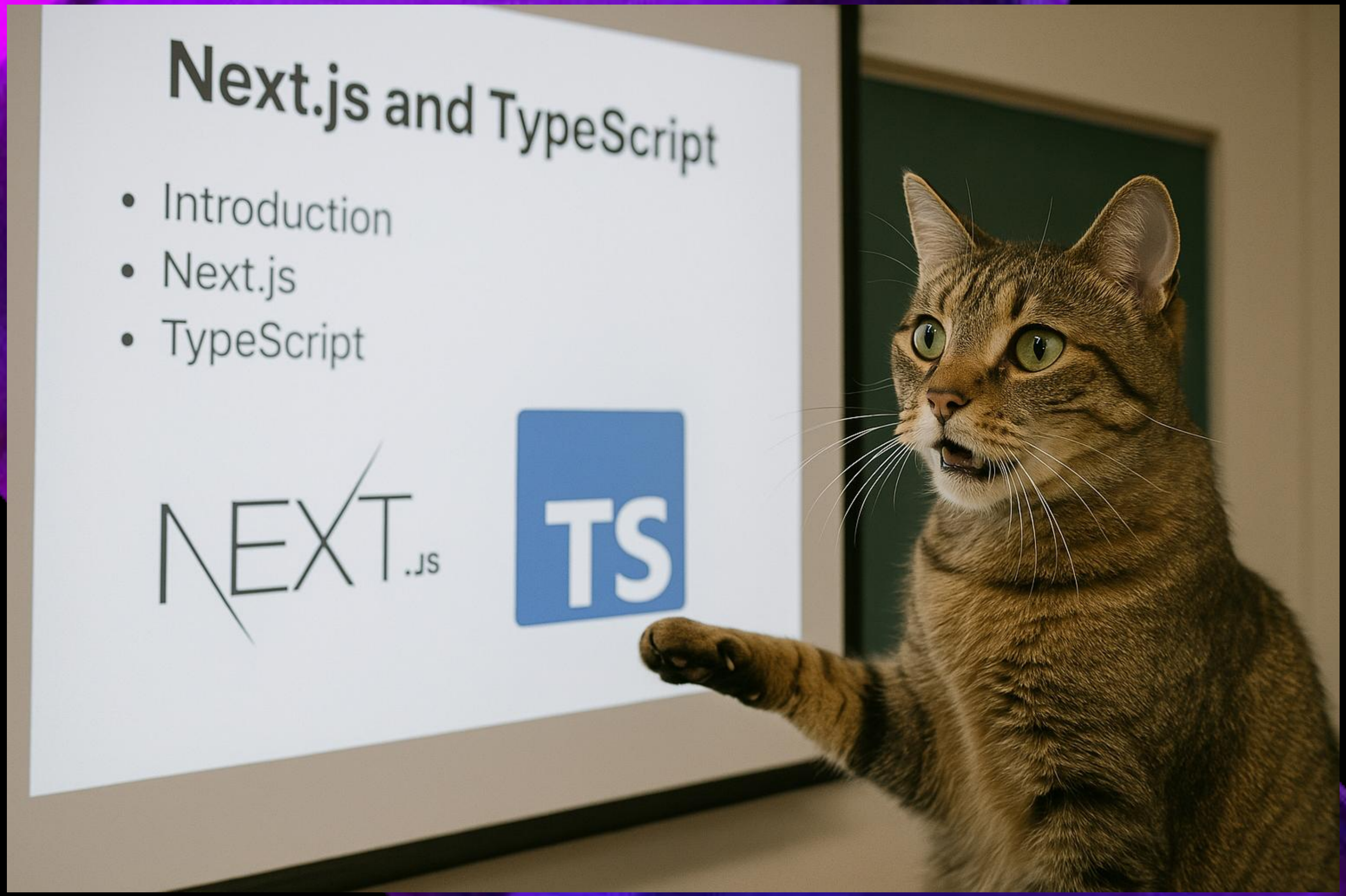
Next.js and TypeScript

With Jake and Nate



TypeScript vs JS

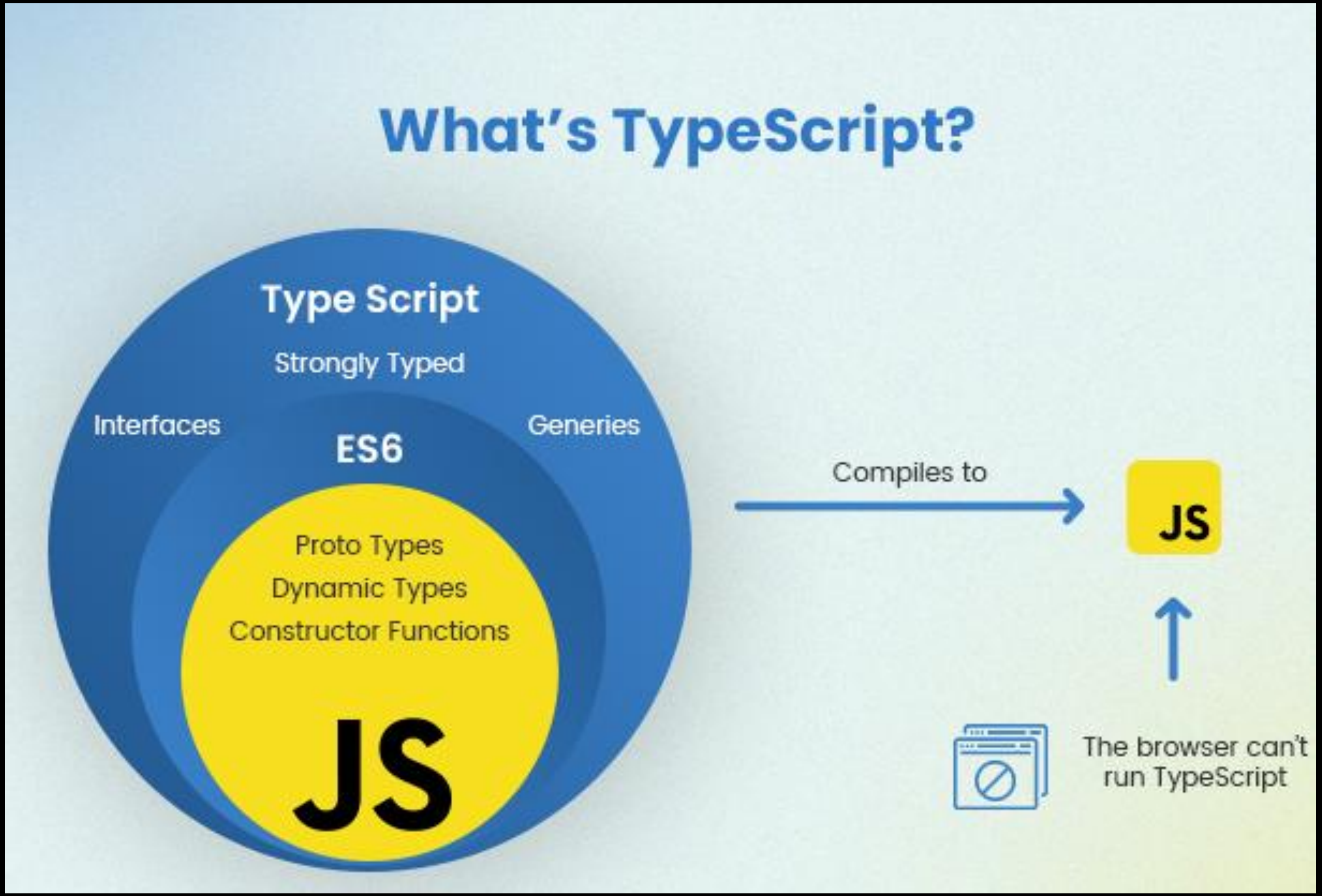
Next.js Overview

Next.js Demo

TypeScript vs. JavaScript

JavaScript:
- Dynamic, interpreted scripting language
- Will run directly in browsers
- Typically, better for speed and simplicity
- Files end with .js

TypeScript:
- Superset of JavaScript with static typing
- Must be compiled to JavaScript
- Typically, better for scalability and easier to maintain in the long run
- Files end with .ts or .tsx for React

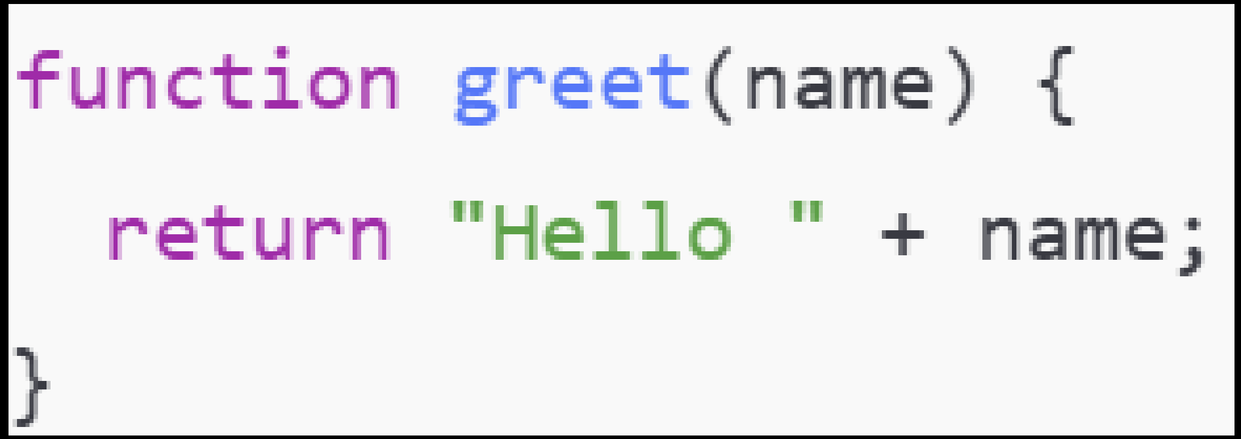


TypeScript vs. JavaScript

JavaScript
- Dynamically typed: Types are checked at runtime
- Weakly typed: Not strict about type enforcement

