Introduction to Next.js

Asst.Prof.Drusawin Vongpramate

Department of Information Technology

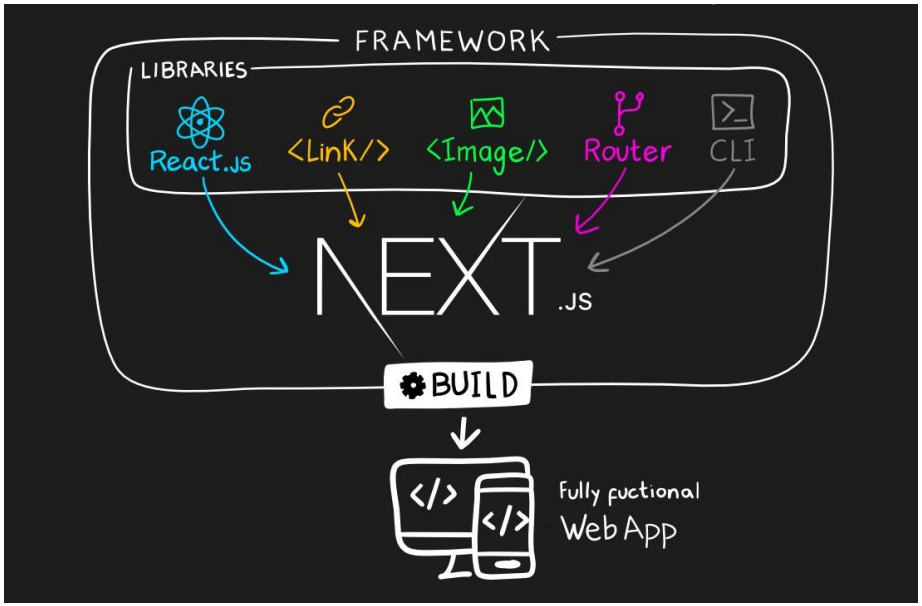
Faculty of Science, BRU

Hint

Main Topic Sub Topic

What's in Next.js?

Next.js is a React framework for building full-stack web applications. You use React Components to build user interfaces, and Next.js for additional features and optimizations.