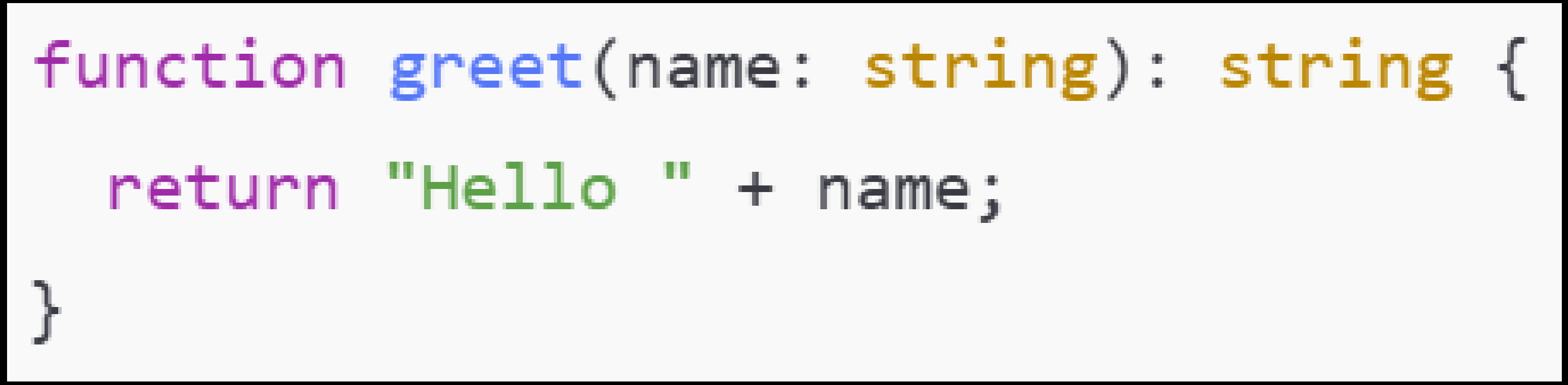
TypeScript
- Statically typed: Types are checked at compile time
- Strongly typed: Strict about type enforcement

JavaScript (Dynamically Typed)



```
function greet(name) {
  return "Hello " + name;
}
```

TypeScript (Statically Typed)



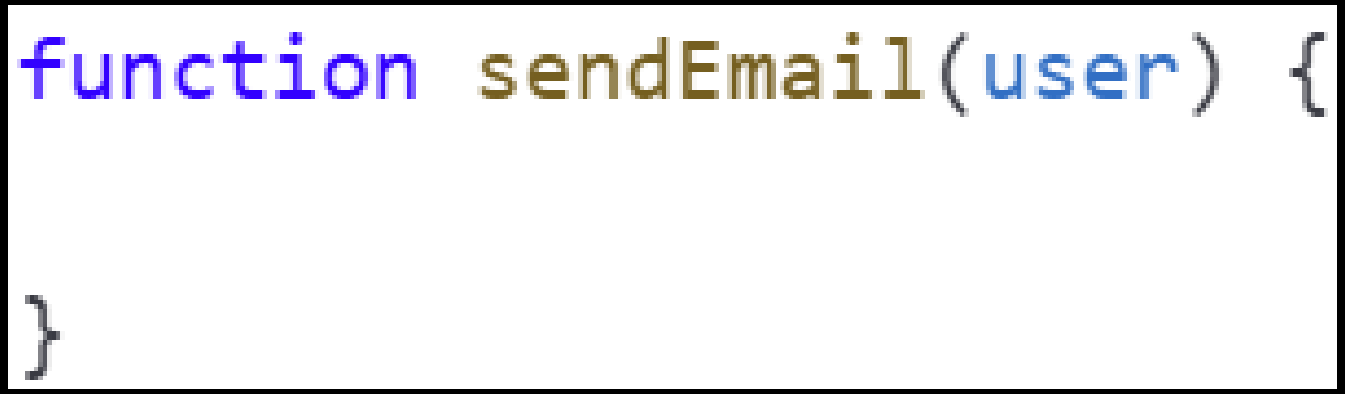
```
function greet(name: string): string {
  return "Hello " + name;
}
```

If you pass a number instead of a string, TypeScript will throw an error, but JavaScript will not

Why Use TypeScript?