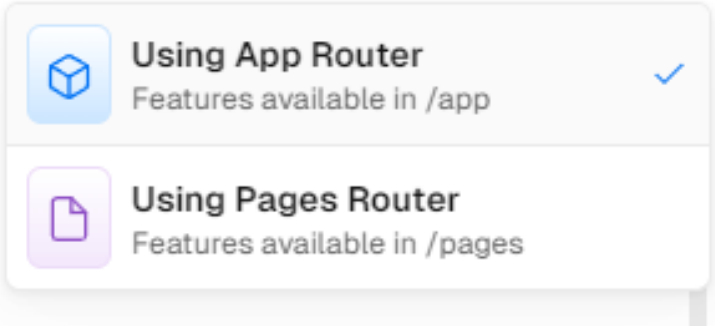
Ref: https://nextjs.org/docs/app/

Main Features

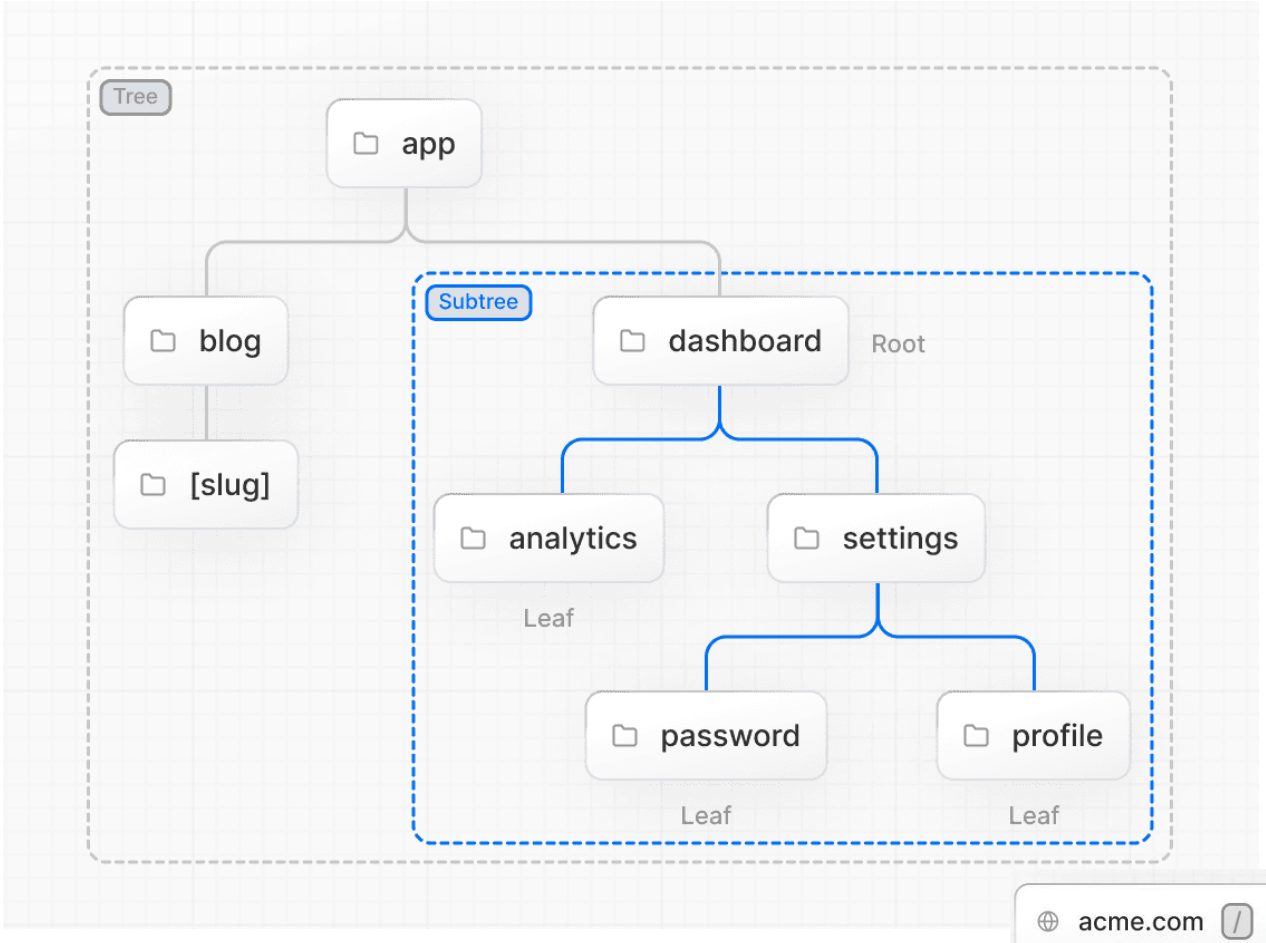
Feature	Description
Routing	A file-system based router built on top of Server Components that supports layouts, nested routing, loading states, error handling, and more.
Rendering	Client-side and Server-side Rendering with Client and Server Components. Further optimized with Static and Dynamic Rendering on the server with Next.js. Streaming on Edge and Node.js runtimes.
Data Fetching	Simplified data fetching with async/await in Server Components, and an extended fetch API for request memoization, data caching and revalidation.
Styling	Support for your preferred styling methods, including CSS Modules, Tailwind CSS, and CSS-in-JS
Optimizations	Image, Fonts, and Script Optimizations to improve your application's Core Web Vitals and User Experience.
TypeScript	Improved support for TypeScript, with better type checking and more efficient compilation, as well as custom TypeScript Plugin and type checker.

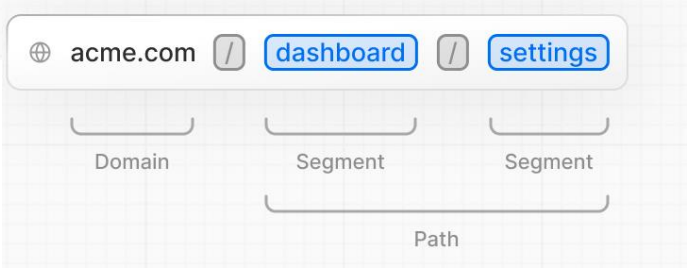
App Router vs Pages Router

- Next.js has two different routers: the App Router and the Pages Router.
- **The App Router** is a newer router that allows you to use React's latest features, such as **Server Components** and Streaming.
- **The Pages Router** is the original Next.js router, which allowed you to build server-rendered React applications and continues to be supported for older Next.js applications.



Routing


Tree
app
blog
[slug]
Subtree
dashboard
Root
analytics
Leaf
settings
password
Leaf
profile
Leaf


acme.com
/
dashboard
/
settings
Domain
Segment
Segment
Path

Roles of Folders and Files

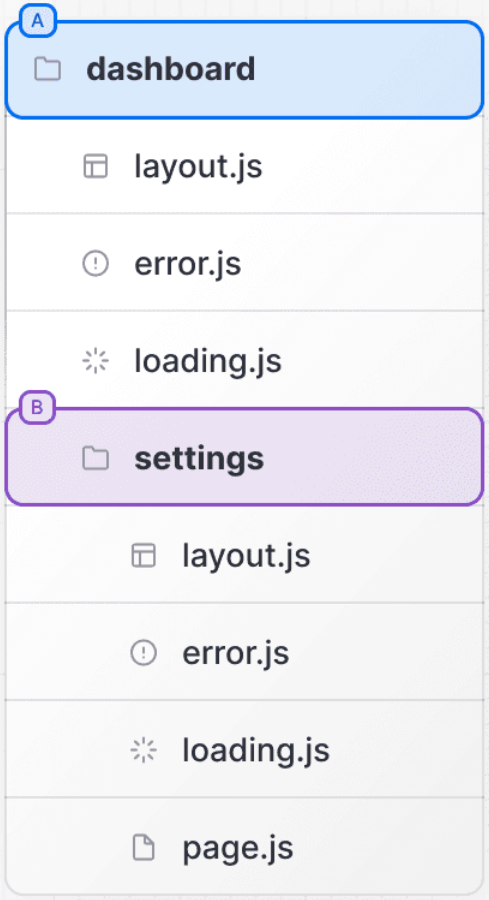
- Next.js uses a file-system based router where:
- Folders are used to define routes. A route is a single path of nested folders, following the file-system hierarchy from the root folder down to a final leaf folder that includes a page.js file. See Defining Routes.
- Files are used to create UI that is shown for a route segment. See special files (next slide)