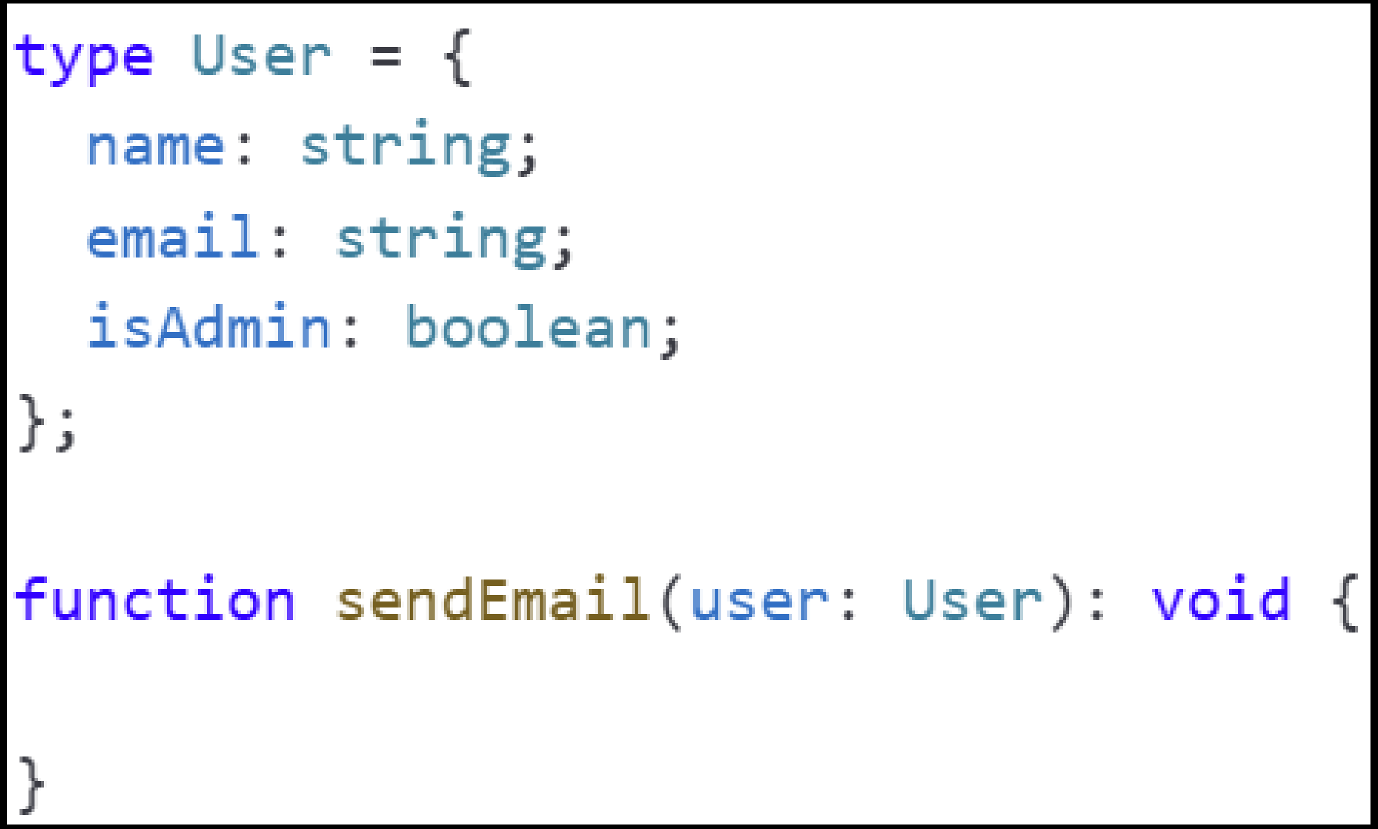
- You can catch bugs earlier because types are checked at compile time instead of runtime
- Safer to refactor because when adding something new, TypeScript will tell you everywhere it breaks so you can fix it.
- Strong types ensure your responses and query parameters are valid and expected
- TypeScript is self-documenting

JavaScript:



```
function sendEmail(user) {

}
```

TypeScript:



```
type User = {
  name: string;
  email: string;
  isAdmin: boolean;
};

function sendEmail(user: User): void {

}
```

What is Next.js?

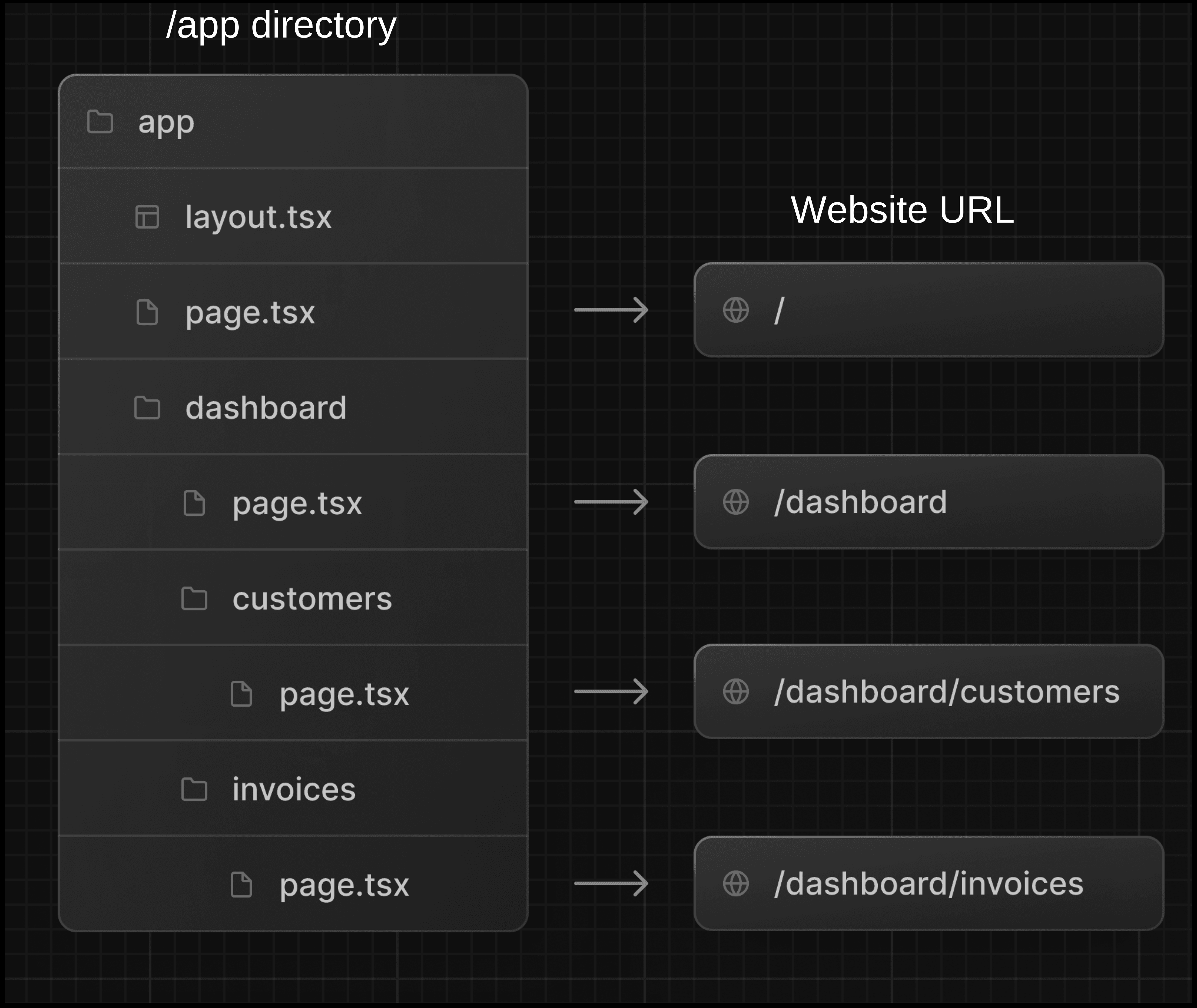
A React framework that enables server-side rendering and static site generation.