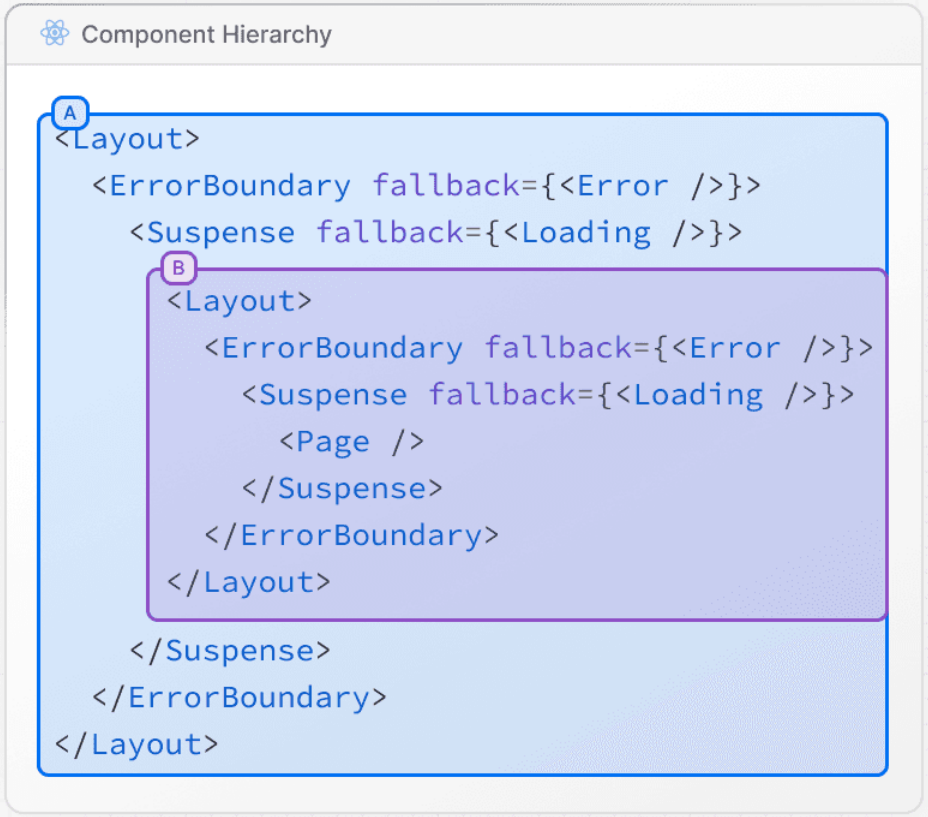
File Conventions

layout	Shared UI for a segment and its children
page	Unique UI of a route and make routes publicly accessible
loading	Loading UI for a segment and its children
not-found	Not found UI for a segment and its children