Key Features:

- File based routing
- Built-in API routes
- SSR
- Optimized Performance and SEO

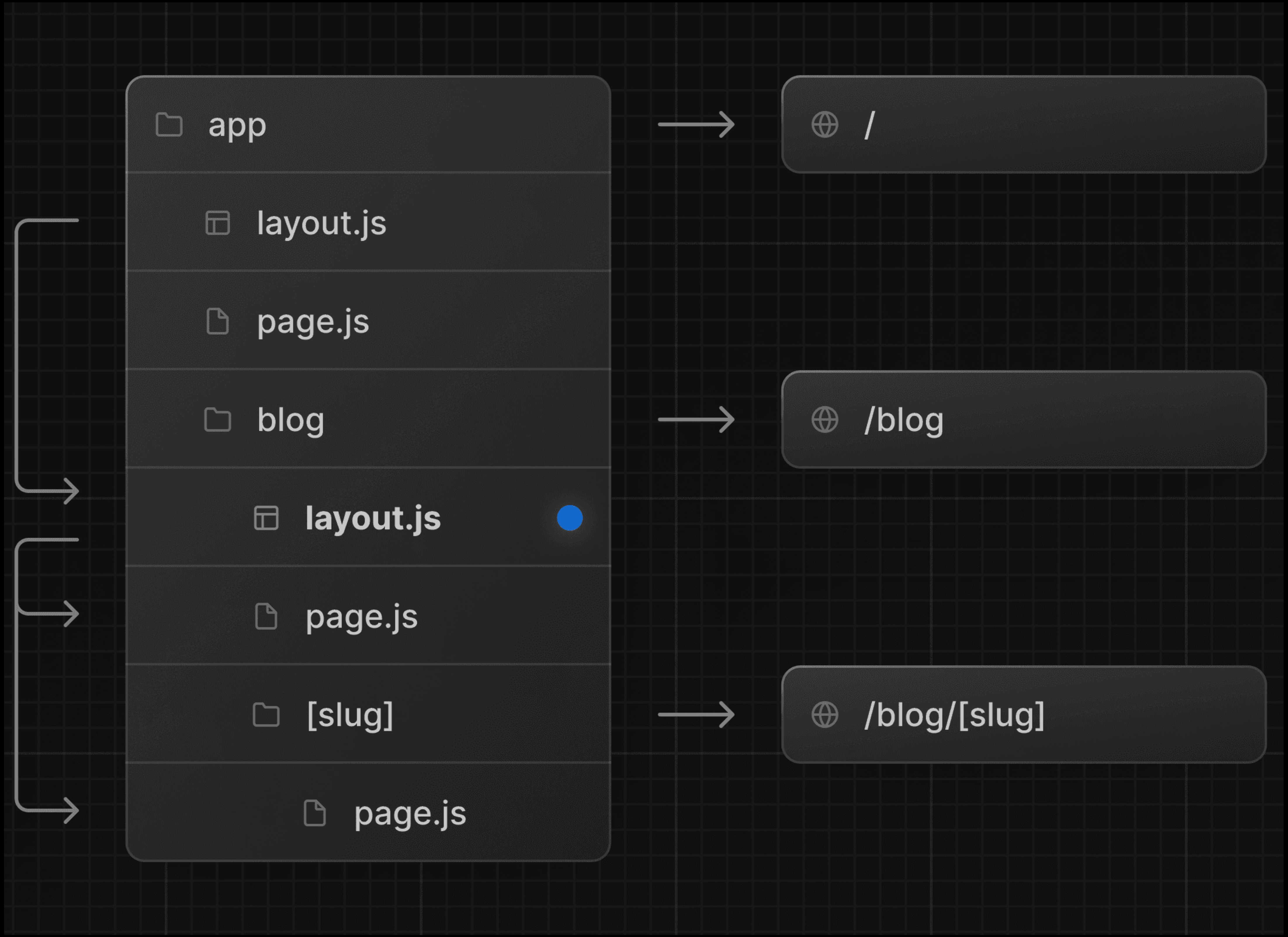
Next App Router

- Each folder in app/ becomes a route. app/settings/page.js is https://example.com/settings
- Use layout.js for shared ui in folder below its location (footers, headers, navbars, etc)
- Server vs Client components: Components are rendered on the server unless 'use client' is denoted at the top of a page. Must be client to use many react features
- Built-in file conventions like loading.tsx and error.tsx



Next Layouts

- A way to define a consistent structure for your website's pages in Next.js
- Helps you reuse common elements (like headers, footers, menus) without rewriting code
- You create layout files that wrap around your individual page content



Next Link/Navigation

- Two ways for client pages, using <Link> component. Or useRouter
- <Link> is a built-in component that extends the HTML <a> tag to provide prefetching and client-side navigation between routes. It is the primary and recommended way to navigate between routes in Next.js.
- The useRouter hook allows you to programmatically change routes from Client Components.

```
import Link from 'next/link'

export default function Page() {
  return <Link href="/dashboard">Dashboard</Link>
}
```

```
'use client'

import { useRouter } from 'next/navigation'

export default function Page() {
  const router = useRouter()

  return (
    <button type="button" onClick={() => router.push('/dashboard')}>
      Dashboard
    </button>
  )
}
```

Fetching Data

- On client: either use fetch in a useEffect or the “use” hook

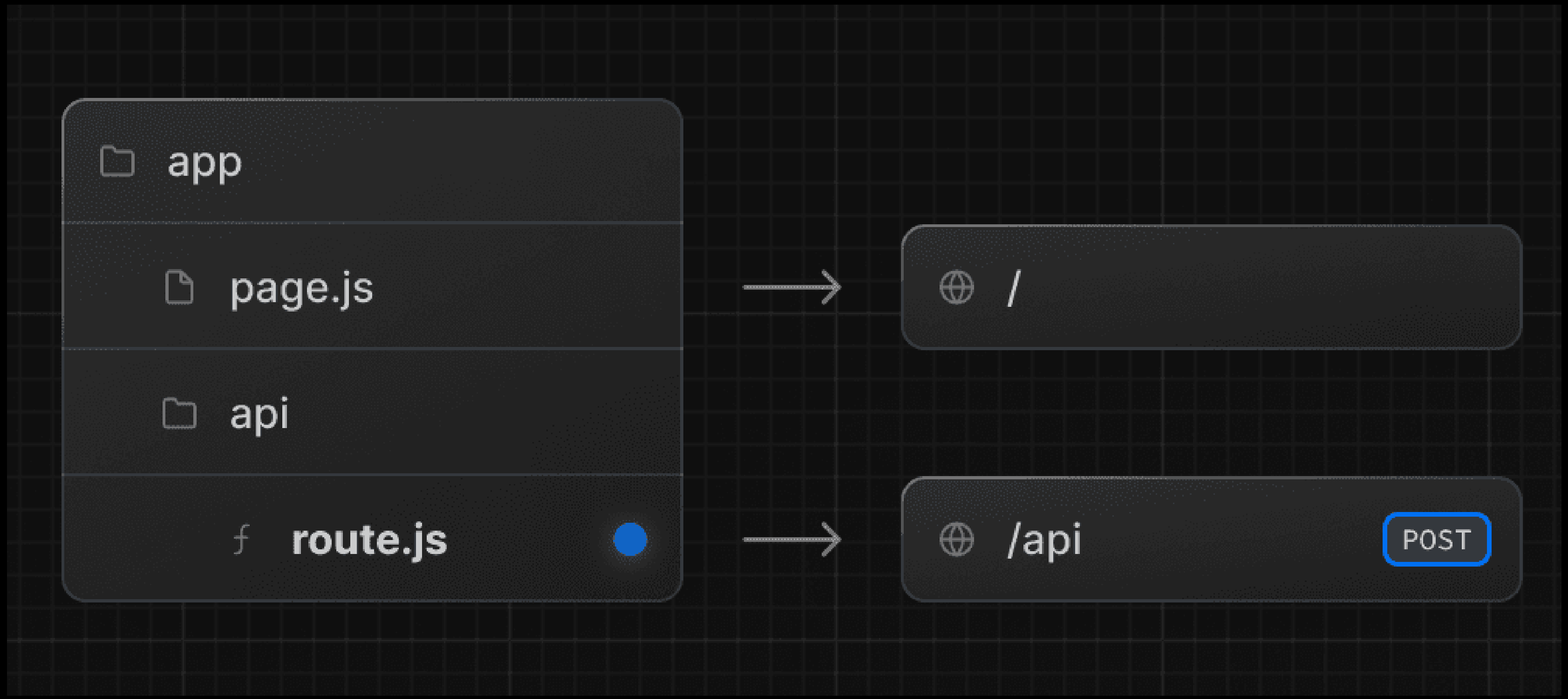
- On server: asynchronously use the fetch api