error	Error UI for a segment and its children
global-error	Global Error UI
route	Server-side API endpoint
template	Specialized re-rendered Layout UI
default	Fallback UI for Parallel Routes

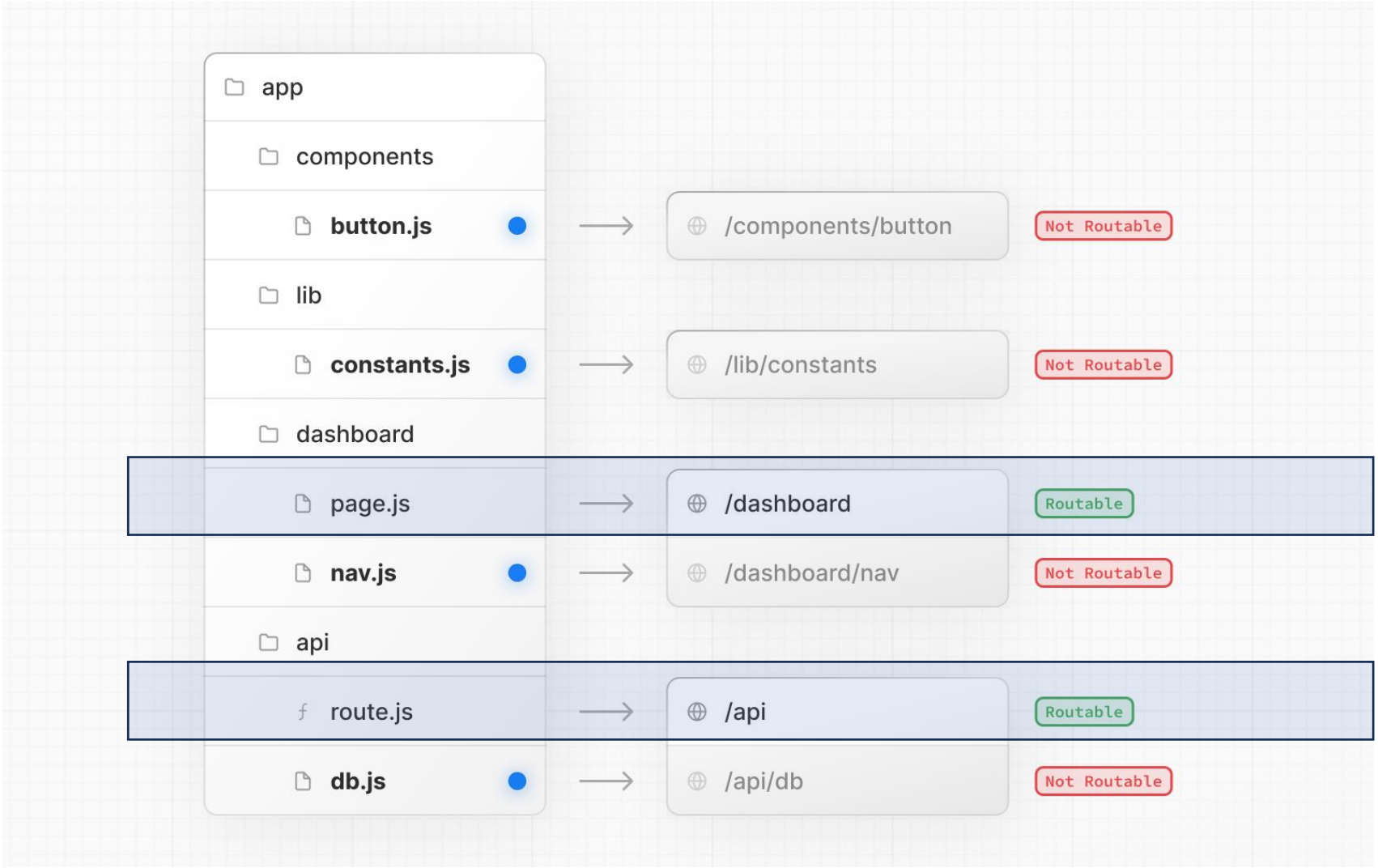
.js, .jsx, or .tsx file extensions can be used for special files.