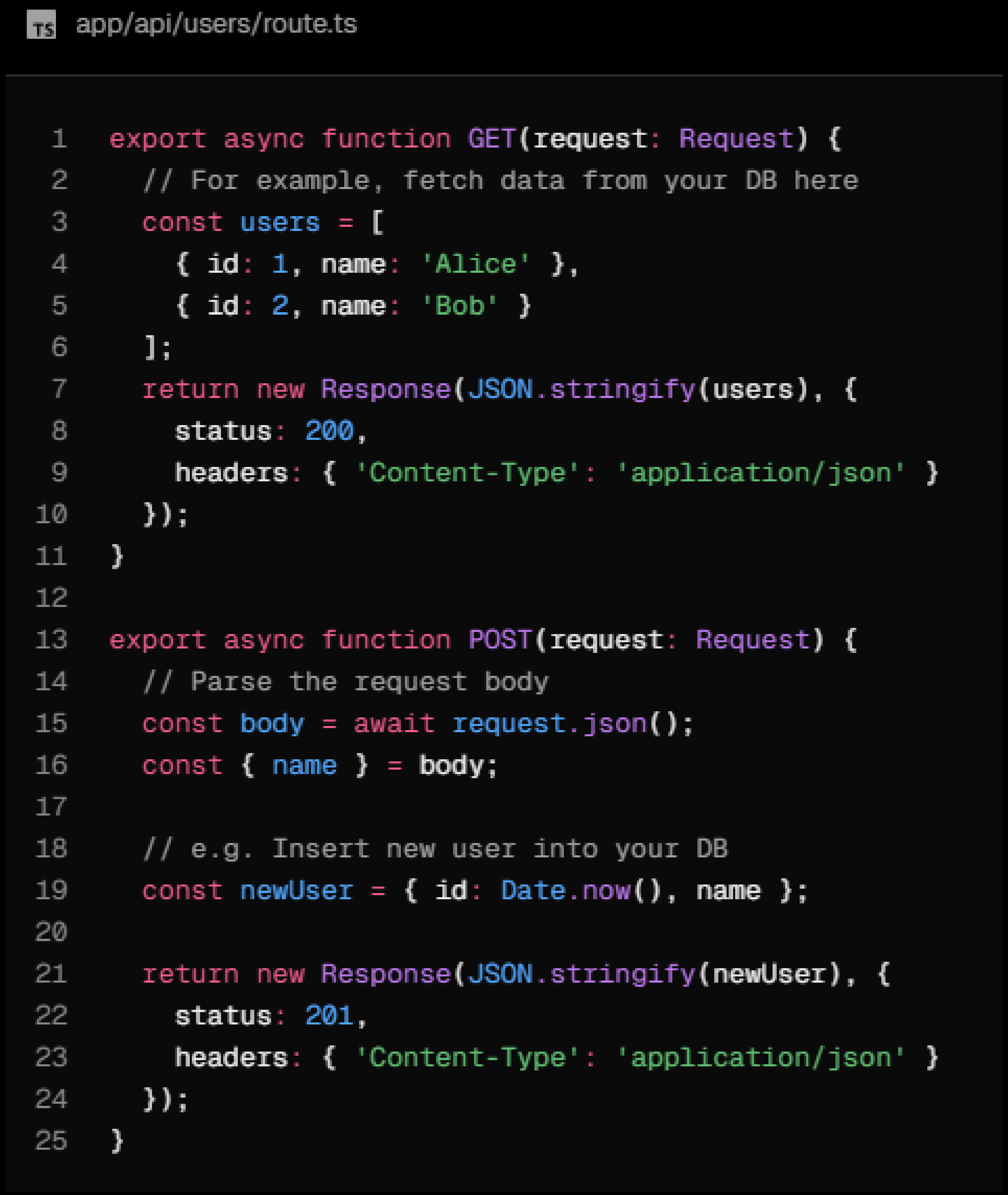
```
export default async function Page() {
  const data = await fetch('https://api.vercel.app/blog')
  const posts = await data.json()
  return (
    <ul>
      {posts.map((post) => (
        <li key={post.id}>{post.title}</li>
      ))}
    </ul>
  )
}
```

```
'use client'

import { useState, useEffect } from 'react'

export function Posts() {
  const [posts, setPosts] = useState(null)

  useEffect(() => {
    async function fetchPosts() {
      const res = await fetch('https://api.vercel.app/blog')
      const data = await res.json()
      setPosts(data)
    }
    fetchPosts()
  }, [])

  if (!posts) return <div>Loading...</div>

  return (
    <ul>
      {posts.map((post) => (
        <li key={post.id}>{post.title}</li>
      ))}
    </ul>
  )
}
```

API route

- Route Handlers allow you to create custom request handlers for a given route using the Web Request and Response APIs.

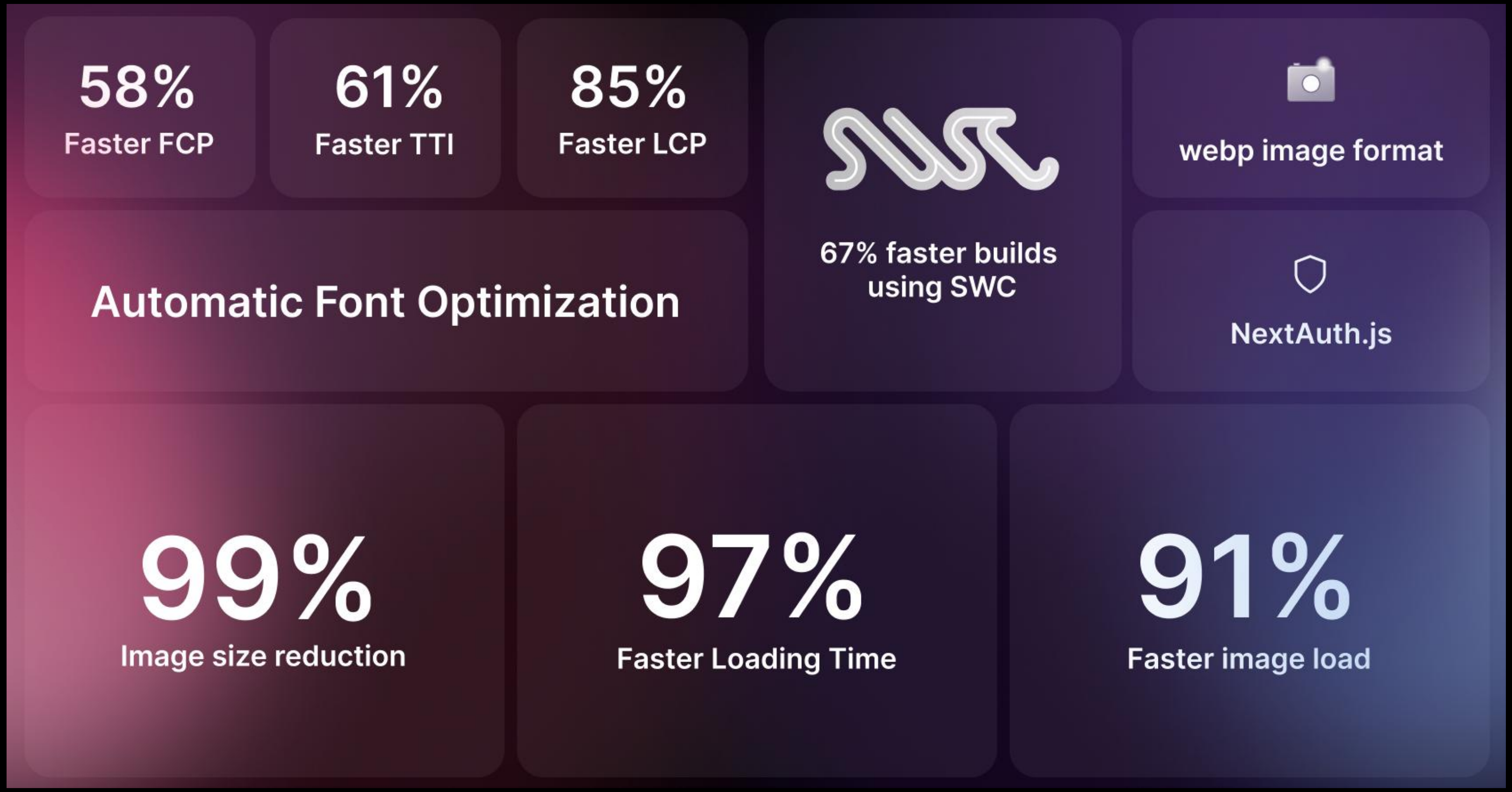



app/api/users/route.ts

```ts
export async function GET(request: Request) {
  // For example, fetch data from your DB here
  const users = [
    { id: 1, name: 'Alice' },
    { id: 2, name: 'Bob' }
  ];
  return new Response(JSON.stringify(users), {
    status: 200,
    headers: { 'Content-Type': 'application/json' }
  });
}

export async function POST(request: Request) {
  // Parse the request body
  const body = await request.json();
  const { name } = body;

  // e.g. Insert new user into your DB
  const newUser = { id: Date.now(), name };

  return new Response(JSON.stringify(newUser), {
    status: 201,
    headers: { 'Content-Type': 'application/json' }
  });
}
```

Optimizations

- Images: Built on the native <img> element. The Image Component optimizes images for performance by lazy loading and automatically resizing images based on device size.
- Link: Built on the native <a> tags. The Link Component prefetches pages in the background, for faster and smoother page transitions.
- Scripts: Built on the native <script> tags. The Script Component gives you control over loading and execution of third-party scripts.
- Next.js /public folder can be used to serve static assets like images, fonts, and other files. Files inside /public can also be cached by CDN providers so that they are delivered efficiently.



DEMO TIME:

tinyurl.com/53vwk6wc