Component Hierarchy



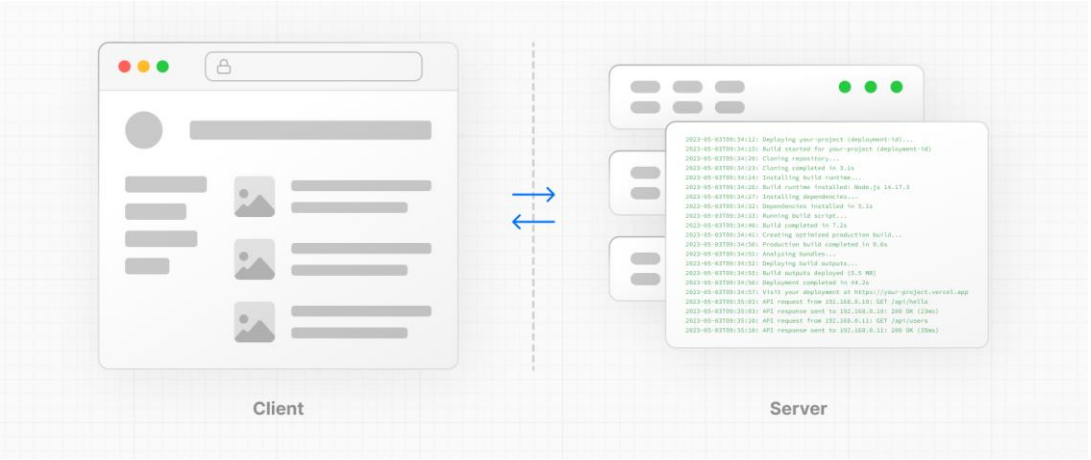


Colocation

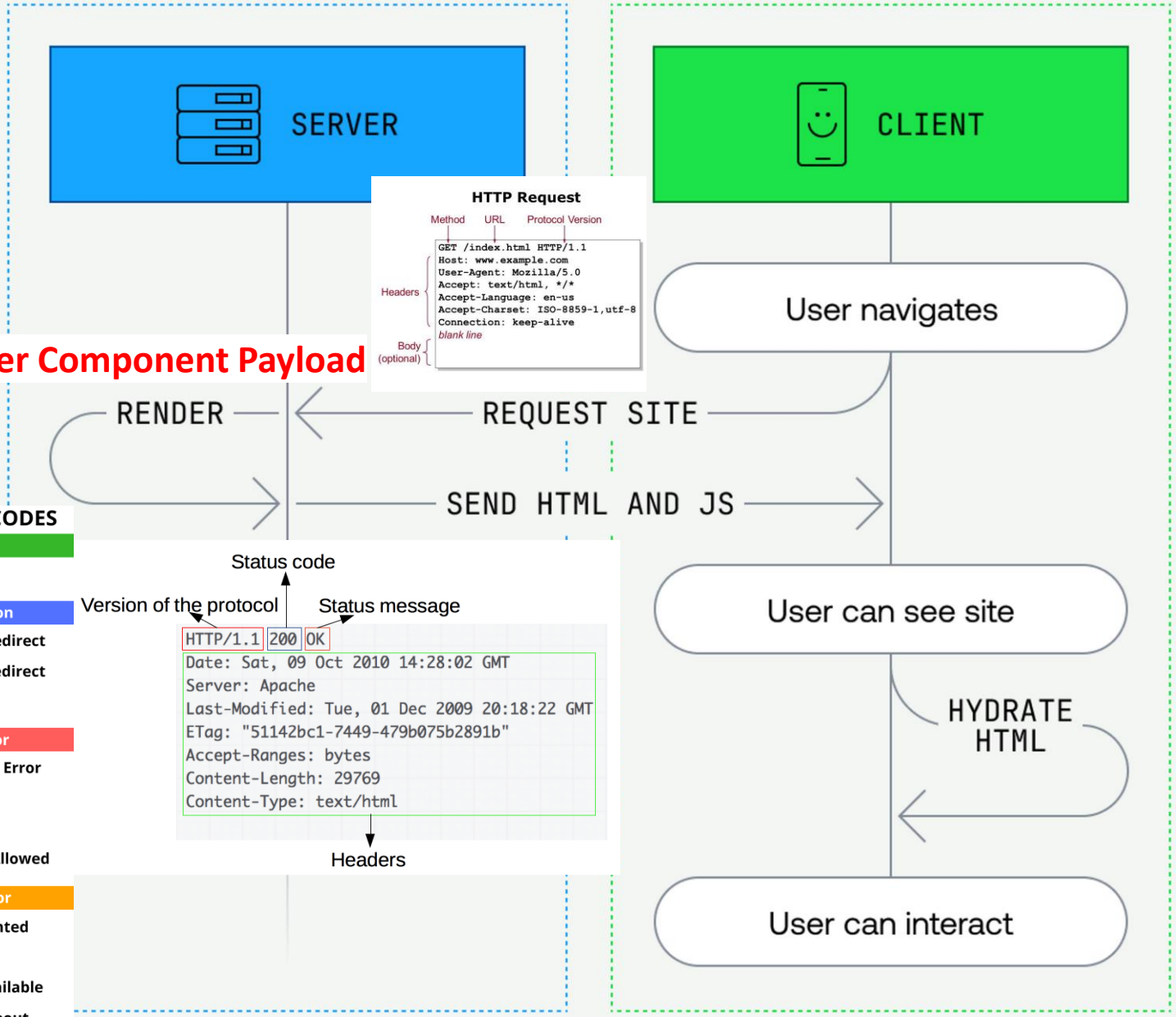

app
components
button.js
/components/button
Not Routable
lib
constants.js
/lib/constants
Not Routable
dashboard
page.js
/dashboard
Routable
nav.js
/dashboard/nav
Not Routable
api
route.js
/api
Routable
db.js
/api/db
Not Routable

Rendering

- Rendering converts the code you write into user interfaces. React and Next.js allow you to create hybrid web applications where parts of your code can be rendered on the server or the client. This section will help you understand the differences between these rendering environments, strategies, and runtimes.



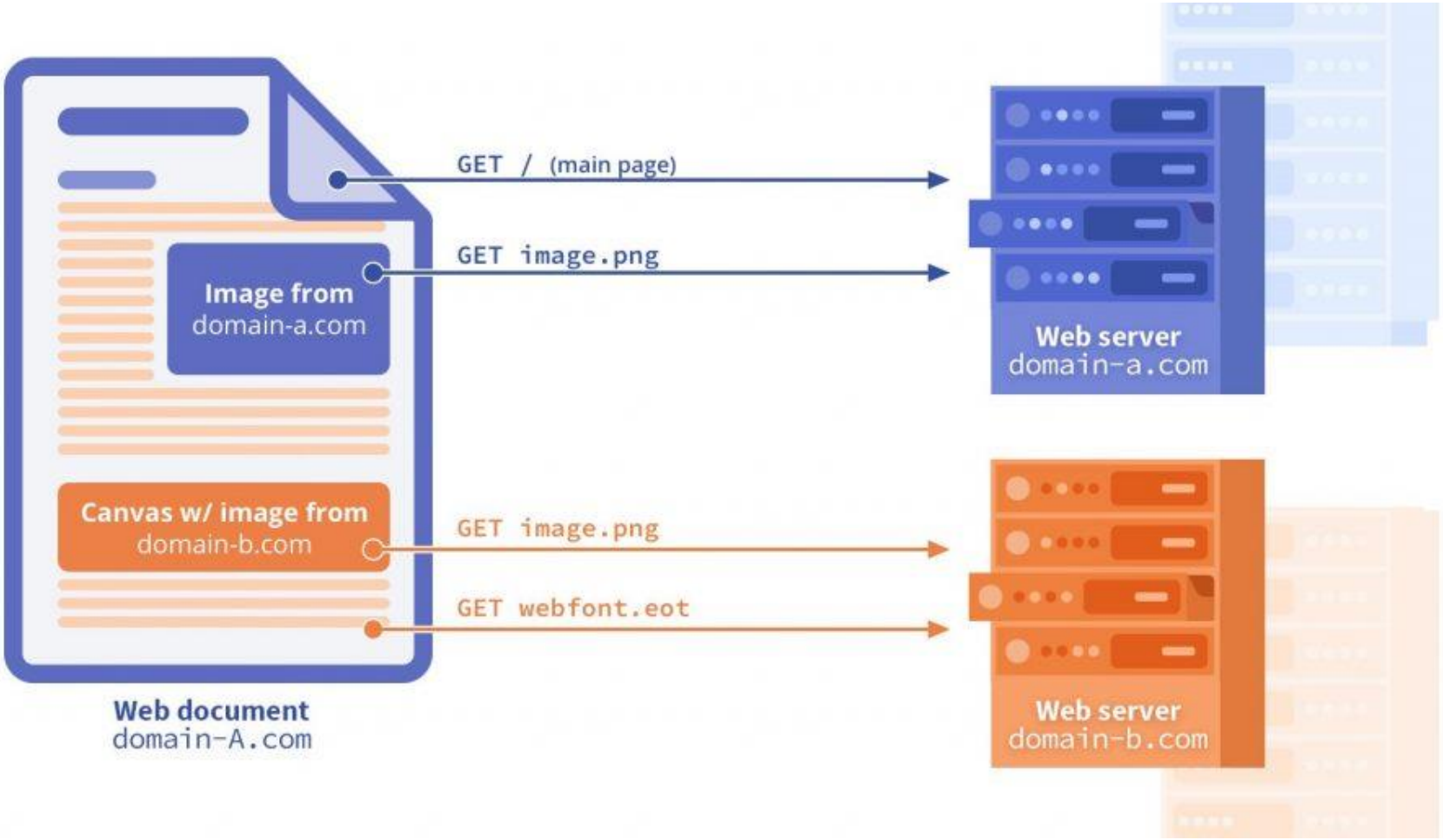
Request-Response Lifecycle



Cross-Origin Resource Sharing (CORS)

- Cross-site HTTP requests are HTTP requests for resources from a different domain than the domain of the resource making the request. For instance, an HTML page from Domain A (http://domaina.example/) makes a request for an image on Domain B (http://domainb.foo/image.jpg) via the img element. Web pages today very commonly load cross-site resources, including CSS stylesheets, images, scripts, and other resources. CORS allows web developers to control how their site reacts to cross-site requests.

Cross-Origin Resource Sharing (CORS)

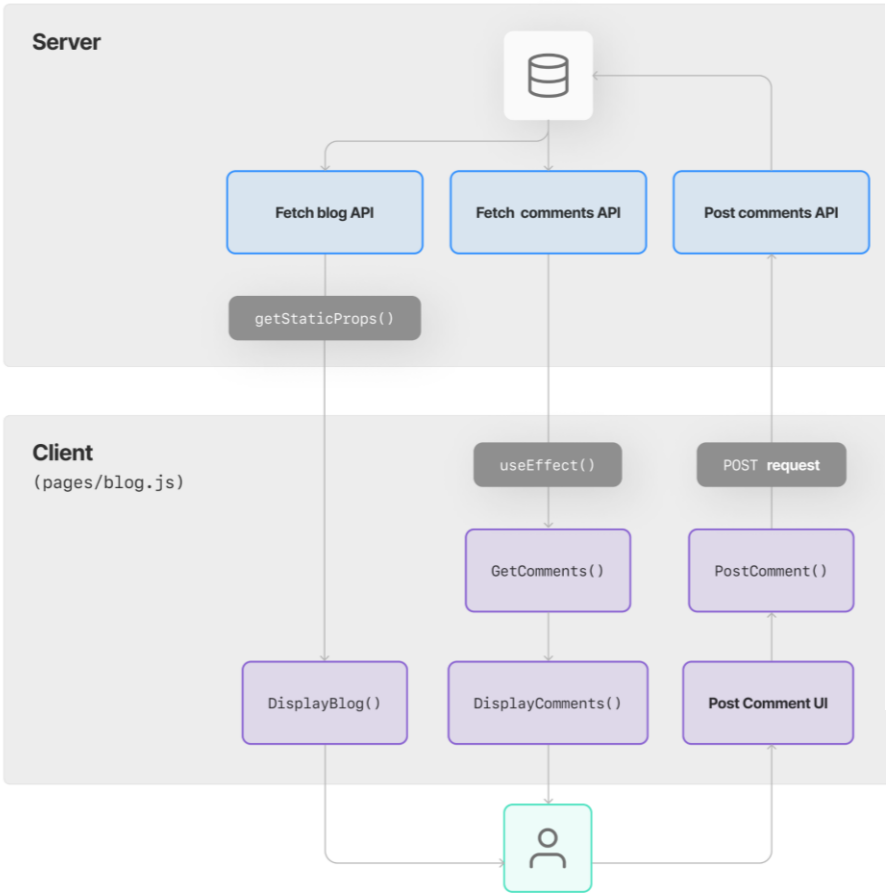


Network Boundary

- The Network Boundary is a conceptual line that separates the different environments. For example, the client and the server, or the server and the data store. the work is split into two parts: the **client module graph** and the **server module graph**. The server module graph contains all the components that are rendered on the server, and the client module graph contains all components that are rendered on the client. You can use the React "**use client**" convention to define the boundary. There's also a "**use server**" convention, which tells React to do some computational work on the server.

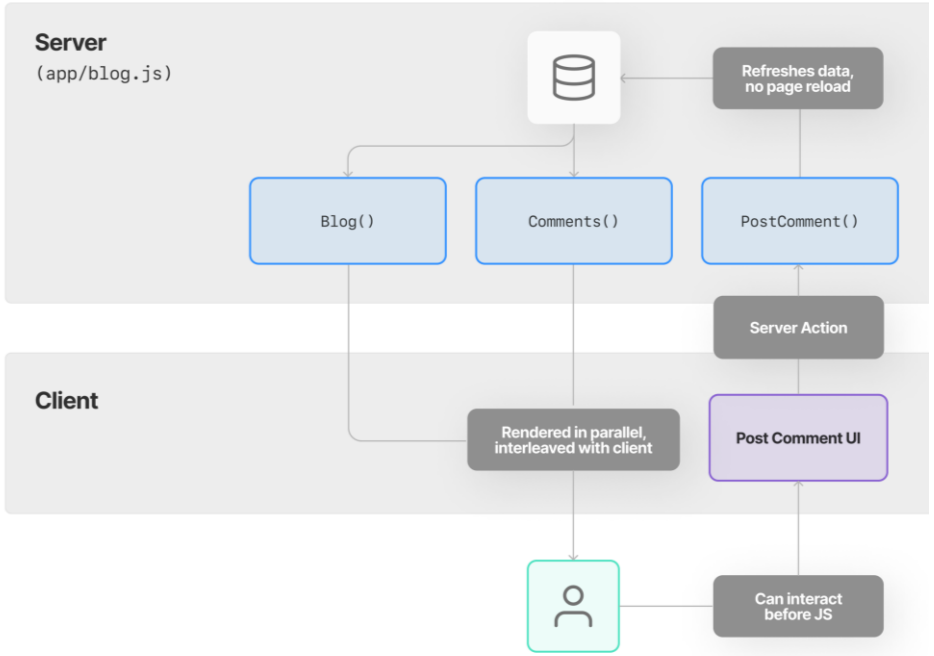
Network Boundary

Without React Server Components



Source code not include data

With React Server Components



Source code included data, SEO loves

What do you need to do?	Server Component	Client Component
Fetch data	✓	✗
Access backend resources (directly)	✓	✗
Keep sensitive information on the server (access tokens, API keys, etc)	✓	✗
Keep large dependencies on the server / Reduce client-side JavaScript	✓	✗
Add interactivity and event listeners (`onClick()`, `onChange()`, etc)	✗	✓
Use State and Lifecycle Effects (`useState()`, `useReducer()`, `useEffect()`, etc)	✗	✓
Use browser-only APIs	✗	✓
Use custom hooks that depend on state, effects, or browser-only APIs	✗	✓
Use React Class components ↗	✗	✓

Q & A